SIAM J. COMPUT.
Vol. 31, No. 1, pp. 1–17

A POLYNOMIAL TIME APPROXIMATION SCHEME FOR GENERAL MULTIPROCESSOR JOB SCHEDULING*

JIANER CHEN† AND ANTONIO MIRANDA‡

Abstract. Recently, there have been considerable interests in the multiprocessor job scheduling problem, in which a job can be processed in parallel on one of several alternative subsets of processors. In this paper, a polynomial time approximation scheme is presented for the problem in which the number of processors in the system is a fixed constant. This result is the best possible because of the strong NP-hardness of the problem and is a significant improvement over the past results: the best previous result was an approximation algorithm of ratio $7/6 + \epsilon$ for 3-processor systems based on Goemans's algorithm for a restricted version of the problem.

Key words. job scheduling, approximation algorithm, polynomial time approximation scheme, multiprocessor processing

AMS subject classifications. 68Q20, 68Q25, 90B35, 90C27, 90C39

PII. S0097539798348110

1. Introduction.

One of the assumptions made in classical scheduling theory is that a job is always executed by one processor at a time. With advances in parallel algorithms, this assumption may no longer be valid for job systems. For example, in semiconductor circuit design workforce planning, a design project is to be processed by a group of people. The project contains n jobs, and each job can be worked on by one of a set of alternatives, where each alternative consists of one or more persons in the group working simultaneously on the particular job. The processing time of each job depends on the subgroup of people being assigned to handle the job. Note that the same person may belong to several different subgroups. Now the question is how we can schedule the jobs so that the project can be finished as early as possible. Other applications include (i) the berth allocation problem [23], where a large vessel may occupy several berths for loading and unloading, (ii) diagnosable microprocessor systems [22], where a job must be performed on parallel processors in order to detect faults, (iii) manufacturing, where a job may need machines, tools, and people simultaneously (this gives an example for a system in which processors may have different types), and (iv) scheduling a sequence of meetings where each meeting requires a certain group of people [11]. In the scheduling literature [17], these kinds of problems are called *multiprocessor job scheduling* problems.

Among the others, two types of multiprocessor job scheduling problems have been extensively studied [7, 24]. The first type is the $P_m|fix|C_{\max}$ problem, in which the subset of processors and the processing time for parallel processing each job are fixed. The second type is a more general version, the $P_m|set|C_{\max}$ problem, in which each job may have a number of alternative processing modes and each processing

*Received by the editors December 2, 1998; accepted for publication (in revised form) December 22, 2000; published electronically May 31, 2001. A preliminary version of this paper appeared in *Proceedings of the* 31*st Annual ACM Symposium on Theory of Computing* (STOC'99), Atlanta, 1999, pp. 418–427.

http://www.siam.org/journals/sicomp/31-1/34811.html

†Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (chen@cs.tamu.edu). This author was supported in part by the National Science Foundation under grants CCR-9613805 and CCR-0000206.

‡Department of Computer Science, Bucknell University, Lewisburg, PA 17837 (amiranda@eg.bucknell.edu).

mode specifies a subset of processors and the job processing time on that particular processor subset. The objective for both problems is to construct a scheduling of minimum makespan on the m-processor system for a given list of jobs. The jobs are supposed to be nonpreemptive.

Approximability of the multiprocessor job scheduling problems has been studied. The $P_2|set|C_{\max}$ problem is a generalized version of the classical job scheduling problem on a 2-processor system [13]; thus it is NP-hard. Hoogeveen, van de Velde, and Veltman [18] showed that the $P_3|fix|C_{\max}$ problem (thus also the $P_3|set|C_{\max}$ problem) is NP-hard in the strong sense; thus it does not have a fully polynomial time approximation scheme unless P = NP (see also [4, 5]). Blazewicz et al. [4] developed a polynomial time approximation algorithm of ratio 4/3 for the problem $P_3|fix|C_{\max}$, which was improved later by Dell'Olmo, Speranza, and Tuza [10], who gave a polynomial time approximation algorithm of ratio 5/4 for the same problem. Both algorithms are based on the study of a special type of schedulings called *normal schedulings*. Goemans [14] further improved the algorithms by giving a polynomial time approximation algorithm of ratio 7/6 for the $P_3|fix|C_{\max}$ problem. More recently, Amoura et al. [1] developed a polynomial time approximation scheme for the problem $P_m|fix|C_{\max}$ for every fixed integer m.

Approximation algorithms for the $P_m|set|C_{\max}$ problem were not as successful as that for the $P_m|fix|C_{\max}$ problem. Bianco et al. [3] presented a polynomial time approximation algorithm for the $P_m|set|C_{\max}$ problem whose approximation ratio is bounded by m. Chen and Lee [8] improved their algorithm by giving a polynomial time approximation algorithm for the $P_m|set|C_{\max}$ problem with an approximation ratio $m/2 + \epsilon$. Miranda [25] showed that the problem $P_3|set|C_{\max}$ can be approximated in polynomial time with a ratio $7/6 + \epsilon$. Before the present paper, it was unknown whether there is a polynomial time approximation algorithm with ratio c for the problem $P_m|set|C_{\max}$, where c is a constant independent of the number m of processors in the system.

In this paper, we present a polynomial time approximation scheme for the problem $P_m|set|C_{\max}$. Our algorithm combines the techniques developed by Amoura et al. [1], who split jobs into large jobs and small jobs, and the techniques developed by Dell'Olmo, Speranza, and Tuza [10] and Goemans [14] on normal schedulings, plus the standard dynamic programming and scaling techniques. More precisely, based on a classification of large jobs and small jobs, we introduce the concept of (m, ϵ)-*canonical schedulings*, which can be regarded as a generalization of the normal schedulings. We show that for any job list, there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan. Then we show how this (m, ϵ)-canonical scheduling can be approximated. Combining these two steps gives us a polynomial time approximation scheme for the $P_m|set|C_{\max}$ problem.

Our result is the best possible in the following sense: because the problem $P_m|set|C_{\max}$ is NP-hard in the strong sense, it is unlikely that our algorithm can be further improved to a fully polynomial time approximation scheme [13]. Moreover, the polynomial time approximation scheme cannot be extended to the more general problem $P|set|C_{\max}$, in which the number m of processors in the system is given as a parameter in the input: it can be shown that there is a constant $\delta > 0$ such that the problem $P|set|C_{\max}$ has no polynomial time approximation algorithms whose approximation ratio is bounded by n^δ [25].

The paper is organized as follows. Section 2 gives necessary background and preliminaries for the problem. In section 3 we introduce (m, ϵ)-canonical schedulings

and study their properties. Section 4 presents the polynomial time approximation scheme for the problem $P_m|set|C_{\max}$ and section 5 concludes with some remarks and further research directions.

2. Preliminaries. We assume readers' familiarity with the basic concepts in approximation theory [13], such as approximation algorithms, approximation ratios, polynomial time approximation schemes, and fully polynomial time approximation schemes.

The $P_m|set|C_{\max}$ problem is a scheduling problem minimizing the makespan for a set of jobs, each of which may have several alternative processing modes. More formally, an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is a list of jobs: $\{J_1, J_2, \ldots, J_n\}$, where each job J_i is associated with a list of alternative *processing modes*: $J_i = [M_{i1}, \ldots, M_{ip_i}]$. Each processing mode (or simply *mode*) M_{ij} is specified by a pair (Q_{ij}, t_{ij}), where Q_{ij} is a subset of processors in the m-processor system and t_{ij} is an integer indicating the parallel processing time of the job J_i on the processor set Q_{ij}. In case there is no ambiguity, we also say that the processor set Q_{ij} is a mode for the job J_i. For each job $J_i = [M_{i1}, \ldots, M_{ip_i}]$, where $M_{ij} = (Q_{ij}, t_{ij})$, we let $\min_i$ be the minimum t_{ij} over all j, $1 \le j \le p_i$. The value $\min_i$ will be called the *minimum parallel processing time* for the job J_i.

Given a list $\mathcal{J} = \{J_1, \ldots, J_n\}$ of jobs, a *scheduling* $\Gamma(\mathcal{J})$ of $\mathcal{J}$ on the m-processor system consists of two parts: (1) determination of a processing mode for each job J_i in $\mathcal{J}$ and (2) determination of the starting execution time for each job under the assigned mode so that at any moment, each processor in the system is used for (maybe parallel) processing at most one job (assuming that the system starts at time $\tau = 0$). The *makespan* of the scheduling $\Gamma(\mathcal{J})$ is the latest finishing time of a job in $\mathcal{J}$ under the scheduling $\Gamma(\mathcal{J})$. Let $Opt(\mathcal{J})$ denote the minimum makespan over all schedulings for $\mathcal{J}$. The $P_m|set|C_{\max}$ problem is for a given instance $\mathcal{J}$ to construct a scheduling of makespan $Opt(\mathcal{J})$ for $\mathcal{J}$.

Let P_m be the set of the m processors in the m-processor system. A collection $\{P'_1, \ldots, P'_k\}$ of k nonempty subsets of P_m is a *k-partition* of P_m if $P_m = \bigcup_{i=1}^{k} P'_i$ and $P'_i \cap P'_j = \emptyset$ for all $i \ne j$. A collection of subsets of P_m is a *partition* of P_m if it is a k-partition for some integer $k \ge 1$. The total number B_m of different partitions of the set P_m is called the mth *Bell number* [16]. It can be proved easily by induction that $B_m \le m!$.

Another combinatorial fact we need for analysis of our scheduling algorithm is the "cut-index" in a nonincreasing sequence of integers.

LEMMA 2.1. *Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a nonincreasing sequence of integers, let $m \ge 2$ be a fixed integer, and let $\epsilon > 0$ be an arbitrary real number. Then there is an index j_0 (with respect to m and ϵ) such that*

(1) *$j_0 = (3mB_m + 1)^k$, where $k \le \lfloor m/\epsilon \rfloor$ is an integer and*
(2) *for any subset $\mathcal{T}'$ of at most $3j_0mB_m$ integers t_q in $\mathcal{T}$ with $q > j_0$, we have*

$$\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m) \sum_{i=1}^{n} t_i.$$

Proof. To simplify expressions, let $b_m = 3mB_m + 1$. Decompose the sum $t_1 + t_2 + \cdots + t_n$ into subsums

$$A_1 = t_1 + \cdots + t_{b_m},$$
$$A_2 = t_{b_m+1} + \cdots + t_{b_m^2},$$

$$\begin{aligned}
&\cdots\cdots\\
A_j &= t_{b_m^{j-1}+1} + \cdots + t_{b_m^j},\\
&\cdots\cdots\\
A_h &= t_{b_m^{h-1}+1} + \cdots + t_n,
\end{aligned}$$

where $h = \lceil \log n / \log b_m \rceil$.

Since $\sum_{j=1}^{h} A_j = \sum_{i=1}^{n} t_i$, there are at most $\lfloor m/\epsilon \rfloor$ subsums A_j larger than $(\epsilon/m)\sum_{i=1}^{n} t_i$. Let A_{k+1} be the first subsum such that $A_{k+1} \le (\epsilon/m)\sum_{i=1}^{n} t_i$; then $k \le \lfloor m/\epsilon \rfloor$.

Let $j_0 = b_m^k = (3mB_m + 1)^k$. Since the sum of the first $b_m^{k+1} - b_m^k = 3j_0 mB_m$ integers t_q in $\mathcal{T}$ with $q > j_0 = b_m^k$ is bounded by $(\epsilon/m)\sum_{i=1}^{n} t_i$,

$$A_{k+1} = t_{b_m^k+1} + \cdots + t_{b_m^{k+1}} \le (\epsilon/m)\sum_{i=1}^{n} t_i,$$

and the sequence $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is nonincreasing, we conclude that for any subset $\mathcal{T}'$ of $\mathcal{T}$ of at most $3j_0 mB_m$ integers t_q with $q > j_0$, we must have

$$\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m)\sum_{i=1}^{n} t_i.$$

This completes the proof. □

For the nonincreasing sequence $\mathcal{T}$ of integers, we will denote by $j_{m,\epsilon}$ the smallest index that satisfies conditions (1) and (2) in Lemma 2.1. The index $j_{m,\epsilon}$ will be called the *cut-index* for the sequence $\mathcal{T}$.

3. On (m, ϵ)-canonical schedulings. In this section, we first assume that the mode assignment for each job in the instance $\mathcal{J}$ is decided and discuss how we schedule the jobs in $\mathcal{J}$ under the mode assignment to the processor set P_m. By this assumption, the job list $\mathcal{J}$ is actually an instance for the $P_m|fix|C_{\max}$ problem (recall that the $P_m|fix|C_{\max}$ problem is the problem $P_m|set|C_{\max}$ with the restriction that every job in an instance has only one processing mode).

Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ be an instance for the $P_m|fix|C_{\max}$ problem, where each job J_i requires a fixed set Q_i of processors for parallel execution with processing time t_i for $i = 1, 2, \ldots, n$. Without loss of generality, assume that the processing time sequence $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is nonincreasing.

For the fixed number m of processors in the system and for an arbitrarily given real number $\epsilon > 0$, let $j_{m,\epsilon}$ be the cut-index for the sequence $\mathcal{T}$ as defined in Lemma 2.1. That is, $j_{m,\epsilon} = (3mB_m + 1)^k$, where k is an integer bounded by $\lfloor m/\epsilon \rfloor$, and for any subset $\mathcal{T}'$ of at most $3j_{m,\epsilon} mB_m$ integers t_q in $\mathcal{T}$ with $q > j_{m,\epsilon}$, we have $\sum_{t_q \in \mathcal{T}'} t_q \le (\epsilon/m)\sum_{i=1}^{n} t_i$. We split the job set $\mathcal{J}$ into two subsets

$$\mathcal{J}_L = \{J_i \mid i \le j_{m,\epsilon}\}, \qquad \mathcal{J}_S = \{J_i \mid i > j_{m,\epsilon}\}. \tag{3.1}$$

The jobs in $\mathcal{J}_L$ will be called *large jobs* and the jobs in $\mathcal{J}_S$ will be called *small jobs*.

Let $\Gamma(\mathcal{J})$ be a scheduling for the job set $\mathcal{J}$. Consider the nondecreasing sequence $\{\tau_1, \tau_2, \ldots, \tau_h\}$ of integers, where $\tau_1 = 0$, $\tau_h = +\infty$, $h = 2j_{m,\epsilon} + 2$, and τ_i, $1 < i < h$, are the starting or finishing times of the $j_{m,\epsilon}$ large jobs in $\Gamma(\mathcal{J})$. A *small job block* χ in $\Gamma(\mathcal{J})$ consists of a subset $P' \subseteq P_m$ of processors and a time interval $[\tau_p, \tau_{p+1}]$,

$1 \le p \le h-1$, such that the subset $P_m - P'$ of processors are exactly those that are executing large jobs in the time interval $[\tau_p, \tau_{p+1}]$. The value $\tau_{p+1} - \tau_p$ will be called the *height* and the processor set P' will be called the *type* of the small job block χ.

Therefore, the subset P' of processors associated with the small job block χ are those processors that are either idle or used for executing small jobs in the time interval $[\tau_p, \tau_{p+1}]$. Note that the small job block χ can be of height 0 when $\tau_p = \tau_{p+1}$. The small job block of time interval $[\tau_{h-1}, +\infty]$, where τ_{h-1} is the latest finish time of a large job, will be called the "last small job block." Note that the last small job block has type P_m.

Let χ be a small job block associated with a processor set P' and a time interval $[\tau_p, \tau_{p+1}]$. The small job block at any time moment τ in the time interval $[\tau_p, \tau_{p+1}]$ can be characterized uniquely as a collection $[Q_1, \ldots, Q_s]$ of pairwise disjoint subsets of the processor set P' such that at the time τ, for $i = 1, \ldots, s$, all processors in the subset Q_i are used for parallel execution on the same small job (thus, the subset $P' - \bigcup_{i=1}^{s} Q_i$ is the subset of idle processors at time τ). The collection $[Q_1, \ldots, Q_s]$ will be called the *type* of the time moment τ. A *layer* in the small job block χ is a maximal time interval $[\tau', \tau'']$ in $[\tau_p, \tau_{p+1}]$ such that all time moments τ between τ' and τ'' are of the same type. The *type* of the layer is equal to the type of any time moment in the layer and the *height* of the layer is $\tau'' - \tau'$.

Let L_1 and L_2 be two layers in the small job block χ of types $[Q_1, \ldots, Q_s]$ and $[R_1, \ldots, R_t]$, respectively. We say that layer L_1 *covers* layer L_2 if $\{R_1, \ldots, R_t\} \subseteq \{Q_1, \ldots, Q_s\}$. In particular, if L_1 and L_2 are two consecutive layers in the small job block χ such that layer L_2 starts right after layer L_1 finishes and L_1 covers L_2, then layer L_2 is actually a continuation of the layer L_1 with some of the small jobs finished.

DEFINITION 3.1. *A* floor σ *in the small job block* χ *is a sequence* $\{L_1, L_2, \ldots, L_z\}$ *of consecutive layers such that* (1) *for* $i = 2, \ldots, z$, *layer* L_i *starts right after layer* L_{i-1} *finishes, and layer* L_{i-1} *covers layer* L_i*; and* (2) *all small jobs interlacing layer* L_1 *start in layer* L_1 *and all small jobs interlacing layer* L_z *finish in layer* L_z.

An example of a floor is given in Figure 3.1(a). Note that a small job block may not have any nonempty floor at all, as shown in Figure 3.1(b).

Remark 1. There are a few important properties of floors in a small job block. Suppose that the layer L_1 starts at time τ' while layer L_z finishes at time τ''. Then by property (2) in the definition, no small jobs cross the floor boundaries τ' and τ''. Therefore, the floor σ can be regarded as a single job that uses the processor set P', starts at time τ', and finishes at time τ''. The *height* of the floor σ is defined to be $\tau'' - \tau'$, which is equal to the sum of the heights of the layers L_1, ..., L_z. Second, since all floors in the small job block χ are for the same processor subset P' and there are no small jobs crossing the starting and finishing times of any floors, the floors in the same small job block χ can be rearranged in any order but can still fit into the small job block without exceeding the height of the small job block. Finally, property (1) in the definition ensures that no matter how the small jobs in a floor are rearranged, a simple greedy algorithm is sufficient to refit the small jobs into the floor without exceeding the floor height. The greedy algorithm is based on the idea of the well-known Graham's list scheduling algorithm for the classical job scheduling problem [15].

DEFINITION 3.2. *Let* $\mathcal{J}$ *be an instance of the problem* $P_m|fix|C_{\max}$ *and let* π *be any permutation of the jobs in* $\mathcal{J}$. *The* list scheduling algorithm *based on the ordering* π *is to schedule each job* J_i *of mode* Q_i *in* $\mathcal{J}$, *following the ordering of* π, *at the earliest time when the processor subset* Q_i *becomes available.*

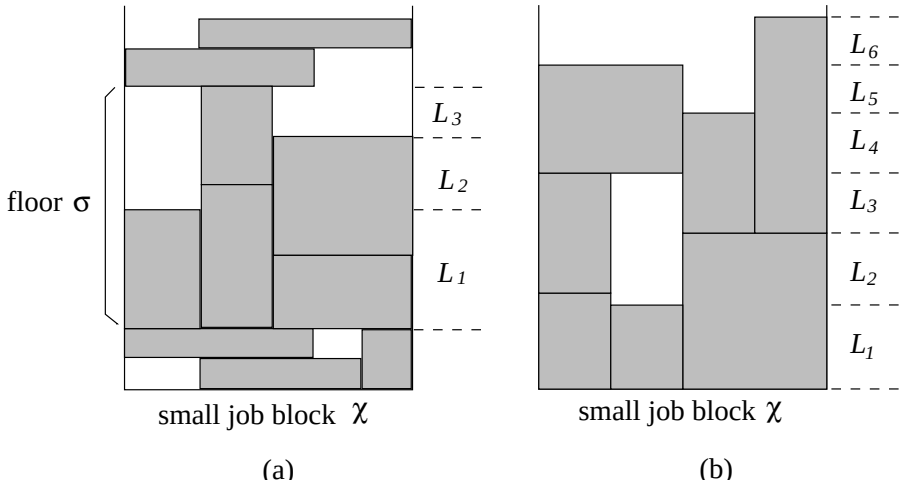


FIG. 3.1. (a) *a floor* $\{L_1, L_2, L_3\}$; (b) *a small job block with no floor.*

.

LEMMA 3.3. *Let $\mathcal{J}_\sigma$ be the set of small jobs in the floor σ. The list scheduling algorithm based on any ordering π of the jobs in $\mathcal{J}_\sigma$ will always reconstruct the floor σ.*

Proof. Suppose that the first layer in the floor σ is of type $[Q_1, \ldots, Q_s]$. Every job in $\mathcal{J}_\sigma$ must have a mode Q_i for some i, and no processor subset Q_i can become idle before its final completion time. The jobs of each mode Q_i in $\mathcal{J}_\sigma$ can be executed by the processor subset Q_i in any order without changing the completion time of Q_i. Since the list scheduling algorithm starts each job at its earliest possible time (thus no subset Q_i can become idle before its final completion time), the completion time for each subset Q_i will not be changed. Therefore, the list scheduling algorithm will construct a floor with exactly the same layers. □

DEFINITION 3.4. *Let $[Q_1, \ldots, Q_s]$ be a partition of the processor subset P'. We say that* we can assign the type $[Q_1, \ldots, Q_s]$ *to a floor $\sigma = \{L_1, \ldots, L_z\}$ if the type of the layer L_1 is a subcollection of $\{Q_1, \ldots, Q_s\}$.*

It is possible that several different types can be assigned to the same floor as long as the type of the floor is a subcollection of the assigned floor types. For example, let $[Q_1, \ldots, Q_s]$ be a partition of the processor subset P'. If the first layer L_1 in a floor σ is of type $[Q_3, \ldots, Q_s]$, then we can assign either type $[Q_1, Q_2, Q_3, \ldots, Q_s]$ or type $[Q_1 \cup Q_2, Q_3, \ldots, Q_s]$ to the floor σ.

DEFINITION 3.5. *A small job block χ is a* tower *if it is constituted by a sequence of floors such that we can assign types to the floors so that no two floors in the tower χ are of the same type.*

Note that since each floor type is a partition of the processor subset P', a tower contains at most B_m floors, where $B_m \leq m!$, the mth Bell number, is the number of different partitions of a set of m elements.

In our discussion, we will be concentrating on schedulings of a special form in the following sense.

DEFINITION 3.6. *Let $\mathcal{J}$ be an instance of the problem $P_m|fix|C_{\max}$, which is divided into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$ as given in* (3.1) *for a fixed integer $m > 2$ and a fixed constant $\epsilon > 0$. A scheduling $\Gamma(\mathcal{J})$ of $\mathcal{J}$ is* (m, ϵ)-canonical *if every*

small job block in $\Gamma(\mathcal{J})$ *is a tower.*

Remark 2. Note that in an (m, ϵ)-canonical scheduling, no small jobs cross the boundary of a tower. Therefore, a tower of height t and associated with a processor set Q can be simply regarded as a job of mode (Q, t).

We first show that an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of $\mathcal{J}$ can be constructed by the list scheduling algorithm when large jobs and towers in $\Gamma(\mathcal{J})$ are given in a proper order.

LEMMA 3.7. *Let* $\Gamma(\mathcal{J})$ *be an* (m, ϵ)*-canonical scheduling for the job set* $\mathcal{J}$*. Let* π *be the sequence of the large jobs and towers in* $\Gamma(\mathcal{J})$*, ordered in terms of their starting times in* $\Gamma(\mathcal{J})$*. Then the list scheduling algorithm based on the ordering* π*, which regards each tower as a single job, constructs a scheduling of* $\mathcal{J}$ *with makespan not larger than that of* $\Gamma(\mathcal{J})$*.*

Proof. Let $\mathcal{J}_i = \{J_1, \ldots, J_i\}$ be any prefix of the ordered sequence π, where each J_j is either a large job or a tower. Let $\Gamma(\mathcal{J}_i)$ be the scheduling of $\mathcal{J}_i$ obtained from $\Gamma(\mathcal{J})$ by removing all large jobs and towers that are not in $\mathcal{J}_i$ and let $\Gamma'(\mathcal{J}_i)$ be the scheduling by the list scheduling algorithm on the jobs in $\mathcal{J}_i$. By induction, it is not difficult to prove that the completion time of *each* processor in $\Gamma'(\mathcal{J}_i)$ is not larger than the completion time of the same processor in $\Gamma(\mathcal{J}_i)$. For $\mathcal{J}_i = \mathcal{J}$, this implies that the makespan of the scheduling constructed by the list scheduling algorithm based on the ordering π is not larger than the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. □

Thus, once the ordering of large jobs and towers is decided, it is easy to construct a scheduling that is not worse than the given (m, ϵ)-canonical scheduling. In the following, we will prove that for any instance $\mathcal{J}$ for the problem $P_m|fix|C_{\max}$, there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan.

THEOREM 3.8. *Let* $\mathcal{J}$ *be an instance for the problem* $P_m|fix|C_{\max}$*. Then for any* $\epsilon > 0$*, there is an* (m, ϵ)*-canonical scheduling* $\Gamma(\mathcal{J})$ *of* $\mathcal{J}$ *such that the makespan of* $\Gamma(\mathcal{J})$ *is bounded by* $(1 + \epsilon)Opt(\mathcal{J})$*.*

Proof. Let $\Gamma_1(\mathcal{J})$ be an optimal scheduling of makespan $Opt(\mathcal{J})$ for $\mathcal{J}$. We construct an (m, ϵ)-canonical scheduling for $\mathcal{J}$ based on the optimal scheduling $\Gamma_1(\mathcal{J})$. Let $\mathcal{J}_L$ and $\mathcal{J}_S$ be the set of large jobs and the set of small jobs in $\mathcal{J}$, respectively, according to the definition in (3.1). Consider a small job block χ in the scheduling $\Gamma_1(\mathcal{J})$.

Assume that the small job block χ is associated with a processor set P' of r processors, $r \leq m$, and a time interval $[\tau_p, \tau_{p+1}]$. Let $[T_1, \ldots, T_y]$ be the list of all partitions of the processor set P', where $y = B_r \leq B_m$. We divide the layers in the small job block χ into groups, each corresponding to a partition of P', as follows. A layer of type T' is put in the group corresponding to a partition T_j if T' is a subcollection of T_j. Note that a layer type T' may be a subcollection of more than one partition of P'. In this case, we put the layer arbitrarily into one and only one of the groups to ensure that each layer belongs to only one group.

For each partition T_j of P', we construct a *floor frame* σ_j whose type is T_j and height is equal to the sum of heights of all layers belonging to the group corresponding to the partition T_j. Note that so far we have not yet actually assigned any small jobs to any floor frames σ_1, …, σ_y. Moreover, since each layer belongs to exactly one of the groups, it is easy to see that the sum $\sum_{j=1}^{y} height(\sigma_j)$ of the heights of the floor frames σ_1, …, σ_y is equal to the sum of the heights of all layers in the small job block χ, which is equal to the height of the small job block χ.

The construction for the floor frames for the last small job block in $\Gamma_1(\mathcal{J})$ is

slightly different: we group layers only in which not all processors are idle. Thus, the sum of the heights of all floor frames in the last small job block is equal to $Opt(\mathcal{J})-\tau_0$, where τ_0 is the latest finish time for some large job in the scheduling $\Gamma_1(\mathcal{J})$.

After the construction of the floor frames for each small job block in the scheduling $\Gamma_1(\mathcal{J})$, we assign the small jobs in $\mathcal{J}_S$ to the floor frames using the following greedy method. For each small job J that requires a parallel processing by a processor subset Q, we assign J to an arbitrary floor frame σ in a small job block as long as the floor frame σ satisfies the following conditions: (1) the type of the floor frame σ contains the subset Q and (2) adding the job J to σ does not exceed the height of the floor frame σ (if there are more than one floor frames satisfying these conditions, arbitrarily pick one of them). Note that we assign a job to a floor frame only when the mode of the job is contained in the type of the floor frame. Therefore, this assignment will never leave a "gap" between two jobs in the same floor frame.

The above assignment of small jobs in $\mathcal{J}_S$ to floor frames stops when none of the small jobs left in $\mathcal{J}_S$ can be assigned to any of the floor frames according to the above rules. Now each floor frame becomes a floor.

For each small job block χ in $\Gamma_1(\mathcal{J})$, let S_χ be the set of floor frames in χ. Since the height of a resulting floor is not larger than the height of the corresponding floor frame, the sum of the heights of the floors resulting from the floor frames in S_χ is not larger than the height of the small job block χ. Therefore, we can put all these floors into the small job block χ (in an arbitrary order) to make χ a tower. Doing this for all small job blocks in $\Gamma_1(\mathcal{J})$ gives an (m,ϵ)-canonical scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ for the job set $\mathcal{J}_L \cup \mathcal{J}'_S$, where $\mathcal{J}'_S$ is the set of small jobs that have been assigned to the floor frames in the above procedure. The makespan of the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ is bounded by $Opt(\mathcal{J})$. Now the only thing left is that we still need to schedule the small jobs that have not been assigned to any floor frames. Let $\mathcal{J}''_S = \mathcal{J}_S - \mathcal{J}'_S$ be the set of small jobs that are not assigned to any floor frames by the above procedure. We want to demonstrate that there are not many jobs in the set $\mathcal{J}''_S$.

By the definition, the number of small job blocks in the scheduling $\Gamma_1(\mathcal{J})$ is $2j_{m,\epsilon}+1 \le 3j_{m,\epsilon}$. Since each small job block is associated with at most m processors, the number of floor frames constructed in each small job block is bounded by B_m. Therefore, the total number of floor frames we constructed from the scheduling $\Gamma_1(\mathcal{J})$ is bounded by $3B_m j_{m,\epsilon}$. Moreover, each floor type is a collection of at most m processor subsets.

If the set $\mathcal{J}''_S$ contains more than $3mB_m j_{m,\epsilon}$ small jobs, then there must be a subset Q of processors such that the number of small jobs of mode Q in $\mathcal{J}''_S$ is larger than the number of the constructed floor frames whose type contains the subset Q. Let $\{\sigma_1, \ldots, \sigma_d\}$ be the set of floor frames whose type contains the subset Q.

By our assignment rules, assigning any job of mode Q in $\mathcal{J}''_S$ to a floor frame in $\{\sigma_1, \ldots, \sigma_d\}$ would exceed the height of the corresponding floor frame. Since there are more than d small jobs of mode Q in $\mathcal{J}''_S$, the sum of processing times of all small jobs of mode Q in $\mathcal{J}_S$ is larger than $\sum_{i=1}^{d} height(\sigma_i)$. On the other hand, by our construction of the floor frames in each small job block χ, the sum of the heights of the floor frames in χ whose type contains Q should not be smaller than the sum of the heights of the layers in χ whose type contains Q. Summarizing this over all small job blocks, we conclude that the sum $\sum_{i=1}^{d} height(\sigma_i)$ is not smaller than the sum of processing times of all small jobs of mode Q in $\mathcal{J}_S$ (since each small job of mode Q must be contained in consecutive layers whose type contains Q). This derives a contradiction. The contradiction shows that there are at most $3mB_m j_{m,\epsilon}$ small jobs

in the set $\mathcal{J}''_S$.

Now we assign the small jobs in $\mathcal{J}''_S$ to the floor frames in the last small job block in the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$. For each small job J of mode Q in $\mathcal{J}''_S$, we arbitrarily assign J to a floor frame whose type contains Q in the last small job block, even if this assignment exceeds the height of the floor frame. Note that the last small job block is associated with the whole processor set P_m, so for any mode Q, there must be a floor frame in the last small job block whose type contains the processor subset Q. This procedure stops with all small jobs in $\mathcal{J}''_S$ assigned to floor frames in the last small job block. It is easy to see that the resulting scheduling is an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of the original job set $\mathcal{J}$. Moreover, since the makespan of the scheduling $\Gamma_2(\mathcal{J}_L \cup \mathcal{J}'_S)$ is bounded by $Opt(\mathcal{J})$, the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ is bounded by

$$Opt(\mathcal{J}) + \sum_{J \in \mathcal{J}''_S} t(J),$$

where $t(J)$ is the parallel processing time of the small job J. Since there are at most $3mB_m j_{m,\epsilon}$ small jobs in the set $\mathcal{J}''_S$, by Lemma 2.1,

$$\sum_{J \in \mathcal{J}''_S} t(J) \le (\epsilon/m) \sum_{i=1}^{n} t_i.$$

It is easy to see that $Opt(\mathcal{J}) \ge (\sum_{i=1}^{n} t_i)/m$. Therefore, the makespan of the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ is bounded by $(1+\epsilon)Opt(\mathcal{J})$. This completes the proof of the theorem. □

Before we close this section, we introduce one more definition.

DEFINITION 3.9. *Let σ be a floor of type $[Q_1, \ldots, Q_s]$ and height l, where Q_1, ..., Q_s are pairwise disjoint subsets of processors in the processor set P_m. Then each subset Q_i plus the height l is called a* room of type Q_i *in the floor σ.*

4. The approximation scheme. Now we come back to the original problem $P_m|set|C_{\max}$. Recall that an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is a set of jobs $\{J_1, J_2, \ldots, J_n\}$, where each job J_i is given by a list of alternative processing modes $[M_{i,1}, \ldots, M_{i,p_i}]$ in which each processing mode $M_{i,j} = (Q_{i,j}, t_{i,j})$ specifies the parallel processing time $t_{i,j}$ of the job J_i on the subset $Q_{i,j}$ of processors in the m-processor system.

In order to describe our polynomial time approximation scheme for the problem, let us first discuss why this problem is more difficult than the classical job scheduling problem.

In the classical job scheduling problem, each job is executed by one processor in the system. Therefore, the order of executions of jobs in each processor is not crucial: the running time of the processor is simply equal to the sum of the processing times of the jobs assigned to the processor. Therefore, the decision of which job should be assigned to which processor, in any order, will uniquely determine the makespan of the resulting scheduling. This makes it possible to use a dynamic programming approach that extends a scheduling for a subset of jobs to that for a larger subset.

The situation in the general multiprocessor job scheduling problem $P_m|set|C_{\max}$, on the other hand, is more complicated. In particular, the makespan of a scheduling depends not only on the assignment of processing modes to jobs but also on the *order* in which the jobs are executed. Therefore, the techniques of extending a scheduling

for a subset of jobs in the classical job scheduling problem are not directly applicable here.

Theorem 3.8 shows that there is an (m, ϵ)-canonical scheduling whose makespan is very close to the optimal makespan. Therefore, constructing a scheduling whose makespan is not larger than the makespan of a good (m, ϵ)-canonical scheduling will give a good approximation to the optimal schedulings.

Nice properties of an (m, ϵ)-canonical scheduling are that within the same tower, the order of the floors does not affect the height of the tower and that within the same floor, the order of the small jobs does not affect the height of the floor (see Remarks 1 and 2 in the previous section). Therefore, the only factor that affects the heights of towers and floors is the assignments of jobs to towers and floors. This makes it become possible, at least for small jobs, to apply the techniques in classical job scheduling problems to our current problem. This is described as follows.

First, suppose that we can somehow divide the job set $\mathcal{J}$ into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$. Let us start with an (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ of the job set $\mathcal{J}$. The scheduling $\Gamma(\mathcal{J})$ gives a nondecreasing sequence $\{\tau_0, \tau_1, \ldots, \tau_{p+1}\}$ of integers, where $\tau_0 = 0$, $\tau_{p+1} = +\infty$, $p = 2j_{m,\epsilon}$, and τ_i, $0 < i < p + 1$, are the starting or finishing times of the $j_{m,\epsilon}$ large jobs in $\mathcal{J}_L$. Let the $p + 1$ corresponding towers be $\{\chi_0, \chi_1, \ldots, \chi_p\}$, where the tower χ_j consists of a subset P'_j of processors and the time interval $[\tau_j, \tau_{j+1}]$.

We suppose that the subset P'_j of processors associated with each tower χ_j is known and that the large jobs and towers of the scheduling $\Gamma(\mathcal{J})$ are ordered into a sequence π in terms of their starting times. However, we assume that the assignment of small jobs to the rooms of the scheduling $\Gamma(\mathcal{J})$ is unknown. We show how this information can be recovered.

For each tower χ_j associated with the processor set P'_j, the number of floors in the tower χ_j is $q_j = B_r \leq B_m$, where r is the number of processors in the set P'_j. Let $\sigma_{j,1}$, ..., σ_{j,q_j} be the floors of all possible different types in the tower χ_j. For each floor $\sigma_{j,q}$, let $\gamma_{j,q,1}$, ..., $\gamma_{j,q,r_{jq}}$ be the rooms in the floor $\sigma_{j,q}$, where $r_{jq} \leq m$. Therefore, the configuration of the small jobs in the (m, ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ can be specified by a $((2j_{m,\epsilon} + 1)B_m m)$-tuple

$$[t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}],$$

where $t_{j,q,r}$ specifies the running time of the room $\gamma_{j,q,r}$ (for index $\{j, q, r\}$ for which the corresponding room $\gamma_{j,q,r}$ does not exist, we can simply set $t_{j,q,r} = -1$).

Suppose that an upper bound T_0 for the running time of rooms is derived; then we can use a Boolean array D of $(2j_{m,\epsilon} + 1)B_m m + 1$ dimensions to describe the configuration of a subset of small jobs in a scheduling

$$D[0..n_S;\ \underbrace{0..T_0,\ \ldots\ldots,\ 0..T_0}_{(2j_{m,\epsilon}+1)B_m m}],$$

where $n_S = n - j_{m,\epsilon}$ is the number of small jobs in $\mathcal{J}$ such that

$$D[i; t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}] = \textsc{True}$$

if and only if there is a scheduling on the first i small jobs to the floors in $\Gamma(\mathcal{J})$ such that the running time of the room $\gamma_{j,q,r}$ is $t_{j,q,r}$ (recall that the running time of a room is dependent only on the assignment of small jobs to the room and independent

of the order in which the small jobs are executed in the room). Initially, all array elements in the array $D[\cdots\cdots]$ have value FALSE.

Suppose that a configuration of a scheduling for the first $i-1$ small jobs is given:

$$D[i-1; t_{0,1,1}, \ldots, t_{j,q,r}, \ldots, t_{2j_{m,\epsilon},B_m,m}] = \text{TRUE}. \tag{4.1}$$

We say that the ith small job J_i' under mode Q_i is *addable* to a room $\gamma_{j,q,r}$ in the configuration in (4.1) if the room $\gamma_{j,q,r}$ is of type Q_i and adding the job J_i' to the room does not exceed the upper bound T_0 of the running time of the room $\gamma_{j,q,r}$.

Now we are ready to present our dynamic programming algorithm for scheduling small jobs into the rooms in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. The algorithm is given in Figure 4.1.

Note that the algorithm SCHEDULE-SMALL may not return an (m,ϵ)-canonical scheduling for the job set $\mathcal{J}$. In fact, there is no guarantee that the height of the towers constructed in the algorithm does not exceed the height of the corresponding towers in the original (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. We first show below that the scheduling constructed by the algorithm SCHEDULE-SMALL has its makespan bounded by the makespan of the original (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.

The following lemma can be proved by induction on the index i.

LEMMA 4.1. *For all i, $0 \le i \le n_S$, the array element $D[i; \ldots, t_{j,q,r}, \ldots] =$ TRUE if and only if there is a way to assign modes to the first i small jobs and arrange them into the rooms such that the room $\gamma_{j,q,r}$ has running time $t_{j,q,r}$ for all $\{j,q,r\}$.*

Lemma 4.1 gives us directly the following corollary.

COROLLARY 4.2. *If the sequence π of large jobs and towers is ordered in terms of their starting times in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$, then the algorithm* SCHEDULE-SMALL *constructs a scheduling for the job set $\mathcal{J}$ with makespan bounded by the makespan of the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.*

Proof. Note that the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$ gives a way to assign and arrange all small jobs in $\mathcal{J}_S$ into the rooms. According to Lemma 4.1, the corresponding array element in the array D must have value TRUE:

$$D[n_S; \ldots, t_{j,q,r}, \ldots] = \text{TRUE}.$$

For this array element, step 3 of the algorithm will construct the towers that have exactly the same types and heights as their corresponding towers in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$. (This may not give exactly the same assignment of small jobs to rooms. However, the running times of the corresponding rooms must be exactly the same.) Now since the sequence π is given in the order sorted by the starting times of the large jobs and towers in the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$, by Lemma 3.7, the call in step 3 to the list scheduling algorithm based on the order π and this configuration will construct a scheduling whose makespan is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma(\mathcal{J})$.

Finally, since step 4 of the algorithm returns the scheduling of the minimum makespan constructed in step 3, we conclude that the algorithm returns a scheduling whose makespan is not larger than the makespan of $\Gamma(\mathcal{J})$. □

We analyze the algorithm SCHEDULE-SMALL.

LEMMA 4.3. *Let T_0 be the upper bound used by the algorithm* SCHEDULE-SMALL *on the running time of the rooms. Then the running time of the algorithm* SCHEDULE-SMALL *is bounded by $O(n2^m\lambda_{m,\epsilon}T_0^{\lambda_{m,\epsilon}})$, where $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$.*

Proof. The number n_S of small jobs in $\mathcal{J}_S$ is bounded by the total number n of jobs in $\mathcal{J}$; each small job may have at most $2^m - 1 \le 2^m$ different modes. Also, as we

Algorithm. SCHEDULE-SMALL
Input: The set $\mathcal{J}_S$ of small jobs and an order π of the large jobs and towers in $\Gamma(\mathcal{J})$.
Output: A scheduling for the job set $\mathcal{J}$.
1. $D[0;0,\ldots,0] = \text{TRUE}$;
2. **for** $i = 1$ **to** n_S **do**
   for each mode Q_{ij} of the small job J'_i with processing time t_{ij}
   for each $D[i-1;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$ such that the job J'_i under mode Q_{ij} is addable to the room $\gamma_{j,q,r}$
   $D[i;\ldots,t_{j,q,r}+t_{ij},\ldots] = \text{TRUE}$;
3. **for** each $D[n_S;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$
   call the list scheduling algorithm based on the order π to construct a scheduling for $\mathcal{J}$ in which the room $\gamma_{j,q,r}$ has running time $t_{j,q,r}$ for all $t_{j,q,r} \geq 0$;
4. return the scheduling constructed in step 3 with the minimum makespan.

FIG. 4.1. *Scheduling small jobs in floors.*

indicated before, the number of rooms is bounded by $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$. Since the running time for each room is bounded by T_0, for each fixed i, there cannot be more than $T_0^{\lambda_{m,\epsilon}}$ elements $D[i-1;*,\ldots,*]$. Finally, for each $D[i-1;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$, we can check each of the $\lambda_{m,\epsilon}$ component values $t_{j,q,r}$ to decide if the job J'_i under mode Q_{ij} is addable to the room $\gamma_{j,q,r}$. In conclusion, the running time of step 2 in the algorithm SCHEDULE-SMALL is bounded by

$$O(n \cdot 2^m \cdot T_0^{\lambda_{m,\epsilon}} \cdot \lambda_{m,\epsilon}).$$

We will also attach the mode assignment and room assignment of the job J'_i to each element $D[i;\ldots,t_{j,q,r},\ldots] = \text{TRUE}$. With this information, from a given configuration $D[n_S;\cdots,t_{j,q,r},\cdots] = \text{TRUE}$, a corresponding scheduling for the small jobs in the rooms can be easily constructed by backtracking the dynamic programming procedure and its makespan can be computed in time $\lambda_{m,\epsilon}$. Therefore, step 3 of the algorithm takes time

$$O(n \cdot T_0^{\lambda_{m,\epsilon}} \cdot \lambda_{m,\epsilon}).$$

In conclusion, the running time of the algorithm SCHEDULE-SMALL is bounded by $O(n2^m\lambda_{m,\epsilon}T_0^{\lambda_{m,\epsilon}})$, where $\lambda_{m,\epsilon} = (2j_{m,\epsilon}+1)B_m m$. □

We now discuss how an upper bound T_0 for the running time of rooms can be derived. Given an instance $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of the problem $P_m|set|C_{\max}$ and a positive real number $\epsilon > 0$, where each job J_i is specified by a list of alternative processing modes, $J_i = [M_{i1}, \ldots, M_{ip_i}]$ and $M_{ij} = (Q_{ij}, t_{ij})$. Recall that $\min_i = \min\{t_{ij} \mid 1 \leq j \leq p_i\}$. Then the sum $T_0 = \sum_{i=1}^{n} \min_i$ is obviously an upper bound on the makespan of the (m,ϵ)-canonical schedulings for $\mathcal{J}$. (T_0 is the makespan of a straightforward scheduling that assigns each job J_i the mode corresponding to $\min_i$, then starts each job J_i when the previous job J_{i-1} finishes. Therefore, if no (m,ϵ)-canonical scheduling has makespan better than T_0, we simply return this straightforward scheduling.) In particular, the value T_0 is an upper bound for the running time for all rooms. Moreover, since the job set $\mathcal{J}$ takes at least T_0 amount of "work" (the *work* taken by a job is equal to the parallel processing time multiplied by the number of processors involved in this processing) and the system has m processors,

the value T_0 also provides a lower bound for the optimal makespan $Opt(\mathcal{J})$:

$$Opt(\mathcal{J}) \geq T_0/m.$$

In order to apply algorithm SCHEDULE-SMALL, we first need to decide how the set $\mathcal{J}$ is split into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$, what are the modes for the large jobs, what are the types for the towers, and what is the ordering π for the large jobs and towers on which the list scheduling algorithm can be applied. According to Lemma 2.1, the number of large jobs is of form $j_{m,\epsilon} = (3mB_m + 1)^k$ for some integer $k \leq \lfloor m/\epsilon \rfloor$ and by the definition, the number of towers is $2j_{m,\epsilon} + 1$. When m and ϵ are fixed, the number of large jobs and the number of towers are both bounded by a constant. Therefore, we can use any brute force method to exhaustively try all possible cases.

To achieve a polynomial time approximation scheme for the problem $P_m|set|C_{\max}$, we combine the standard scaling techniques [20] with the concept of (m, ϵ)-canonical schedulings as follows.

Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ be an instance of the $P_m|set|C_{\max}$ problem, where $J_i = [M_{i1}, \ldots, M_{ip_i}]$ and $M_{ij} = (Q_{ij}, t_{ij})$. We let $K = \epsilon \cdot T_0/(nm)$ and construct another instance $\mathcal{J}' = \{J'_1, \ldots, J'_n\}$ for the problem, where $J'_i = [M'_{i1}, \ldots, M'_{ip_i}]$ and $M'_{ij} = (Q_{ij}, \lfloor t_{ij}/K \rfloor)$. In other words, the jobs in $\mathcal{J}'$ are identical to those in $\mathcal{J}$ except that all processing times t_{ij} are replaced by $\lfloor t_{ij}/K \rfloor$. We say that the job set $\mathcal{J}'$ is obtained from the job set $\mathcal{J}$ by *scaling the processing times by* K. We apply the algorithm described above to the instance $\mathcal{J}'$ to construct a scheduling for $\mathcal{J}'$ from which a scheduling for $\mathcal{J}$ is induced. The formal algorithm is presented in Figure 4.2.

We explain how step 5 converts the scheduling $\Gamma_0(\mathcal{J}')$ for the job set $\mathcal{J}'$ into a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$. We first multiply the processing time and the starting time of each job J'_i in the scheduling $\Gamma_0(\mathcal{J}')$ by K (but keeping the processing mode). That is, for the job J'_i of mode Q_{ij} and processing time $\lfloor t_{ij}/K \rfloor$ that starts at time τ_i in $\Gamma_0(\mathcal{J}')$, we replace it by a job J''_i of mode Q_{ij} and processing time $K \cdot \lfloor t_{ij}/K \rfloor$ and let it start at time $K\tau_i$. This is equivalent to proportionally "expanding" the scheduling $\Gamma_0(\mathcal{J}')$ by a factor K. Now on this expansion of the scheduling $\Gamma_0(\mathcal{J}')$, following the order in terms of their finish times, we do "correction" on processing times by increasing the processing time of each job J''_i from $K \cdot \lfloor t_{ij}/K \rfloor$ to t_{ij}. (Note that this increase in processing time may cause many jobs in the scheduling to delay their starting time by $(t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor)$ units. In particular, this increase may cause the makespan of the scheduling to increase by $(t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor)$ units.) After the corrections on the processing time for all jobs in $\mathcal{J}$, we obtain a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$.

LEMMA 4.4. *For fixed $m \geq 2$ and $\delta > 0$, the running time of the algorithm* APPROX-SCHEME *for the problem $P_m|set|C_{\max}$ is bounded by $O(n^{\lambda_{m,\epsilon}+j_{m,\epsilon}+1})$, where $\epsilon = \delta/2$, $j_{m,\epsilon} \leq (3mB_m + 1)^{\lfloor m/\epsilon \rfloor}$ and $\lambda_{m,\epsilon} = (2j_{m,\epsilon} + 1)B_m m$.*

Proof. Since the integer k is bounded by $\lfloor m/\epsilon \rfloor$, the number j_0 of large jobs in $\mathcal{J}'_L$ is bounded by $j_{m,\epsilon} \leq (3mB_m+1)^{\lfloor m/\epsilon \rfloor}$. Therefore, there are at most $\binom{n}{j_{m,\epsilon}} = O(n^{j_{m,\epsilon}})$ ways to choose the large job set $\mathcal{J}'_L$. Since each job may have up to $2^m - 1 < 2^m$ alternative mode assignments, the total number of mode assignments to each large job set $\mathcal{J}'_L$ is bounded by $(2^m)^{j_{m,\epsilon}} = 2^{mj_{m,\epsilon}}$. Each tower is associated with a subset of the processor set P_m of m processors. Thus, each tower may be associated with $2^m - 1 \leq 2^m$ different subsets of P_m. Therefore, the number of different sequences of up to $2j_{m,\epsilon}+1$ towers is bounded by $(2^m)^{2j_{m,\epsilon}+1} = 2^{2mj_{m,\epsilon}+m}$. Finally, the number of permutations of the j_0 large jobs and $2j_0+1$ towers is $(3j_0+1)!$. Summarizing all these

Algorithm. APPROX-SCHEME
Input: An instance $\mathcal{J}$ for the problem $P_m|set|C_{\max}$ and $\delta > 0$.
Output: A scheduling of $\mathcal{J}$.
1. $\epsilon = \delta/2$; $T_0 = \sum_{i=1}^{n} \min_i$; $K = \epsilon \cdot T_0/(nm)$;
2. let $\mathcal{J}'$ be the job set obtained by scaling the job set $\mathcal{J}$ by K;
3. **for** $k = 0$ **to** $\lfloor m/\epsilon \rfloor$ **do**
   $j_0 = (3mB_m + 1)^k$;
3.1. **for** each subset $\mathcal{J}'_L$ of j_0 jobs in $\mathcal{J}'$
3.2. **for** each mode assignment A to the jobs in $\mathcal{J}'_L$
3.3. **for** each possible sequence of $2j_0 + 1$ towers
3.4. **for** each ordering π of the j_0 jobs in $\mathcal{J}'_L$ and the $2j_0 + 1$ towers
$\mathcal{J}'_S = \mathcal{J}' - \mathcal{J}'_L$;
call SCHEDULE-SMALL on small job set $\mathcal{J}'_S$ and the ordering π to construct a scheduling for the job set $\mathcal{J}'$ (use $T'_0 = \lceil T_0/K \rceil$ as the upper bound for the running time of rooms);
4. let $\Gamma_0(\mathcal{J}')$ be the scheduling constructed in step 3 with the minimum makespan;
5. replace each job J'_i in $\Gamma_0(\mathcal{J}')$ by the corresponding job J_i to obtain a scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$;
6. return the job scheduling $\Gamma_0(\mathcal{J})$.

FIG. 4.2. *The approximation scheme.*

together, we conclude that the number of times that the algorithm SCHEDULE-SMALL is called is bounded by

$$O(\lfloor m/\epsilon \rfloor \cdot n^{j_{m,\epsilon}} \cdot 2^{mj_{m,\epsilon}} \cdot 2^{2mj_{m,\epsilon}+m} \cdot (3j_{m,\epsilon} + 1)!). \tag{4.2}$$

When the algorithm SCHEDULE-SMALL is applied to the job set $\mathcal{J}'_S$, the upper bound on the running time of the rooms is

$$T'_0 = \lceil T_0/K \rceil = \lceil (nm)/\epsilon \rceil \leq (nm)/\epsilon + 1.$$

According to Lemma 4.3, each call to the algorithm SCHEDULE-SMALL takes time

$$O(n2^m \lambda_{m,\epsilon}(T'_0)^{\lambda_{m,\epsilon}}) = O(n2^m \lambda_{m,\epsilon}((nm)/\epsilon + 1)^{\lambda_{m,\epsilon}}), \tag{4.3}$$

where $\lambda_{m,\epsilon} = (2j_{m,\epsilon} + 1)B_m m$.

Combining (4.2) and (4.3) and noting that m and δ thus ϵ are fixed constants, we conclude that the running time of the algorithm APPROX-SCHEME is bounded by $O(n^{\lambda_{m,\epsilon}+j_{m,\epsilon}+1})$. □

Now we are ready to present our main theorem.

THEOREM 4.5. *The algorithm* APPROX-SCHEME *is a polynomial time approximation scheme for the problem* $P_m|set|C_{\max}$.

Proof. As proved in Lemma 4.4, the algorithm APPROX-SCHEME runs in polynomial time when m and δ are fixed constants. Therefore, we need only to show that the makespan of the scheduling $\Gamma_0(\mathcal{J})$ constructed by the algorithm APPROX-SCHEME for an instance $\mathcal{J}$ of the problem $P_m|set|C_{\max}$ is at most $(1 + \delta)$ times the optimal makespan $Opt(\mathcal{J})$ for the instance $\mathcal{J}$. Again let $\epsilon = \delta/2$.

Let $\Gamma(\mathcal{J})$ be an optimal scheduling of makespan $Opt(\mathcal{J})$. Under the scheduling $\Gamma(\mathcal{J})$, the mode assignments of the jobs are fixed. Thus, this particular mode assignment makes us able to split the job set $\mathcal{J}$ into large job set $\mathcal{J}_L$ and small job set $\mathcal{J}_S$ in terms of job processing time. According to Theorem 3.8, there is an (m, ϵ)-canonical

scheduling $\Gamma_1(\mathcal{J})$ for the instance $\mathcal{J}$, under the same mode assignments, such that the makespan of $\Gamma_1(\mathcal{J})$ is bounded by $(1+\epsilon)Opt(\mathcal{J})$.

Consider a room $\gamma_{j,q,r}$ in the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$. Suppose that J_{p_1}, ..., J_{p_q} are the small jobs assigned to the room $\gamma_{j,q,r}$ by the scheduling $\Gamma_1(\mathcal{J})$. Then $\sum_{i=1}^{q} t_{p_i} \leq T_0$, where t_{p_i} is the processing time for the job J_{p_i} under $\Gamma_1(\mathcal{J})$, which is the same as under $\Gamma(\mathcal{J})$. Thus we must have

$$\sum_{i=1}^{q} \lfloor t_{p_i}/K \rfloor \leq \sum_{i=1}^{q} t_{p_i}/K \leq T_0/K \leq T_0'.$$

Therefore, under the same mode assignments (with processing time t_{ij} replaced by $\lfloor t_{ij}/K \rfloor$) and the same room assignments, the corresponding scheduling $\Gamma_1(\mathcal{J}')$ for the job set $\mathcal{J}'$ has no rooms with running time exceeding T_0'. Thus, by Lemma 4.1, when step 3 of the algorithm Approx-Scheme loops to the stage in which the large job set and their mode assignments, the tower types, and the ordering of the large jobs and the towers all match that in the scheduling $\Gamma_1(\mathcal{J}')$, the array element $D[n_S; \ldots, t_{j,q,r}, \ldots]$ corresponding to the room configurations of the scheduling $\Gamma_1(\mathcal{J}')$ must have value True. Thus, a scheduling $\Gamma_1'(\mathcal{J}')$ based on this configuration is constructed and its makespan is calculated. Note that the scheduling $\Gamma_1'(\mathcal{J}')$ may not be exactly the scheduling $\Gamma_1(\mathcal{J}')$. However, they must have exactly the same makespan.

Since step 4 of the algorithm Approx-Scheme picks the scheduling $\Gamma_0(\mathcal{J}')$ that has the smallest makespan over all schedulings for $\mathcal{J}'$ constructed in step 3, we conclude that the makespan of the scheduling $\Gamma_0(\mathcal{J}')$ is not larger than the makespan of the scheduling $\Gamma_1'(\mathcal{J}')$, thus not larger than the makespan of the scheduling $\Gamma_1(\mathcal{J}')$.

As we described in the paragraph before Lemma 4.4, to obtain the corresponding scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, we first expand the scheduling $\Gamma_0(\mathcal{J}')$ by K (i.e., multiplying the job processing times and starting times in $\Gamma_0(\mathcal{J}')$ by K). Let the resulting scheduling be $\Gamma_0(\mathcal{J}'')$. Similarly we expand the scheduling $\Gamma_1(\mathcal{J}')$ by K to obtain a scheduling $\Gamma_1(\mathcal{J}'')$. The makespan of the scheduling $\Gamma_0(\mathcal{J}'')$ is not larger than the makespan of the scheduling $\Gamma_1(\mathcal{J}'')$ since they are obtained by proportionally expanding the schedulings $\Gamma_0(\mathcal{J}')$ and $\Gamma_1(\mathcal{J}')$, respectively, by the same factor K.

Moreover, the makespan of $\Gamma_1(\mathcal{J}'')$ is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$. To see this, observe that these two schedulings use the same large job set under the same mode assignment, the same small job set under the same mode assignment and room assignment, and the same order of large jobs and towers. The only difference is that the processing time t_{ij} of each job J_i in $\Gamma_1(\mathcal{J})$ is replaced by a possibly smaller processing time $K \cdot \lfloor t_{ij}/K \rfloor$ of the corresponding job J_i'' in $\Gamma_1(\mathcal{J}'')$. In consequence, we conclude that the makespan of the scheduling $\Gamma_0(\mathcal{J}'')$ is not larger than the makespan of the (m,ϵ)-canonical scheduling $\Gamma_1(\mathcal{J})$, which is bounded by $(1+\epsilon)Opt(\mathcal{J})$.

Finally, to obtain the scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, we make corrections on the processing times of the jobs in the scheduling $\Gamma_0(\mathcal{J}'')$. More precisely, we replace the processing time $K \cdot \lfloor t_{ij}/K \rfloor$ for job J_i'' by t_{ij}, which is the processing time of the job J_i in the job set $\mathcal{J}$. Correcting the processing time for each job J_i'' in $\Gamma_0(\mathcal{J}'')$ may make the makespan of the scheduling increase by

$$t_{ij} - K \cdot \lfloor t_{ij}/K \rfloor < K.$$

Therefore, after the corrections of processing times for all jobs in $\mathcal{J}''$, the makespan of the finally resulting scheduling $\Gamma_0(\mathcal{J})$ for the job set $\mathcal{J}$, constructed by the algorithm

Approx-Scheme, is bounded by

$$\begin{aligned}\text{the makespan of } \Gamma_1(\mathcal{J}) + n \cdot K &\leq (1+\epsilon)Opt(\mathcal{J}) + \epsilon T_0/m \\ &\leq (1+2\epsilon)Opt(\mathcal{J}) \\ &= (1+\delta)Opt(\mathcal{J}).\end{aligned}$$

Here we have used the fact that $Opt(\mathcal{J}) \geq T_0/m$.

This completes the proof of the theorem. □

5. Conclusion and remarks. In this paper, we have developed a polynomial time approximation scheme for the $P_m|set|C_{\max}$ problem for any fixed constant m. The result is achieved by combinations of the recent techniques developed in the area of multiprocessor job schedulings plus the classical dynamic programming and scaling techniques. This result is a significant improvement over the previous results on the problem: no previous approximation algorithms for the problem $P_m|set|C_{\max}$ have their approximation ratio bounded by a constant that is independent of the number m of processors in the system. Our result also confirms a conjecture made by Amoura et al. [1]. In the following we make a few remarks on further work on the problem.

The multiprocessor job scheduling problem seems an intrinsically difficult problem. For example, if the number m of processors in the system is given as a variable in the input, then the problem becomes highly nonapproximable: there is a constant δ such that no polynomial time approximation algorithm for the problem can have an approximation ratio smaller than n^δ unless P = NP [25]. Observing this plus the difficulties in developing good approximation algorithms for the problem, people had suspected whether the $P_m|set|C_{\max}$ problem for some fixed m should be MAX-NP hard [8]. The present paper completely eliminates this possibility [2].

Our study shows that there are very "normalized" schedulings whose makespan is close to the optimal ones and that these "good" normalized schedulings can be constructed systematically. We are interested in investigating the tradeoff between the degree of this kind of normalization and the time complexity of approximation algorithms.

The current form of our polynomial time approximation scheme may not be practically useful, yet. Even for a small integer m and a reasonably small constant ϵ, the time complexity of our algorithm is bounded by a polynomial of very high degree. More recently, Jansen and Porkolab [21] use the approach of Amoura et al. [1] and are able to develop a linear time approximation scheme for the $P_m|set|C_{\max}$ problem, which still does not seem practical because of the huge constant factor in the complexity of the algorithm.

We are especially interested in developing more practical polynomial time approximation algorithms for systems with small number of processors, such as $P_4|set|C_{\max}$. In particular, we would like to develop practical approximation algorithms for the $P_m|set|C_{\max}$ problem with approximation ratio better than $m/2$, which is still the best known bound for the problem [8]. Some progress has recently been made toward this direction for systems of four processors [19].

Acknowledgment. The authors would like to acknowledge the helpful and stimulating discussions with Nancy Amato, Don Friesen, Klaus Jansen, Chung-Yee Lee, Lorant Porkolab, and Roberto Solis-Oba. Frank Ruskey has helped in the formula for the Bell numbers. The authors would especially like to thank two anonymous referees whose comments and suggestions have greatly improved the presentation of the paper.

REFERENCES

[1] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, in Proceedings of the 5th Annual European Symposium on Algorithms, Graz, Austria, Lecture Notes in Comput. Sci. 1284, Springer-Verlag, Berlin, 1997, pp. 1–12.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[3] L. Bianco, J. Blazewicz, P. Dell'Olmo, and M. Drozdowski, *Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors*, Ann. Oper. Res., 58 (1995), pp. 493–517.

[4] J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M. Speranza, *Scheduling multiprocessor tasks on three dedicated processors*, Inform. Process. Lett., 41 (1992), pp. 275–280.

[5] J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M. Speranza, *Erratum, Scheduling multiprocessor tasks on three dedicated processors*, Inform. Process. Lett., 49 (1994), pp. 269–270.

[6] J. Blazewicz, M. Drozdowski, and J. Weglarz, *Scheduling multiprocessor tasks to minimize scheduling length*, IEEE Trans. Comput., 35 (1986), pp. 389–393.

[7] J. Blazewicz, W. Drozdowski, and J. Weglarz, *Scheduling multiprocessor tasks—a survey*, Internat. J. Microcomput. Appl., 13 (1994), pp. 89–97.

[8] J. Chen and C.-Y. Lee, *General multiprocessor tasks scheduling*, Naval Res. Logist., 46 (1999), pp. 57–74.

[9] J. Chen and A. Miranda, *A polynomial time approximation scheme for general multiprocessor job scheduling*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99), Atlanta, 1999, pp. 418–427.

[10] P. Dell'Olmo, M. G. Speranza, and Zs. Tuza, *Efficiency and effectiveness of normal schedules on three dedicated processors*, Discrete Math., 164 (1997), pp. 67–79.

[11] G. Dobson and U. Karmarkar, *Simultaneous resource scheduling to minimize weighted flow times*, Oper. Res., 37 (1989), pp. 592–600.

[12] J. Du and J. Y.-T. Leung, *Complexity of scheduling parallel task systems*, SIAM J. Discrete Math., 2 (1989), pp. 473–487.

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

[14] M. X. Goemans, *An approximation algorithm for scheduling on three dedicated machines*, Discrete Appl. Math., 61 (1995), pp. 49–59.

[15] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[16] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1994.

[17] L. A. Hall, *Approximation algorithms for scheduling*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, 1997, pp. 1–45.

[18] J. A. Hoogeveen, S. L. van de Velde, and B. Veltman, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, Discrete Appl. Math., 55 (1994), pp. 259–272.

[19] J. Huang, J. Chen, and S. Chen, *A simple linear time approximation algorithm for multiprocessor job scheduling on four processors*, in Proceedings of the 11th Annual International Symposium on Algorithms and Computation (ISAAC 2000), Lecture Notes in Comput. Sci. 1969, Springer-Verlag, New York, 2000, pp. 60–71.

[20] O. H. Ibarra and C. E. Kim, *Fast approximation algorithms for the Knapsack and sum of subset problems*, J. Assoc. Comput. Mach., 22 (1975), pp. 463–468.

[21] K. Jansen and L. Porkolab, *General multiprocessor task scheduling: Approximate solutions in linear time*, in Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS '99), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, New York, Berlin, 1999, pp. 110–121.

[22] H. Krawczyk and M. Kubale, *An approximation algorithm for diagnostic test scheduling in multicomputer systems*, IEEE Trans. Comput., 34 (1985), pp. 869–872.

[23] C.-Y. Lee and X. Cai, *Scheduling multiprocessor tasks without prespecified processor allocations*, IIE Transactions, to appear.

[24] C.-Y. Lee, L. Lei, and M. Pinedo, *Current trends in deterministic scheduling*, Ann. Oper. Res., 70 (1997), pp. 1–42.

[25] A. Miranda, *Approximation Algorithms in Multiprocessor Task Scheduling*, Ph.D. thesis, Department of Computer Science, Texas A&M University, College Station, TX, 1998.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 18–26

A DECOMPOSITION THEOREM FOR MAXIMUM WEIGHT BIPARTITE MATCHINGS*

MING-YANG KAO†, TAK-WAH LAM‡, WING-KIN SUNG‡, AND HING-FUNG TING‡

Abstract. Let G be a bipartite graph with positive integer weights on the edges and without isolated nodes. Let n, N, and W be the node count, the largest edge weight, and the total weight of G. Let $k(x, y)$ be $\log x/\log(x^2/y)$. We present a new decomposition theorem for maximum weight bipartite matchings and use it to design an $O(\sqrt{n}W/k(n, W/N))$-time algorithm for computing a maximum weight matching of G. This algorithm bridges a long-standing gap between the best known time complexity of computing a maximum weight matching and that of computing a maximum cardinality matching. Given G and a maximum weight matching of G, we can further compute the weight of a maximum weight matching of $G - \{u\}$ for all nodes u in $O(W)$ time.

Key words. all-cavity matchings, maximum weight matchings, minimum weight covers, graph algorithms, unfolded graphs

AMS subject classifications. 05C05, 05C70, 05C85, 68Q25

PII. S0097539799361208

1. Introduction.

Let $G = (X, Y, E)$ be a bipartite graph with positive integer weights on the edges. A *matching* of G is a subset of node-disjoint edges of G. Let $\mathrm{mwm}(G)$ (respectively, $\mathrm{mm}(G)$) denote the maximum weight (respectively, cardinality) of any matching of G. A *maximum weight* matching is one whose weight is $\mathrm{mwm}(G)$. Let N be the largest weight of any edge. Let W be the total weight of G. Let n and m be the numbers of nodes and edges of G; to avoid triviality, we maintain $m = \Omega(n)$ throughout the paper.

The problem of finding a maximum weight matching of a given G has a rich history. The first known polynomial-time algorithm is the $O(n^3)$-time Hungarian method [15]. Fredman and Tarjan [5] used Fibonacci heaps to improve the time to $O(n(m + n\log n))$. Gabow [6] introduced scaling to solve the problem in $O(n^{3/4}m\log N)$ time by taking advantage of the integrality of edge weights. Gabow and Tarjan [7] improved the scaling method to further reduce the time to $O(\sqrt{n}m\log(nN))$. For the case where the edges all have weight 1, i.e., $N = 1$ (and $W = m$), Hopcroft and Karp [11] gave an $O(\sqrt{n}W)$-time algorithm, and Feder and Motwani [4] improved the time complexity to $O(\sqrt{n}W/k(n, m))$, where $k(x, y) = \log x/\log(x^2/y)$. It has remained open whether the gap between the running times of the Gabow–Tarjan algorithm and the latter two algorithms can be closed for the case where $W = o(m\log(nN))$.

We resolve this open problem in the affirmative by giving an $O(\sqrt{n}W/k(n, W/N))$-time algorithm for general W. Note that $W/N = m$ when all the edges have the same weight. The algorithm does not use scaling but instead employs a novel decomposition theorem for weighted bipartite matchings (Theorem 2.2). We also use the theorem to

*Received by the editors September 15, 1999; accepted for publication (in revised form) November 8, 2000; published electronically May 31, 2001. A preliminary version appeared in the *Proceedings of the 7th Annual European Symposium on Algorithms*, Lecture Notes in Comput. Sci. 1643, Springer, Prague, Czech Republic, 1999, pp. 439–449.

http://www.siam.org/journals/sicomp/31-1/36120.html

†Department of Computer Science, Yale University, New Haven, CT 06520 (kao-ming-yang@cs.yale.edu). This author's research was supported in part by NSF grant 9531028.

‡Department of Computer Science and Information Systems, University of Hong Kong, Hong Kong (twlam@csis.hku.hk, wksung@csis.hku.hk, hfting@csis.hku.hk). The research of these authors was supported in part by Hong Kong RGC grant HKU-7027/98E.

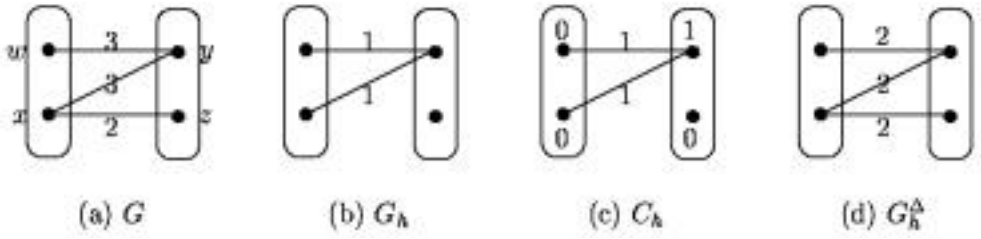


FIG. 1. *Consider $h = 1$. G is decomposed into G_h and G_h^Δ; C_h is a minimum weight cover of G_h.*

solve the *all-cavity maximum weight matching* problem which, given G and a maximum weight matching of G, asks for $\text{mwm}(G-\{u\})$ for all nodes u in G. This problem has applications to tree comparisons [2, 14]. The case where $N = 1$ has been studied by Chung [2]. Recently, Kao, Lam, Sung, and Ting [12] gave an $O(\sqrt{n}m \log N)$-time algorithm for general N. This paper presents a new algorithm that runs in $O(W)$ time.

Section 2 presents the decomposition theorem and uses it to compute the weight of a maximum weight matching. Section 3 gives an algorithm to construct a maximum weight matching. Section 4 solves the all-cavity matching problem.

2. The decomposition theorem. In section 2.1, we state the decomposition theorem and use the theorem to design an algorithm to compute the weight $\text{mwm}(G)$ in $O(\sqrt{n}W/k(n, W/N))$ time. In section 2.2, we prove the decomposition theorem. In section 3, we further construct a maximum weight matching itself within the same time bound.

2.1. An algorithm for computing mwm(G). Let $V(G)$ be the node set of G, i.e., $X \cup Y$. Let $w(u, v)$ denote the weight of an edge $uv \in G$; if u is not adjacent to v, let $w(u, v) = 0$. A *cover* of G is a function $C : X \cup Y \to \{0, 1, 2, \ldots\}$ such that $C(x) + C(y) \geq w(x, y)$ for all $x \in X$ and $y \in Y$. Let $w(C) = \sum_{z \in X \cup Y} C(z)$ be the weight of C. C is a *minimum weight* cover if $w(C)$ is the smallest possible. Let $\text{mwc}(G)$ denote the weight of a minimum weight cover of G. A minimum weight cover is a dual of a maximum weight matching as stated in the next fact.

FACT 2.1 (see [1]). *Let C be a cover and M be a matching of G. The following statements are equivalent.*

1. *C is a minimum weight cover and M is a maximum weight matching of G.*
2. *$\sum_{uv \in M} w(u, v) = \sum_{u \in X \cup Y} C(u)$.*
3. *Every node in $\{u \mid C(u) > 0\}$ is matched by some edge in M, and $C(u) + C(v) = w(u, v)$ for all $uv \in M$.*

For an integer $h \in [1, N]$, we divide G into two lighter bipartite graphs G_h and G_h^Δ as follows:

- G_h is formed by the edges uv of G with $w(u, v) \in [N - h + 1, N]$. Each edge uv in G_h has weight $w(u, v) - (N - h)$. For example, G_1 is formed by the heaviest edges of G, and the weight of each edge is exactly one.
- Let C_h be a minimum weight cover of G_h. G_h^Δ is formed by the edges uv of G with $w(u, v) - C_h(u) - C_h(v) > 0$. The weight of uv is $w(u, v) - C_h(u) - C_h(v)$.

An example is depicted in Figure 1. Note that the total weight of G_h and G_h^Δ is at most W.

The next theorem is the decomposition theorem.

THEOREM 2.2. $\mathrm{mwm}(G) = \mathrm{mwm}(G_h) + \mathrm{mwm}(G_h^{\Delta})$; *in particular,* $\mathrm{mwm}(G) = \mathrm{mm}(G_1) + \mathrm{mwm}(G_1^{\Delta})$.

Proof. See section 2.2. □

Theorem 2.2 suggests the following recursive algorithm to compute $\mathrm{mwm}(G)$.

PROCEDURE Compute-MWM(G).

1. Construct G_1 from G.
2. Compute $\mathrm{mm}(G_1)$ and find a minimum weight cover C_1 of G_1.
3. Construct G_1^{Δ} from G and C_1.
4. If G_1^{Δ} is empty, then return $\mathrm{mm}(G_1)$; otherwise, return $\mathrm{mm}(G_1)$+Compute-MWM(G_1^{Δ}).

THEOREM 2.3. Compute-MWM(G) *finds* $\mathrm{mwm}(G)$ *in* $O(\sqrt{n}W/k(n, W/N))$ *time.*

Proof. The correctness of Compute-MWM follows from Theorem 2.2. Below, we analyze the running time. We initialize a maximum heap [3] in $O(m)$ time to store the edges of G according to their weights. Let $T(n, W, N)$ be the running time of Compute-MWM excluding this initialization. Let L be the set of the heaviest edges in G. Then Step 1 takes $O(|L| \log m)$ time. In Step 2, we can compute $\mathrm{mm}(G_1)$ in $O(\sqrt{n}|L|/k(n, |L|))$ time [4]. From this matching, C_1 can be found in $O(|L|)$ time [1]. Let L_1 be the set of the edges of G adjacent to some node u with $C_1(u) > 0$; i.e., L_1 consists of the edges of G whose weights are reduced in G_1^{Δ}. Let $\ell_1 = |L_1|$. Step 3 updates every edge of L_1 in the heap in $O(\ell_1 \log m)$ time. As $L \subseteq L_1$, Steps 1 to 3 altogether use $O(\sqrt{n}\ell_1/k(n, \ell_1))$ time. Since the total weight of G_1^{Δ} is at most $W - \ell_1$, Step 4 uses at most $T(n, W - \ell_1, N')$ time, where $N' < N$ is the maximum edge weight of G_1^{Δ}. In summary, for some positive integer $\ell_1 \le W$,

$$T(n, W, N) = O(\sqrt{n}\ell_1/k(n, \ell_1)) + T(n, W - \ell_1, N'),$$

where $T(n, 0, N') = 0$. By recursion, for some positive integers $\ell_1, \ell_2, \ldots, \ell_p$ with $p \le N$ and $\sum_{1 \le i \le p} \ell_i = W$,

$$\begin{aligned} T(n, W, N) &= O\left(\sqrt{n}\left(\frac{\ell_1}{k(n, \ell_1)} + \frac{\ell_2}{k(n, \ell_2)} + \cdots + \frac{\ell_p}{k(n, \ell_p)}\right)\right) \\ &= O\left(\frac{\sqrt{n}}{\log n}\left(\left(\sum_{1 \le i \le p} \ell_i\right)\log n^2 - \sum_{1 \le i \le p} \ell_i \log \ell_i\right)\right). \end{aligned}$$

Since $x \log x$ is convex, by Jensen's inequality [10],

$$\sum_{1 \le i \le p} \ell_i \log \ell_i \ge \left(\sum_{1 \le i \le p} \ell_i\right) \log \frac{\sum_{1 \le i \le p} \ell_i}{p} \ge W \log \frac{W}{N}.$$

Therefore,

$$\begin{aligned} T(n, W, N) &= O\left(\frac{\sqrt{n}}{\log n}\left(W \log n^2 - W \log \frac{W}{N}\right)\right) \\ &= O\left(\frac{\sqrt{n}W}{\log n / \log(n^2/\frac{W}{N})}\right) = O\big(\sqrt{n}W/k(n, W/N)\big). \end{aligned}$$

□

2.2. Proof of Theorem 2.2. This section proves the statement that $\text{mwm}(G) = \text{mwm}(G_h)+\text{mwm}(G_h^\Delta)$, where G_h^Δ is defined according to an arbitrary minimum weight cover C_h of G_h. By Fact 2.1, it suffices to prove $\text{mwc}(G) = w(C_h) + \text{mwc}(G_h^\Delta)$.

To show the direction $\text{mwc}(G) \leq w(C_h) + \text{mwc}(G_h^\Delta)$, note that any cover D of G_h^Δ augmented with C_h gives a cover C of G, where $C(u) = C_h(u) + D(u)$ for each node u of G. Then $C(u) + C(v) \geq w(u, v)$ for all edges uv of G. Thus, $\text{mwc}(G) \leq w(C_h) + \text{mwc}(G_h^\Delta)$.

To show the direction $w(C_h) + \text{mwc}(G_h^\Delta) \leq \text{mwc}(C)$, let C be a minimum weight cover of G. A node u of G is called *bad* if $C(u) < C_h(u)$. Lemma 2.4 below shows that G must have a minimum weight cover C allowing no bad node. Then we can construct a cover D of G_h^Δ as follows. For each node u of G, define $D(u) = C(u) - C_h(u)$, which must be at least 0. D is a cover of G_h^Δ because for any edge uv of G_h^Δ, $D(u) + D(v) = C(u) + C(v) - C_h(u) - C_h(v) \geq w(u, v) - C_h(u) - C_h(v)$. Note that $w(D) = w(C) - w(C_h)$. Thus, $\text{mwc}(G_h^\Delta) \leq w(C) - w(C_h)$, or equivalently, $\text{mwc}(G_h^\Delta) + w(C_h) \leq \text{mwc}(G)$.

The next lemma concludes the proof of Theorem 2.2.

LEMMA 2.4. *There exists a minimum weight cover of G such that no node of G is bad.*

Proof. Suppose, for the sake of contradiction, that every minimum weight cover allows some bad node. Then we can obtain a contradiction by constructing another minimum weight cover with no bad node.

Let C be a minimum weight cover of G with u as a bad node, i.e., $C(u) < C_h(u)$. Recall that C_h is a minimum weight cover of G_h. Consider a maximum weight matching M of G_h. By Fact 2.1, since $C_h(u) > C(u) \geq 0$, u is matched by an edge in M, say, to a node v, and $C_h(u) + C_h(v) = w(u, v) - (N - h)$. We call v the *mate* of u. Note that v cannot be a bad node; otherwise, $C(u)+C(v) < w(u, v)-(N-h) \leq w(u, v)$ and a contradiction occurs.

Since C is a cover of G, $C(u) + C(v) \geq w(u, v)$. Thus, $C(v) \geq w(u, v) - C(u) \geq N - h + C_h(u) + C_h(v) - C(u)$. Define another cover C' of G as follows. For each bad node defined by C, let v be the mate of u, define $C'(u) = C_h(u)$ and $C'(v) = C(v) - (C_h(u) - C(v))$. Note that u is not a bad node with respect to C', and neither is v since $C'(v) \geq N - h + C_h(v) \geq C_h(v)$. For all other nodes x, $C'(x)$ is the same as $C(x)$. Therefore, if C' is a cover of G, C' allows no bad node. Also, $w(C') = w(C)$.

It remains to prove that C' is a cover of G. By the definition of C', $C'(v) < C(v)$ if and only if v is the mate of a bad node with respect to C. Suppose C' is not a cover of G. Then there exists an edge vt such that $C'(v) + C'(t) \leq w(v, t)$ and v is the mate of a bad node. Recall that the latter implies that $C'(v) \geq N - h + C_h(v)$. In other words,

$$C'(t) < w(v, t) - C'(v) \leq w(v, t) - (N - h) - C_h(v).$$

We can derive a contradiction as follows.

Case 1: $w(v, t) \leq N - h$. Then $C'(t) < -C_h(v) \leq 0$, which contradicts that $C'(t) \geq C_h(t) \geq 0$.

Case 2: $w(v, t) > N - h$. Then G_h contains the edge vt and $C_h(v) + C_h(t) \geq w(v, t) - (N - h)$. Thus, $C'(t) < w(v, t) - (N - h) - C_h(v) \leq C_h(t)$, which contradicts the fact that C' allows no bad node.

In conclusion, C' is a cover of G. Together with the fact that $w(C) = w(C')$, we obtain the desired contradiction that C' is a minimum weight cover of G with no bad node. Lemma 2.4 follows. ∎

3. Construct a maximum weight matching. The algorithm in section 2.1 only computes the value of mwm(G). To report the edges involved, we show below how to first construct a minimum weight cover of G in $O(\sqrt{n}W/k(n, W/N))$ time and then use this cover to construct a maximum weight matching in $O(\sqrt{n}m/k(n, m))$ time. Thus, the time required to construct a maximum weight matching is $O(\sqrt{n}W/k(n, W/N))$.

LEMMA 3.1. *Assume that h, G_h, C_h, and G_h^Δ are defined as in section 2. Let C_h^Δ be any minimum weight cover of G_h^Δ. If D is a function on $V(G)$ such that for every $u \in V(G)$, $D(u) = C_h(u) + C_h^\Delta(u)$, then D is a minimum weight cover of G.*

Proof. Consider any edge uv of G. If uv is not in G_h^Δ, then $w(u, v) \le C_h(u) + C_h(v) \le D(u) + D(v)$. Assume that uv is in G_h^Δ. Note that its weight in G_h^Δ is $w(u, v) - C_h(u) - C_h(v)$. Since C_h^Δ is a cover, $C_h^\Delta(u) + C_h^\Delta(v) \ge w(u, v) - C_h(u) - C_h(v)$. Thus, $D(u) + D(v) = C_h(u) + C_h^\Delta(u) + C_h(v) + C_h^\Delta(v) \ge w(u, v)$. It follows that D is a cover of G. To show that D is a minimum weight one, we observe that

$$\begin{array}{lcll} \sum_{u\in V(G)} D(u) & = & \sum_{u\in V(G)} C_h(u) + C_h^\Delta(u) & \\ & = & \sum_{u\in V(G)} C_h(u) + \sum_{u\in V(G)} C_h^\Delta(u) & \\ & = & \mathrm{mwm}(G_h) + \mathrm{mwm}(G_h^\Delta) & \text{by Fact 2.1} \\ & = & \mathrm{mwm}(G) & \text{by Theorem 2.2.} \end{array}$$

By Fact 2.1, D is minimum. ∎

By Lemma 3.1, a minimum weight cover of G can be computed using a recursive procedure similar to Compute-MWM as follows.

PROCEDURE Compute-Min-Cover(G).

1. Construct G_1 from G.
2. Find a minimum weight cover C_1 of G_1.
3. Construct G_1^Δ from G and C_1.
4. If G_1^Δ is empty, then return C_1; otherwise, let C_1^Δ = Compute-Min-Cover(G_1^Δ) and return D, where for all nodes u in G, $D(u) = C_1(u) + C_1^\Delta(u)$.

THEOREM 3.2. Compute-Min-Cover(G) *correctly computes a minimum weight cover of G in $O(\sqrt{n}W/k(n, W/N))$ time.*

Proof. The correctness of Compute-Min-Cover(G) follows from Lemma 3.1. For the time complexity, the analysis is similar to that of Theorem 2.3. ∎

Now, we show how to recover a maximum weight matching of G from a minimum weight cover D of G.

PROCEDURE Recover-Max-Matching(G, D).

1. Let H be the subgraph of G that contains all edges uv with $w(u, v) = D(u) + D(v)$.
2. Make two copies of H. Call them H^a and H^b. For each node u of H, let u^a and u^b denote the corresponding nodes in H^a and H^b, respectively.
3. Union H^a and H^b to form H^{ab}, and add to H^{ab} the set of edges $\{u^a u^b \mid u \in V(H),\ D(u) = 0\}$.
4. Find a maximum cardinality matching K of H^{ab} and return the matching $K^a = \{uv \mid u^a v^a \in K\}$.

THEOREM 3.3. Recover-Max-Matching(G, D) *correctly computes a maximum weight matching of G in $O(\sqrt{n}m/k(n, m))$ time.*

Proof. The running time of Recover-Max-Matching(G, D) is dominated by the construction of K. Since H^{ab} has at most $2n$ nodes and at most $3m$ edges, K can be constructed in $O(\sqrt{n}m/k(n, m))$ time using the Feder–Motwani algorithm [4].

It remains to show that K^a is a maximum weight matching of G. First, we argue that H^{ab} has a perfect matching. Let M be a maximum weight matching of G. By

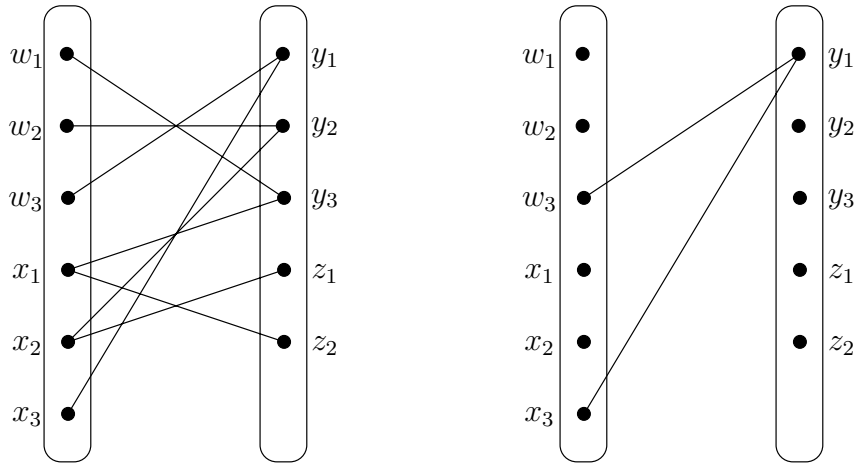


FIG. 2. (a) *The unfolded graph $\phi(G)$ of the bipartite graph given in Figure* 1(a). (b) *With respect to the cover C_h defined in Figure* 1(c)*, the node y_1 in $\phi(G)$ is the only node satisfying the condition that $1 \leq C_h(y)$. Thus, $\phi(G)|C_h$ comprises only the edges incident to y_1.*

Fact 2.1, $D(u) + D(v) = w(u, v)$ for every edge $uv \in M$. Therefore, M is also a matching of H. Let U be the set of nodes in H unmatched by M. By Fact 2.1, $D(u) = 0$ for all $u \in U$. Let Q be $\{u^a u^b \mid u \in U\}$. Let $M^a = \{u^a v^a \mid uv \in M\}$ and $M^b = \{u^b v^b \mid uv \in M\}$. Note that $Q \cup M^a \cup M^b$ forms a matching in H^{ab} and every node in H^{ab} is matched by either Q, M^a, or M^b. Thus, H^{ab} has a perfect matching.

Since K is a maximum cardinality matching of H^{ab}, K must be a perfect matching. For every node u with $D(u) > 0$, u^a must be matched by K. Since there is no edge between u^a and any x^b in H^{ab}, there exists some v^a with $u^a v^a \in K$. Thus, every node u with $D(u) > 0$ must be matched by some edge in K^a. Therefore, $\sum_{uv \in K^a} w(u, v) = \sum_{u \in X \cup Y, D(u) > 0} D(u) = \sum_{u \in X \cup Y} D(u) = \mathrm{mwm}(G)$, and K^a is a maximum weight matching of G. ∎

4. All-cavity maximum weight matchings. In section 4.1, we introduce the notion of an *unfolded graph*. In section 4.2, we use this notion to design an algorithm which, given a weighted bipartite graph G and a maximum weight matching of G, computes $\mathrm{mwm}(G - \{u\})$ for all nodes u in G using $O(W)$ time.

4.1. Unfolded graphs. The *unfolded graph* $\phi(G)$ of G is defined as follows.

- For each node u of G, $\phi(G)$ has α copies of u, denoted as $u^1, u^2, \ldots, u^\alpha$, where α is the weight of the heaviest edge incident to u.
- For each edge uv of G, $\phi(G)$ has the edges $u^1 v^\beta, u^2 v^{\beta-1}, \ldots, u^\beta v^1$, where $\beta = w(u, v)$.

See Figure 2(a) for an example. Let M be a matching of G. Consider M as a weighted bipartite graph; then, by definition, $\phi(M) = \bigcup_{uv \in M} \{u^1 v^\beta, \ldots, u^\beta v^1 \mid \beta = w(u, v)\}$ is a matching of $\phi(G)$. The number of edges in $\phi(M)$ is equal to the total weight of the edges in M, i.e., $|\phi(M)| = \sum_{uv \in M} w(u, v)$. The next lemma relates G and $\phi(G)$.

LEMMA 4.1. *Assume that M is a maximum weight matching of G.*

1. $\mathrm{mwm}(G) = \mathrm{mm}(\phi(G))$.
2. *The set $\phi(M)$ is a maximum cardinality matching of $\phi(G)$.*

Proof. Statement 4.1 follows from Statement 4.1. Statement 4.1 is proved as follows. Since M is a maximum weight matching of G, $\text{mwm}(G) = \sum_{uv \in M} w(u,v) = |\phi(M)| \leq \text{mm}(\phi(G))$. By Fact 2.1, $\text{mwm}(G) \geq \text{mm}(\phi(G))$ if and only if $\text{mwc}(G) \geq \text{mwc}(\phi(G))$. We prove the latter as follows. Given a minimum weight cover C of G, we can obtain a cover C' of $\phi(G)$ as follows. For any node u of G, $C'(u^i) = 1$ if $C(u) > 0$ and $i \leq C(u)$; otherwise, $C'(u^i) = 0$. Note that $w(C') = w(C) = \text{mwc}(G)$. Therefore, $\text{mwc}(G) \geq \text{mwc}(\phi(G))$ and $\text{mwm}(G) \geq \text{mm}(\phi(G))$. □

4.2. An algorithm for all-cavity maximum weight matchings. Let M be a given maximum weight matching of G.

By Lemma 4.1(2), $\phi(M)$ is a maximum cardinality matching of $\phi(G)$. In light of this maximality, we say that a path in $\phi(G)$ is *alternating* for $\phi(M)$ if (1) its edges alternate between being in $\phi(M)$ and being not in $\phi(M)$ and (2) in the case the first (respectively, last) node is matched by $\phi(M)$, the path contains the matched edge of u as the first (respectively, last) edge. The length of an alternating path is its number of edges. An alternating path may have zero length; in this case, the path contains exactly one unmatched node. An alternating path P can modify $\phi(M)$ to another matching, i.e., $(\phi(M) \cup P) - (\phi(M) \cap P)$. If P is of even length, the resulting matching has the same size as $\phi(M)$. If P is of odd length, P modifies M to a strictly smaller or bigger matching; yet the latter is impossible because $\phi(M)$ is maximum. Intuitively, we would like to maximize the size of the resultant matching and even-length alternating paths are preferred.

Our new algorithm for computing $\text{mwm}(G - \{u\})$ is based on the observation that $\text{mwm}(G - \{u\})$ can be determined by detecting the smallest i such that u^i has an even-length alternating path for $\phi(M)$. Details are as follows.

Definition. For each u^i in $\phi(G)$, let $\rho(u^i) = 0$ if there is an even-length alternating path for $\phi(M)$ starting from u^i; otherwise, let $\rho(u^i) = 1$.

The following lemma states a monotone property of $\rho(u^i)$ over different i's.

LEMMA 4.2. *Consider any node u in G. Let $u^1, u^2, \ldots, u^\beta$ be its corresponding nodes in $\phi(G)$. If $\rho(u^i) = 0$, then $\rho(u^j) = 0$ for all $j \in [i, \beta]$. Furthermore, there exist $\beta - i + 1$ node-disjoint even-length alternating paths $P_i, P_{i+1}, \ldots, P_\beta$ for $\phi(M)$, where each P_j starts from u^j.*

Proof. As $\rho(u^i) = 0$, let $P_i = u_0^{a_0}, v_0^{b_0}, u_1^{a_1}, v_1^{b_1}, \ldots, u_{p-1}^{a_{p-1}}, v_{p-1}^{b_{p-1}}, u_p^{a_p}$ be a shortest even-length alternating path for $\phi(M)$, where $u_0^{a_0} = u^i$.

Based on P_i, we can construct an even-length alternating path P_{i+1} for $\phi(M)$ starting from u^{i+1} as follows. If u^{i+1} is not matched by $\phi(M)$, P_{i+1} is simply a path of zero length. From now on, we assume that u^{i+1} is matched by $\phi(M)$. As P is of even length, $u_p^{a_p}$ is not matched by $\phi(M)$. Then, by the definition of $\phi(M)$, $u_p^{a_p+1}$ is also not matched by $\phi(M)$. Let h be the smallest integer in $[1, p]$ such that $u_h^{a_h+1}$ is not matched by $\phi(M)$. Notice that, for all $\ell < h$, $u_\ell^{a_\ell+1}$ is matched to $v_\ell^{b_\ell-1}$; furthermore, $\phi(G)$ contains an edge between $v_\ell^{b_\ell-1}$ and $u_{\ell+1}^{a_{\ell+1}+1}$. Thus, $P_{i+1} = u^{i+1}, v_0^{b_0-1}, u_1^{a_1+1}, v_1^{b_1-1}, \ldots, u_h^{a_h+1}$ is an even-length alternating path for $\phi(M)$. Similarly, for $j = i+2, \ldots, \beta$, we can use P_i to define an even-length alternating path P_j for $\phi(M)$ starting from u^j. By construction, $P_i, P_{i+1}, \ldots, P_\beta$ are node-disjoint. □

The next lemma is the basis of our cavity matching algorithm. It shows that given $\text{mwm}(G)$ (i.e., the weight of M), we can compute $\text{mwm}(G - \{u\})$ from the values $\rho(u^i)$, and all the $\rho(u^i)$'s can be found in $O(W)$ time.

LEMMA 4.3.

1. $\sum_{1\le i\le\beta}\rho(u^i) = \mathrm{mwm}(G) - \mathrm{mwm}(G-\{u\})$.
2. *For all* $u^i \in \phi(G)$, $\rho(u^i)$ *can be computed in* $O(W)$ *time in total.*

Proof. The two statements are proved as follows.

Statement 1. Let k be the largest integer such that $\rho(u^k) = 1$. By Lemma 4.2, $\rho(u^i) = 1$ for all $1 \le i \le k$, and 0 otherwise. Note that if $\rho(u^i) = 1$, u^i must be matched by $\phi(M)$. Thus, $\sum_{1\le i\le\beta}\rho(u^i) = k$. Below, we prove the following two equalities:

(1) $\mathrm{mm}(\phi(G) - \{u^1,\ldots,u^k\}) = \mathrm{mm}(\phi(G)) - k$.
(2) $\mathrm{mm}(\phi(G) - \{u^1,\ldots,u^\beta\}) = \mathrm{mm}(\phi(G) - \{u^1,\ldots,u^k\})$.

Then, by Lemma 4.1, $\mathrm{mwm}(G) = \mathrm{mm}(\phi(G))$ and $\mathrm{mwm}(G-\{u\}) = \mathrm{mm}(\phi(G) - \{u^1,\ldots,u^\beta\})$. Thus, $\mathrm{mwm}(G) - \mathrm{mwm}(G-\{u\}) = k$ and Statement 1 follows.

To show equality (1), let H be the set of edges of $\phi(M)$ incident to u^i with $1 \le i \le k$. Let $M' = \phi(M) - H$. Then, $|M'| = |\phi(M)| - k$. We claim that M' is a maximum cardinality matching of $\phi(G) - \{u^1,\ldots,u^k\}$. Hence, $\mathrm{mwm}(\phi(G) - \{u^1,\ldots,u^k\}) = |\phi(M)| - k$, and equality (1) follows. We prove the claim by contradiction. Suppose M' is not a maximum cardinality matching of $\phi(G) - \{u^1,\ldots,u^k\}$. Then, there exists an alternating path P that can modify M' to a larger matching of $\phi(G) - \{u^1,\ldots,u^k\}$ [8, 9]; in particular, the length of P must be odd and both of its endpoints are not matched by M'. P must start from some node v^j with $u^iv^j \in \phi(M)$ and $i < k$; otherwise, P is alternating for $\phi(M)$ in G and $\phi(M)$ cannot be a maximum cardinality matching of $\phi(G)$. Let Q be a path formed by joining u^iv^j with P. Q is an even-length alternating path for $\phi(M)$ starting from u^i in $\phi(G)$. This contradicts the fact that there is no even-length alternating path for $\phi(M)$ starting from u^i for $i < k$.

To show equality (2), we first note that $\mathrm{mm}(\phi(G) - \{u^1,\ldots,u^\beta\}) \le \mathrm{mm}(\phi(G) - \{u^1,\ldots,u^k\})$. It remains to prove the other direction. By Lemma 4.2, we can find $\beta - k$ node-disjoint even-length alternating paths $P_{k+1},\ldots,P_\beta$ for $\phi(M)$, which start from $u^{k+1},\ldots,u^\beta$. P_j starts at u^j. Let $M'' = (\phi(M) \cup (P_{j+1} \cup \cdots \cup P_\beta)) - (\phi(M) \cap (P_{j+1} \cup \cdots \cup P_\beta))$. Note that $|M''| = |\phi(M)|$ and there are no edges in M'' incident to any of $u^{k+1},\ldots,u^\beta$. M'' is a matching of $\phi(G) - \{u^{k+1},\ldots,u^\beta\}$ and $M'' - H$ of $\phi(G) - \{u^1,\ldots,u^\beta\}$. $|M'' - H| \ge |M''| - k = |\phi(M)| - k$. Since $\mathrm{mm}(\phi(G) - \{u^1,\ldots,u^k\}) = |\phi(M)| - k$ by equality (1), it follows that $\mathrm{mm}(\phi(G) - \{u^1,\ldots,u^\beta\}) \ge |M'' - H| \ge \mathrm{mm}(\phi(G) - \{u^1,\ldots,u^k\})$. Therefore, equality (2) holds.

Statement 2. We want to determine whether $\rho(u^i) = 0$ for all nodes $u^i \in \phi(G)$ in $O(W)$ time. By definition, $\rho(u^i) = 0$ if and only if there is an even-length alternating path for $\phi(M)$ starting from u^i. Let us partition the nodes of $\phi(G)$ into two parts: $\phi(X) = \{u^i \in \phi(G) \mid u \in X\}$ and $\phi(Y) = \{u^i \in \phi(G) \mid u \in Y\}$. Below, we give the details of computing $\rho(u^i)$ for all $u^i \in \phi(X)$. The case where $u^i \in \phi(Y)$ is symmetric.

Let D be a directed graph over the node set $\phi(X)$. D contains an edge u^iv^j if there exists a node $w^k \in \phi(Y)$ such that $u^iw^k \in \phi(G) - \phi(M)$ and $w^kv^j \in \phi(M)$. Consider any node v^j of D that is unmatched by $\phi(M)$. A directed path in D from v^j to a node u^i corresponds to a path in $\phi(G)$, which is indeed an even-length alternating path for $\phi(M)$ starting from u^i. Therefore, for any $u^i \in \phi(X)$, $\rho(u^i) = 0$ if and only if u^i is reachable from some node in D that is unmatched by $\phi(M)$. We can identify all such u^i by using a depth-first search on D starting with all the nodes unmatched by M. The time required is $O(|D|)$. As $|D| \le |\phi(G)| = W$, the lemma follows. ∎

The following procedure computes $\mathrm{mwm}(G-\{u\})$ for all nodes u of G. Let M be a maximum weight matching of G.

PROCEDURE Compute-All-Cavity(G, M).

1. Construct $\phi(G)$ and $\phi(M)$.
2. For every $j \in [0, n/2]$, determine A_j from $\phi(M)$.
3. For every node u^i of $\phi(G)$, if $u^i \in \bigcup_j A_j$ then $\rho(u^i) = 0$; otherwise $\rho(u^i) = 1$.
4. For every node u of G, compute $\text{mwm}(G - \{u\}) = \text{mwm}(G) - \sum_{1 \le i \le \beta} \rho(u^i)$, where u^1, u^2, ..., u^β are the nodes corresponding to u in $\phi(G)$.

THEOREM 4.4. Compute-All-Cavity(G, M) *correctly computes* $\text{mwm}(G - \{u\})$ *for all* u *of* G *in* $O(W)$ *time.*

Proof. The proof follows from Lemma 4.3 □

Acknowledgments. The authors wish to thank the anonymous referee for extremely helpful comments, which significantly improved the presentation of the paper. In particular, Theorem 2.2 was originally proved using unfolded graphs (see the conference version of this paper [13]); the new proof is based on a suggestion by the referee.

REFERENCES

[1] J. BONDY AND U. MURTY, *Graph Theory with Applications*, North-Holland, New York, 1976.
[2] M. J. CHUNG, *$O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees*, J. Algorithms, 8 (1987), pp. 106–112.
[3] T. H. CORMEN, C. L. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
[4] T. FEDER AND R. MOTWANI, *Clique partitions, graph compression and speeding-up algorithms*, J. Comput. System Sci., 51 (1995), pp. 261–272.
[5] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.
[6] H. N. GABOW, *Scaling algorithms for network problems*, J. Comput. System Sci., 31 (1985), pp. 148–168.
[7] H. N. GABOW AND R. E. TARJAN, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.
[8] Z. GALIL, *Efficient algorithms for finding maximum matching in graphs*, ACM Computing Surveys, 18 (1986), pp. 23–38.
[9] A. M. H. GERARDS, *Matching*, in Handbooks in Operations Reserach and Management Science 7: Network Models, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., North-Holland, Amsterdam, 1995, pp. 135–224.
[10] G. H. HARDY, J. E. LITTLEWOOD, AND G. PÓLYA, *Inequalities*, Cambridge University Press, Cambridge, UK, 1988. Reprint of the 1952 edition.
[11] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225–231.
[12] M. Y. KAO, T. W. LAM, W. K. SUNG, AND H. F. TING, *All-cavity maximum matchings*, in Proceedings of the 8th Annual International Symposium on Algorithms and Computation, Lecture Notes Comput. Sci. 1350, H. Imai and H. W. Leong, eds., Springer-Verlag, New York, NY, 1997, pp. 364–373.
[13] M. Y. KAO, T. W. LAM, W. K. SUNG, AND H. F. TING, *A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees*, in Proceedings of the 7th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1643, J. Nešetřil, ed., Springer-Verlag, New York, NY, 1999, pp. 438–449.
[14] M. Y. KAO, T. W. LAM, W. K. SUNG, AND H. F. TING, *Cavity matchings, label compressions, and unrooted evolutionary trees*, SIAM J. Comput. 30 (2000), pp. 602–624.
[15] H. W. KUHN, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quarterly, 2 (1955), pp. 83–97.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 27–66

TRAVELING SALESMAN-BASED CURVE RECONSTRUCTION IN POLYNOMIAL TIME*

ERNST ALTHAUS† AND KURT MEHLHORN†

Abstract. An instance of the curve reconstruction problem is a finite sample set V of an unknown collection of curves γ. The task is to connect the points in V in the order in which they lie on γ. Giesen [*Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SCG '99)*, 1999, pp. 207–216] showed recently that the traveling salesman tour of V solves the reconstruction problem for single closed curves under otherwise weak assumptions on γ and V; γ must be a single closed curve. We extend his result along several directions:

- we weaken the assumptions on the sample;
- we show that traveling salesman-based reconstruction also works for single open curves (with and without specified endpoints) and for collections of closed curves;
- we give alternative proofs; and
- we show that in the context of curve reconstruction, the traveling salesman tour can be constructed in polynomial time.

Key words. traveling salesman, polynomial time, curve reconstruction

AMS subject classifications. 68Q25, 05C85

PII. S0097539700366115

1. Introduction.

An instance of the curve reconstruction problem is a finite sample set V of an unknown collection of curves γ. The task is to construct a graph G on V so that two points in V are connected by an edge of G iff the points are adjacent on γ. The curve reconstruction problem and the related surface reconstruction problem have received a lot of attention in the graphics and the computational geometry community. We are interested in *reconstruction algorithms with guaranteed performance*, i.e., algorithms which provably solve the reconstruction problem under certain assumptions on γ and V. Figure 1.1 illustrates the curve reconstruction problem.

Many curve reconstruction algorithms have been proposed in the past; we restrict our discussion to algorithms that provably solve the reconstruction problem for a certain class of curves and under certain assumptions on the sample set. The algorithms differ with respect to the following aspects:

- whether a collection of curves or just a single curve can be handled;
- whether (collections of) open and closed curves can be handled or only (collections of) closed curves;
- whether the sampling must be uniform or not. Uniform sampling with density ϵ requires that the sample set V contains at least one point from every curve segment of length ϵ. In nonuniform sampling, the sampling frequency may depend on local properties of the curve, e.g., can be lower in parts of low curvature;
- whether nonsmooth curves can be handled or not. A smooth curve has a

*Received by the editors February 1, 2000; accepted for publication (in revised form) November 20, 2000; published electronically May 31, 2001. This paper appeared in preliminary form in the *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, January 2000. This research was partially supported by EG-projects ALCOM-IT and GALIA.
http://www.siam.org/journals/sicomp/31-1/36611.html
†Max-Planck-Institute für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (althaus@mpi-sb.mpg.de, mehlhorn@mpi-sb.mpg.de).

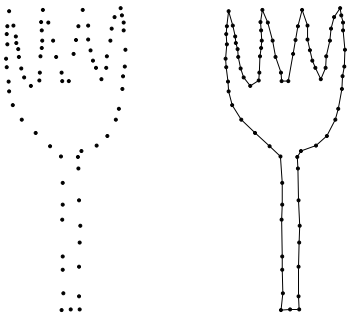


FIG. 1.1. *Part* (a) *shows a finite set V of points; part* (b) *shows the traveling salesman tour of the points.*

tangent everywhere. Figure 2.1 shows nonsmooth curves.

For uniformly sampled collections of closed, smooth curves several methods are known to work ranging over minimum spanning trees [12], α-shapes [5, 11], β-skeletons [16], and r-regular shapes [4]. A survey of these techniques appears in [10]. The case of nonuniformly sampled collections of closed smooth curves was first treated successfully by Amenta, Bern, and Eppstein [3], and subsequently improved algorithms such as [8, 15] appeared. Nonuniformly sampled collections of open and closed smooth curves were treated in [9]. All papers mentioned so far require the curves to be smooth.

Giesen [14] recently obtained the first result for nonsmooth curves. He considered the class of *benign semiregular curves*. An (open or closed) curve is *semiregular* if a left and a right tangent exists in every point of the curve; the two tangents may, however, be different. A semiregular curve is *benign* if the turning angle at every point of the curve is less than π; see Figure 2.1. Giesen showed that the traveling salesman tour of the sample set V solves the curve reconstruction problem for uniformly sampled benign closed semiregular curves. More precisely, he showed that for every benign semiregular closed curve γ there exists a positive ϵ such that the optimal traveling salesman tour of V is a polygonal reconstruction of γ provided that for every $x \in \gamma$ there is a $p \in V$ with $\|xp\| \leq \epsilon$, where $\|xy\|$ is the Euclidean distance of the two points x and y. Giesen's result is an existence result; he did not quantify ϵ in terms of properties of the curve γ. We extend Giesen's result in several directions:

- We relate ϵ to local properties of the curve γ and show that the optimal traveling salesman tour solves the reconstruction problem even if sampling is nonuniform. For smooth curves our sampling condition is similar to the one used in [3, 8, 15, 9].
- We show that the traveling salesman path is able to reconstruct open curves for a suitable sampling condition. We treat the case of paths with and without specified endpoints.
- We show that the optimal traveling salesman tour (path) can be constructed in polynomial time if our sampling condition is satisfied.
- We give a simplified proof that the traveling salesman tour (path) solves the curve reconstruction problem.
- We show that an extension of the traveling salesman tour algorithm is able to reconstruct nonuniformly sampled collections of closed nonsmooth curves.

We have implemented a number of curve reconstruction algorithms. The JAVA-Applet `http://review.mpi-sb.mpg.de:81/Curve-Reconstruction/` makes our implemen-

tations available. Our experiments show that the traveling salesman-based curve reconstruction is able to solve the reconstruction problem for surprisingly small sampling density and that its speed is comparable to Delaunay diagram-based reconstruction algorithms. The details are given in the companion paper [2].

A preliminary version of the present paper appeared in SODA 2000. It contained the result for open curves with specified endpoints and for closed curves. It did not contain the result for open curves without specified endpoints and for collections of closed curves. Also the proof of polynomiality was incomplete.

Since the publication of the conference version of this paper, Funke and Ramos [13] have presented an algorithm that works for collections of nonsmooth curves. The algorithm is based on filtering the Delaunay diagram and can handle collections of open and closed curves; our algorithm can only handle collections of closed curves.

This paper is structured as follows. In section 2 we give definitions and state our main results. In sections 3, 4, and 5 we prove our main theorem for open curves with and without specified endpoints and for closed curves, respectively, assuming real arithmetic. The result is extended to finite precision arithmetic in section 6. In section 7 we discuss collections of closed curves. For computational purposes it is desirable to restrict the search for the reconstruction to a sparse graph defined on the sample set. In section 8, we show that the edges of the polygonal reconstruction are in the Delaunay diagram for a slightly strengthened sampling condition. In section 10 we relate our sampling condition to the sampling conditions used by other papers, and in Section 11 we relate our result to so-called necklace tours. Necklace tours are a polynomially solvable case of the traveling salesman problem. Curve reconstruction problems are allowed to "invent" curves if the input does not satisfy the required sampling condition. This issue is discussed in section 9. Finally, in section 12 we offer conclusions and state open problems.

2. Definitions and statements of results. An open *curve* is given by an *embedding* $\gamma : [0,1] \to \mathbb{R}^2$ and a closed curve is given by an embedding $\gamma : S^2 \to \mathbb{R}^2$, where S^2 is the unit circle.

DEFINITION 2.1 (see [14]). *Let*

$$T = \{\, (t_1, t_2) \ ; \ t_1 < t_2, t_1, t_2 \in [0,1] \,\}$$

and

$$\tau : T \to S^2, (t_1, t_2) \mapsto \frac{\gamma(t_2) - \gamma(t_1)}{|\, \gamma(t_2) - \gamma(t_1) \,|}.$$

The curve γ is called left (right) regular at $\gamma(t_0)$ with left (right) tangent $\mathrm{t}_l(\gamma(t_0))$ ($\mathrm{t}_r(\gamma(t_0))$) if for every sequence (ξ_n) in T which converges to (t_0, t_0) from the left (right) in the closure of T the sequence $\tau(\xi_n)$ converges to $\mathrm{t}(\gamma(t_0))$. We call γ semiregular if it is left and right regular in all points $\gamma(t)$, $t \in [0,1]$. We call γ regular if it is semiregular and the left and right tangent coincide in every point of the curve.

Figure 2.1 shows two semiregular curves. Tangents are unit vectors. The angle between two vectors with the same source is the smaller of the two angles between the vectors. The angle is zero if the two vectors point to the same direction and the angle is π if the vectors point to opposite directions. The angle between the left and right tangent at a point $p \in \gamma$ is called the *turning angle* at p. If the curve has a tangent at p, the turning angle at p is zero. If the turning angle at p is nonzero, we call p a *singularity* of the curve. A semiregular curve is *benign* if the turning angle is less than π at every point of the curve.

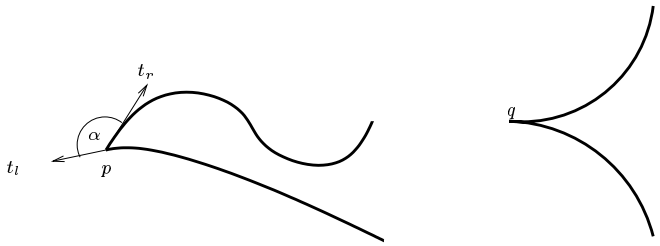


FIG. 2.1. *Two semiregular curves, one benign and one not. In the left curve, the two tangents t_l and t_r at a point p of the curve are shown. The turning angle at p is α. In the curve on the right the turning angle at q is π as the left and right tangents at q point to opposite directions.*

A traveling salesman path (tour) for a set V of points is a path (cycle) passing through all points in V. An optimal traveling salesman path (tour) is a traveling salesman path (tour) of shortest length. An optimal traveling salesman path with specified endpoints a and b, where $a \in V$ and $b \in V$ is a shortest traveling salesman path among the paths with endpoints a and b. We can now state Giesen's result.

THEOREM 2.2 (see [14]). *For every benign semiregular closed curve γ there exists an $\epsilon > 0$ with the following property: If V is a finite sample set of γ so that for every $x \in \gamma$ there is a $p \in V$ with $\|pv\| \leq \epsilon$, the optimal traveling salesman tour is a polygonal reconstruction of γ.*

The construction of optimal traveling salesman paths or tours is an NP-hard problem. A successful method for solving the Traveling Salesman problem is to formulate the problem as an integer linear program (ILP) and to use a branch-and-cut algorithm based on the LP-relaxation of the problem. We give the formulation for traveling salesman paths with specified endpoints a and b. We introduce a variable x_{uv} for every edge uv between two sample points and describe the set of all paths with endpoints a and b in the following way:

$$\sum_{v \in V} x_{uv} = 2 \text{ for all } u \in V \setminus \{a, b\},$$

$$\sum_{v \in V} x_{uv} = 1 \text{ for } u \in \{a, b\},$$

$$\sum_{u \in V', v \in V'} x_{uv} \leq |\, V' \,| - 1 \text{ for } V' \subset V,\ V' \neq \emptyset,$$

$$x_{uv} \in \{0, 1\} \text{ for all } u, v \in V.$$

We refer to this program as the *subtour-ILP for the traveling salesman path problem with specified endpoints*. Slight variations of this ILP characterize paths with unspecified endpoints (see section 4) and tours (see section 5). The equality constraints in the subtour-ILP are called *degree constraints*, the inequality constraints are called *subtour elimination constraints*, and the constraints $x_{uv} \in \{0, 1\}$ are called the *integrality constraints*. Relaxing the integrality constraints to $0 \leq x_{uv} \leq 1$ gives the *subtour-LP for the traveling salesman problem with specified endpoints*. The objective function for both programs is $\sum_{uv} \|uv\| x_{uv}$, i.e., the total Euclidean length of the edges selected.

In general, the optimal solution of the subtour-LP is fractional. Our main result states the optimal solution of the subtour-LP is integral, whenever V is a sufficiently dense sample of a benign semiregular curve. The reader might be interested to know

that we discovered this fact in our experiments on curve reconstruction. We had implemented a branch-and-cut algorithm based on the subtour-LP (the algorithm solved the subtour-LP and was then supposed to branch on a fractional variable) and observed that the algorithm never branched. After seeing this behavior in a large number of examples, we formulated it as a conjecture and set out to prove it.

THEOREM 2.3 (main theorem). *Let γ be an closed benign semiregular curve; let V be a finite set of samples of γ. If V satisfies the sampling condition given below, then*

- *the optimal traveling salesman tour of V is a polygonal reconstruction of γ;*
- *the subtour-LP for traveling salesman tours has an optimal integral solution and this solution is unique.*

In the case of an open curve let a and b be the first and last sample points, respectively (in the order on γ). The statements above hold true for the optimal traveling salesman path with endpoints a and b and the subtour-LP for traveling salesman paths with specified endpoints and also for the traveling salesman path with unspecified endpoints and for the subtour-LP for traveling salesman paths with unspecified endpoints. The latter result required a strengthened sampling condition.

We will prove our Main Theorem for open curves with specified and unspecified endpoints in sections 3 and 4, respectively. The proof for closed curves follows in section 5.

Our main theorem suggests a reconstruction algorithm for benign semiregular curves: *Solve the subtour-LP. If the optimal solution is integral, output it.* We briefly discuss two strategies for solving the subtour-LP.

A potentially exponential, but practically very efficient algorithm uses the simplex method and the cutting plane framework. One starts with the LP consisting only of the degree constraints and then solves a sequence of LPs. In each iteration one checks whether the solution X^* to the current LP satisfies all subtour elimination constraints and, if not, one adds a violated subtour elimination constraint to the LP. The check for a violated subtour elimination constraint is tantamount to a min-cut problem. One assigns capacity x_e^* to edge e for every edge e and computes a minimum (a, b)-cut. If the cut has value less than two, a violated inequality has been found. If the cut has value two, all cut constraints are satisfied. We use the simplex-based strategy in our in our experiments on curve reconstruction.

The ellipsoid method (see [18]) solves the subtour-LP in time polynomial in the size of the bit representations of the coefficients of the cost function. Distance values are, in general, nonrational numbers and hence the ellipsoid method is not directly applicable in our setting. In section 6, we extend our results to the situation where the position of the points and the distances between points are only approximately known and show how to obtain a polynomial time algorithm.

3. Open curves. We assume that our open curve is oriented and write $p < q$ if p precedes q on γ. We use $B(p, r)$ and $B^0(p, r)$ to denote the closed and open ball with center p and radius r, respectively.

3.1. The Held–Karp bound. Our proof of Theorem 2.3 exploits the connection between the subtour-LP and what is known as the Held–Karp bound. The purpose of this section is to review the relevant facts about the Held–Karp bound.

Let $G = (V, E)$ be an undirected graph, let a and b be two designated vertices of G, and let c be an arbitrary cost function on the edges of G. A function $\mu : V \to \mathbb{R}$ is called a *potential function*. It gives rise to a modified distance function c_μ via $c_\mu(u, v) = c(u, v) - \mu(u) - \mu(v)$. Now consider any traveling salesman path

T with endpoints a and b. Its costs under c and c_μ are related by $c_\mu(T) = c(T) - 2\sum_{v\in V}\mu(v) + \mu(a) + \mu(b)$ since the path uses two edges incident to every vertex except for a and b. Observe that $c_\mu(T) - c(T)$ does not depend on T and hence the optimal traveling salesman path for endpoints a and b is the same under both cost functions. Let T_0 be an optimal traveling salesman path for endpoints a and b.

Let MST_μ be a minimum spanning tree with respect to the cost function c_μ and let $C_\mu = c_\mu(\mathrm{MST}_\mu)$ be its cost. Then $C_\mu \le c_\mu(T_0)$, since a traveling salesman path is a spanning tree.

FACT 1. *Let μ be any potential function. If MST_μ is a traveling salesman path with endpoints a and b, it is an optimal Traveling Salesman path for endpoints a and b.*

Proof. From $C_\mu \le c_\mu(T_0)$, we conclude that MST_μ is an optimal traveling salesman path with respect to c_μ. Since the ranking of paths is the same under both cost functions, it is also optimal with respect to c. □

The inequality $C_\mu \le c_\mu(T_0) = c(T_0) - 2\sum_{v\in V}\mu(v) + \mu(a) + \mu(b)$ is valid for every potential function and hence

$$\max_\mu C_\mu + 2\sum_{v\in V}\mu(v) - \mu(a) - \mu(b) \le c(T_0)\ .$$

The quantity on the left is called the Held–Karp bound.[1] The following fact is crucial for our proof.

FACT 2. *The Held–Karp bound is equal to the optimal objective value of the subtour-LP.*

Proof. The fact follows by relaxing the degree constraints of the subtour-LP in a Lagrangian fashion. For a short introduction to Lagrangian relaxation, see [7, p. 259]. □

We remark (but will not use) that an optimal choice of μ in the Held–Karp bound is given by the optimal solution of the linear programming dual of the subtour-LP; μ corresponds to the dual variables for the degree constraints. We next draw a simple consequence from the two facts above, which forms the basis for our proof.

LEMMA 3.1. *Let μ be any potential function. If MST_μ is the unique minimum spanning tree with respect to c_μ and moreover is a traveling salesman path with endpoints a and b, the subtour-LP has a unique optimal solution and this solution is integral.*

Proof. If MST_μ is a traveling salesman path, it is optimal (Fact 1) and hence $c(\mathrm{MST}_\mu) = c(T_0)$. The Held–Karp bound is therefore equal to $c(T_0)$ and the same holds true for the optimal objective value of the subtour-LP (Fact 2). The incidence vector of MST_μ is a feasible solution of the subtour-LP of cost $c(\mathrm{MST}_\mu)$ and hence is *an* optimal solution of the subtour-LP. We will next argue that it is the unique optimal solution. Assume that there is an optimal solution of the subtour-LP with $x_e > 0$ for some $e \notin \mathrm{MST}_\mu$. Since MST_μ is unique there is a $\eta > 0$ so that decreasing the cost of e by η will not change the minimum spanning tree and hence will not change the

[1] A simple iterative algorithm can be used to approximate the Held–Karp bound. Let α_t, $t = 0, 1, \ldots$, be positive reals with $\alpha_t \to 0$ as $t \to \infty$ and $\sum_t \alpha_t = \infty$. Start with $\mu(v) = 0$ for all $v \in V$ and compute MST_μ. As long as MST_μ is not a path with endpoints a and b, update μ as follows: $\mu(v) = \mu(v) + \alpha_t \cdot (d(v) - deg_\mu(v))$, where $d(a) = d(b) = 1$ and $d(v) = 2$ for all other nodes, t is the index of the iteration, and $deg_\mu(v)$ is the degree of v in MST_μ. The cost of MST_μ converges to the Held–Karp bound. We found in our experiments that the convergence is rather slow and that the cutting plane approach to solving the subtour-LP is faster.

value of the Held–Karp bound. However, the objective value of the subtour-LP will decrease. This is a contradiction to the equality of the two bounds. □

We can now describe our proof strategy for open curves. We define a potential function μ such that MST_μ is the unique minimum spanning tree in the complete network $G = (V, V \times V, c)$, where c is the Euclidean distance function, and moreover MST_μ coincides with the polygonal reconstruction (and hence is a traveling salesman path with endpoints a and b, where a and b are the first and last sample point, respectively). Then MST_μ and hence the polygonal reconstruction is the unique optimal solution of the subtour-LP. We want to stress that the definition of μ is only needed for the proof of our main theorem. The reconstruction algorithm simply solves the subtour-LP.

3.2. Intuition. When will the minimum spanning tree of the sample set be the correct reconstruction? Let $V = \{v_1, v_2, \ldots, v_n\}$, where we assume the points to be numbered according to their order on the curve. Kruskal's algorithm considers the edges $v_i v_j$ in increasing order of length and adds an edge to the spanning tree if it does not close a cycle. Kruskal's algorithm will therefore construct the path v_1–v_2–$\cdots$–v_n if the potential function μ is such that

$$c_\mu(v_i, v_{i+1}) < c_\mu(v_h, v_j) \quad \text{whenever} \quad h \le i < j, j - h \ge 2\ . \tag{3.1}$$

Let us consider two special situations: γ is essentially straight (any two left or right tangents to γ form an angle of less than $\pi/3$) or γ consists of a sharp corner (a point in which γ turns by at least $7\pi/24$) and two incident straight line segments. See Figure 3.1. We will show in section 3.3 that any curve γ can be decomposed into subcurves which are either essentially straight or which consist of a sharp corner with two incident essentially straight legs.

For an essentially straight curve the minimum spanning tree will reconstruct for a large choice of potential functions. It will work without a potential function, i.e., $\mu(p) = 0$ for all p, and, more generally, it will work for any potential function that does not change too fast as a function of the position of its argument. For a point p which belongs to an essentially straight part of γ, we will essentially[2] define $\mu(p) = d(p)/3$, where $d(p)$ is maximal such that $B^0(p, d(p)) \cap \gamma$ is connected and essentially straight and $B^0(p, r)$ denotes the open ball with center p and radius r. This choice guarantees that $\mu(p)$ changes slowly with the position of its argument (with at most one third of the change in argument) and that $\mu(p)$ depends on local properties of the curve and is large in parts of γ that are intuitively simple to reconstruct. For sharp corners, the definition above leads to a potential value of zero.

Corners with a turning angle of more than $\pi/2$ will confuse the minimum spanning tree when used without a potential function as Figure 3.1 shows. One of our insights is that a simple potential function can be used to make the minimum spanning trees work. Assume that our curve consists of the two line segments $y = \pm m \cdot x$ for $0 \le x \le 1$ and let V be a finite set of samples. We define the potential as a function of the x-coordinates of the sample points. Fix $\pi(O)$ arbitrarily, let $p(x) = (x, mx)$ and $q(x) = (x, -mx)$ and define $\pi(p(x)) = \pi(q(x)) = \pi(O) - x$. Then $c_\pi(p(x), p(x_0)) = \sqrt{1 + m^2}\,|\,x - x_0\,| - 2\pi(O) + x_0 + x$ and $c_\pi(q(x), p(x_0)) = \sqrt{(x - x_0)^2 + (mx + mx_0)^2} - 2\pi(O) + x_0 + x$. It is an easy exercise in calculus to show that $c_\pi(p(x), p(x_0))$ is an increasing function of $|\,x - x_0\,|$ and that $c_\pi(q(x), p(x_0))$ is an increasing function of x. We conclude that the minimum spanning tree for the modified distance function

[2]The precise definition given in section 3.4 is somewhat more involved.

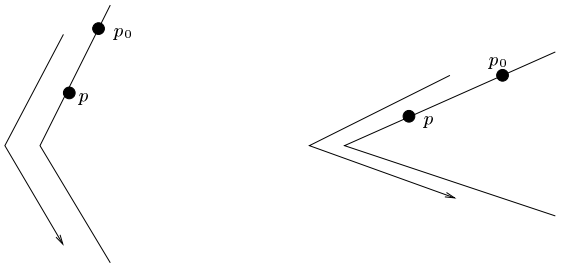


FIG. 3.1. *In the figure on the left, the Euclidean distance between p and p_0 grows as p moves away from p_0 along γ. In the figure on the right, the distance first grows and then shrinks again as p moves around the corner. The minimum spanning tree with $\mu(p) = 0$ for all p will reconstruct the curve on the left, but may fail on the curve on the right.*

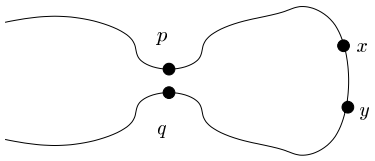


FIG. 3.2. *The edge pq does not belong to the polygonal reconstruction. Our definition of the potential function ensures that $c_\mu(p,q) > 0$. Our sampling condition is that any edge in the polygonal reconstruction has non-positive reduced cost.*

reconstructs. In the argument above the choice of $\pi(O)$ is arbitrary. The actual choice of $\pi(O)$ will depend on local properties of the curve. For every sharp corner s we will consider the largest open disk $B^0(s, c_s)$ such that $\gamma \cap B^0(s, c_s)$ is connected and is essentially a sharp corner with two incident straight legs. We set $\pi(O) = c_s$.

We can now sketch our definition of μ. For every point $p \in \gamma$, the first definition $\mu(p) = d(p)/3$ is applicable. It assigns potential zero to sharp corners. Near sharp corners we use the second definition, namely, $\mu(p) = c_s - \|sp_s\|$, where p is near the sharp corner s and p_s is the projection of p onto the angular bisector of the two tangents at s; see Figure 3.6.

The analysis above suggests that with this definition of μ, the minimum spanning tree solves the reconstruction task locally, i.e., if given the points in $V \cap \gamma'$, where γ' is a subcurve of γ that is either essentially straight or a sharp corner with two incident straight legs. In other words, inequality (3.1) holds if v_h,v_i, and v_j belong to the same γ'.

Does it also hold globally? Consider the situation shown in Figure 3.2. We have two points p and q that belong to distinct essentially straight parts of γ. We have $\max(d(p), d(q)) \leq \|pq\|$ and hence $c_\mu(p,q) > 0$. More generally, we will show in section 3.3 that any edge pq, where p and q do not belong to an either essentially straight subcurve or to a sharp corner with its incident legs, has positive modified cost.

The previous paragraph suggests our sampling condition. We require that any edge of the polygonal reconstruction has nonpositive reduced cost. Then (3.1) holds certainly when $c_\mu(v_j, v_j) > 0$. When $c_\mu(v_h, v_j) \leq 0$, v_h and v_j are guaranteed to lie in a common essentially straight subcurve or near a common sharp corner, and the local analysis applies.

This ends the informal description of our proof.

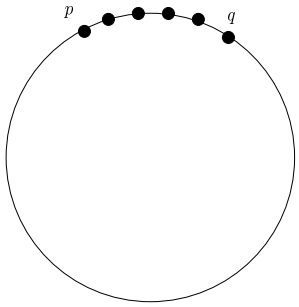


FIG. 3.3. *p and q are adjacent sample points and $c_\mu(p,q) \leq 0$ if p and q are sufficiently close. However, the sample set contradicts our intuition of what constitutes a dense sample set. Condition* (b) *excludes the case.*

3.3. The sampling condition and the global reasoning for open curves. We will give the detailed definition of our potential function in section 3.4; it assigns a positive real $\mu(x)$ to every point x of γ. Define the turning angle of a subcurve γ' of γ as the opening angle of the smallest double-cone that contains all (left and right) tangents to points $p \in \gamma'$. We require the following.

Sampling condition.

(a) For any two adjacent (on γ) samples u and v: $c_\mu(u,v) \leq 0$.

(b) For any two adjacent samples u and v: $\gamma[u,v]$ turns by less than π. For two adjacent points p, q on the curve, $\gamma[p,q]$ denotes the subcurve of γ with endpoints p and q not containing an other sample point. (In the case of closed curves we always have at least 3 sample points.)

Condition (a) implies condition (b) in the case of open curves as we will see below. For closed curves, condition (a) does not suffice as the example in Figure 3.3 indicates. Condition (a) states that adjacent sample points must be sufficiently close in a metric sense and condition (b) states that the curve must not turn too much between adjacent sample points.

The sampling condition is easily satisfied. Let $\epsilon = \inf_{x\in\gamma} \mu(x)$. Then $\epsilon > 0$ since γ is compact and hence a sample set in which γ turns by less than π between adjacent samples and in which there is at least one sample point in every curve segment of length $\epsilon/2$ satisfies the sampling condition. We want to stress that the sampling condition can also be satisfied with nonuniform sampling. In regions of γ where μ is large, the sampling may be less dense than in regions where μ is small. In section 10 we will relate our sampling condition to the conditions used in other papers on curve reconstruction.

In order to show that MST_μ is the polygonal reconstruction of γ from V, we define a family Γ of (overlapping) subcurves γ' of γ so that

(P1) each subcurve γ' is connected and the minimum spanning tree (with respect to cost function c_μ) of the points in $V \cap \gamma'$ is unique and coincides with the polygonal reconstruction of γ';

(P2) for every edge e with $c_\mu(e) \leq 0$ there is a subcurve $\gamma' \in \Gamma$ containing both endpoints of e.

We show that these conditions imply that MST_μ is the polygonal reconstruction of γ.

LEMMA 3.2. *Conditions* (P1) *and* (P2) *imply that MST_μ is the polygonal reconstruction of γ.*

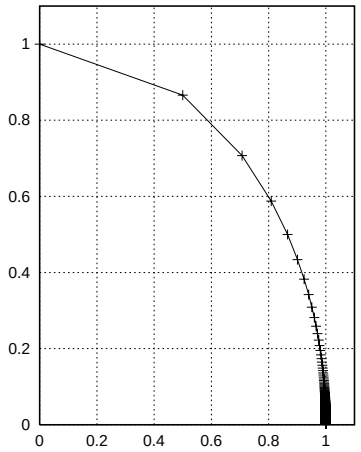


FIG. 3.4. *A semiregular curve with an infinite number of corners.*

Proof. Let MST_μ be any minimum spanning tree in (V, E, c_μ) and let $e = uv$ be any edge which does not belong to the polygonal reconstruction. We show that $e \notin \mathrm{MST}_\mu$. Observe first that any edge in the minimum spanning tree has nonpositive modified cost since there is a spanning tree, namely, the polygonal reconstruction, in which every edge has nonpositive cost. This follows from the cycle rule for minimum spanning trees. So assume $c_\mu(e) \le 0$. Then there is a subcurve $\gamma' \in \Gamma$ containing both endpoints of e by (P2). We even have $\gamma[u, v] \subseteq \gamma'$ since γ' is connected by (P1). Moreover, the minimum spanning tree of $V \cap \gamma'$ is unique and coincides with the polygonal reconstruction of γ'. Thus $c_\mu(e') < c_\mu(e)$ for every edge e' on the part of the polygonal reconstruction between u and v. We conclude that $e \notin \mathrm{MST}_\mu$. □

3.4. The definition of the potential function. In this section, we give the precise definition of our potential function. The definition depends on the parameters $\theta_{max\_sharp}, \theta_{turn}, f_{scale}, f_{wriggle}, \theta_{wriggle}$, and f_{shrink} whose choice is somewhat arbitrary but not completely independent. In section 3.6 we summarize the conditions.

Singularities cause difficulties for most curve reconstruction algorithms; the difficulties grow with the turning angle. We call a singularity p a *sharp corner* if the turning angle at p is at least $\theta_{max\_sharp} = 7\pi/24 = 52.5°$.

A semiregular curve may have an infinite number of singularities. For example, the convex hull of the points $(\cos(\pi/n), \sin(\pi/n))$, $n \ge 2$, has an infinite number of singularities; see Figure 3.4. However, a semiregular curve can have only a finite number of sharp corners. Assume otherwise. Then the sharp corners have an articulation point p. Let $p_1, p_2, \ldots$ be a increasing sequence of sharp corners converging to p. For any sharp corner p_i we can choose two points q_i and r_i in the vicinity of the corner so that the tangents at q_i and r_i form an angle of at least $\pi/6$ and so that the sequence $q_1, r_1, q_2, r_2, \ldots$ increases and converges to p. The sequence shows that γ has no left tangent at p.

We use S to denote the set of sharp corners of γ.

We are now ready to define our potential function. The definition consists of two parts dealing with the neighborhoods of sharp corners and curve parts "far away" from all sharp corners, respectively. We start with the latter parts.

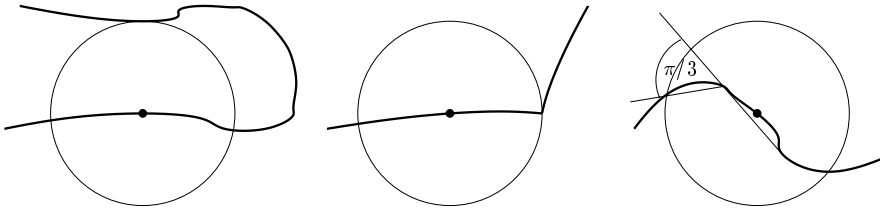


FIG. 3.5. *The closed ball intersects γ in more than one part or has a singularity on its boundary, or γ turns by $\pi/3$ in the ball.*

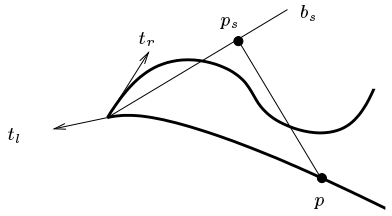


FIG. 3.6. *Illustration of the definition of b_s and p_s.*

For every point $p \in \gamma$ let $d(p)$ be maximal such that the open ball $B^0(p, d(p))$ has the following properties:

- $B^0(p, d(p)) \cap S = \emptyset$.
- $B^0(p, r) \cap \gamma$ is connected for all r with $r \leq d(p)$.
- $B^0(p, d(p)) \cap \gamma$ turns by less than $\theta_{turn} = \pi/3$.

We will define our potential in parts that are far away from sharp corners as $f_{scale} d(p)$ and choose $f_{scale} = 1/3$. We define later what we mean exactly with far away.

Observe that the closed ball $B(p, d(p))$ has one of the following properties: it has a point in S on its boundary or it intersects γ in more than one component or γ turns by $\pi/3$ in the ball (see Figure 3.5). For sharp corners $s \in S$ we have $d(s) = 0$ and for points $p \in \gamma \setminus S$ we have $d(p) > 0$. The function $p \mapsto d(p)$ is continuous. Thus $d(p)$ will be an increasing function of p as p moves away from a sharp corner for a neighborhood of any sharp corner. In section 10 we will relate $d(p)$ to the distance of p to the medial axis of γ.

For sharp corners we define a quantity δ_s. For a sharp corner $s \in S$ let b_s be the bisector of the angle between the right tangent and the reversal of the left tangent, let α_s be the turning angle at s (we have $\alpha_s \geq 7\pi/24$).

For an angle α, let $\bar{\alpha} = \pi - \alpha$. For a point $p \in \gamma$, let p_s be the orthogonal projection of p onto b_s. See Figure 3.6 for an illustration of these definitions. For every $s \in S$ let δ_s be maximal so that

- $\gamma \cap B^0(s, \delta_s)$ is connected. We call the two components of $(\gamma \setminus s) \cap B^0(s, \delta_s)$ the two legs of γ incident to s.
- The angle between any segment with both endpoints on one leg and the tangent in s of the same leg is less than $\min\{f_{wriggle} \bar{\alpha}_s, \theta_{wriggle}\}$. We choose $f_{wriggle} = 1/4$ and $\theta_{wriggle} = \pi/9$. The second bound guarantees that the angle between any segment and the perpendicular bisector is less than $\pi/2$.
- For either of the two legs $d(p)$ increases as p moves away from s.
- $B^0(s, 2\delta_s)$ contains no sharp corner different from s.

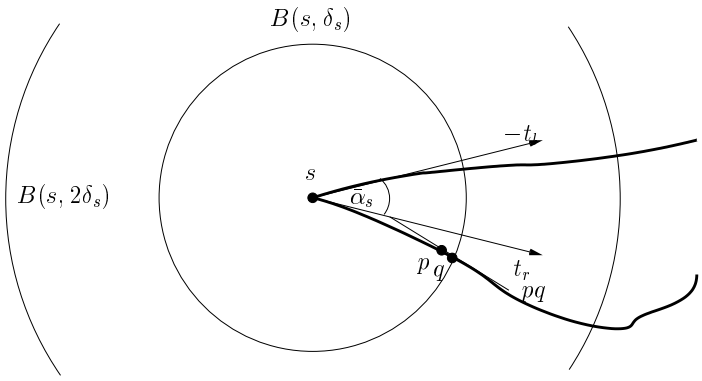


FIG. 3.7. *By the definition of δ_s, we know that γ is connected in $B^0(s,\delta_s)$, there is no sharp corner in $B^0(s,2\delta_s)$, the angle between a segment through two points of a leg in $B^0(s,\delta_s)$, say, pq, and the tangent of this leg, say, t_r, is less than* $\min(\alpha_s/4,\pi/9)$*, and the d values are increasing in $B(s,\delta_s)$.*

The last condition ensures that the balls $B^0(s,\delta_s)$, $s \in S$, are pairwise disjoint. For an illustration of the definition see Figure 3.7. Clearly, $\delta_s > 0$ for all $s \in S$.

Recall from section 3.2 that we want to define the potential near a sharp corner as $c_s - \|sp_s\|$. The change form the potential for sharp corners and smooth areas is made by choosing the maximum of the two possibilities. We use the constant c_s to specify the exact point where we change from one definition to the other. We choose c_s maximal under the restriction that all points outside the $B(s,\delta_s f_{shrink})$ ball ($f_{shrink} = 1/5$) have potential $d(p)/3$. Thus let q^1 and q^2 be the points where the two legs intersect the boundary of the circle $B(s,\delta_s/5)$ and let

$$c_s = \min\left\{ d(q^i)/3 + \|sq_s^i\| \;;\; i = 1,2 \right\}.$$

Then $c_s \le 2\delta_s/5$ since $d(q^i) \le \|sq^i\| = \delta_s/5$ and $\|q_s^i\| \le \|sq_i\| = \delta_s/5$.

Remark. In the proofs of section 3.5 we will use only the fact that $0 < c_s \le 2\delta_s/5$; the exact value of c_s does not matter. This will become important in section 5.

We are now ready to define our potential-function μ:

$$\mu(p) = \begin{cases} d(p)/3 & \text{if } p \text{ is in no } B(s,\delta_s), \\ \max\{c_s - \|sp_s\|, d(p)/3\} & \text{if } p \in B(s,\delta_s). \end{cases}$$

Observe that this definition "combines" the two cases discussed in section 3.2. We use T to denote the set of points $p \in \gamma$ with $\mu(p) = d(p)/3$, i.e., the points that are not affected by the singularities. Then $q^i \in T$ since $c_s \le d(q^i)/3 + \|sq_s^i\|$ and hence $c_s - \|sq_s^i\| \le d(q^i)/3$. Also since $d(p)$ and $\|sp_s\|$ increase as p moves away from s (at least as long as $p \in B^0(s,\delta_s)$), we have $p \in T$ for any curve point p that is not contained in $\cup_{s\in S} B^0(s,\delta_s/5)$. This also implies that μ is a continuous function. For a point $p \in B(s,\delta_s/5)$ we have $d(p) \le \delta_s/5$ and hence $\mu(p) \le c_s$.

We will frequently use the following simple observation.

LEMMA 3.3. *Let $u \in T \cap B(s,\delta_s)$ for some $s \in S$ and let v be a point on the other leg of s. Then $d(u) \le \|uv\|$.*

Proof. Assume otherwise, i.e., $\|uv\| < d(u)$. We have $d(u) \le \|us\|$ and hence $B^0(u,d(u)) \cap \gamma$ consists of at least two components, one containing u and one containing v. □

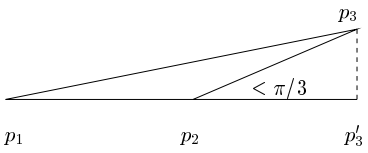


FIG. 3.8. *The situation in the proof of Lemma* 3.8. *We have* $\|p_1p_3\| \geq \|p_1p_3'\| > \|p_1p_2\| + \|p_2p_3\| \cos \pi/3$.

3.5. Local reasoning. We consider the following family Γ of subcurves:

1. $B^0(p, d(p)) \cap T \cap \gamma$ for all $p \in T$.
2. $B^0(s, \delta_s) \cap \gamma$ for all $s \in S$.

We call the subcurves of the first kind regular subcurves and the subcurves of the second kind singular subcurves.

LEMMA 3.4. *The subcurves* $\gamma' \in \Gamma$ *are connected.*

Proof. This is obvious for singular subcurves. So consider a subcurve $\gamma' = B^0(p, d(p)) \cap T \cap \gamma$ for some $p \in T$. The subcurve $B^0(p, d(p)) \cap \gamma$ is connected by definition. If γ' were not connected, $B^0(p, d(p)) \cap \gamma$ decomposes into three nontrivial segments γ_1, γ_2, and γ_3 with $\gamma_1 \cap T \neq \emptyset$, $\gamma_2 \cap T = \emptyset$, and $\gamma_3 \cap T \neq \emptyset$. This implies that γ_2 passes through a sharp corner, a contradiction to the definition of $d(p)$. ∎

LEMMA 3.5. *Let* u *and* v *be adjacent sample points and let* $\gamma' \in \Gamma$. *If* γ' *contains* u *and* v, *then* $\gamma[u, v] \subseteq \gamma'$.

Proof. γ' is connected and hence either $\gamma[u, v] \subseteq \gamma'$ or $\gamma \setminus \gamma[u, v] \subseteq \gamma'$. The latter case is impossible since $\gamma \setminus \gamma[u, v]$ is not connected in the case of an open curve and turns by more than π according to our second sampling condition in the case of a closed curve. However, γ' turns by less than π according to the definition of Γ. ∎

For open curves Lemma 3.5 holds true without the second sampling condition. Since curves $\gamma' \in \Gamma$ turn by less than π, the second sampling condition is implied by the first for open curves. We will next verify the properties (P1) and (P2).

LEMMA 3.6 (property (P2)). *Let* $e = pq$ *be an edge with nonpositive modified cost. Then there is a subcurve* $\gamma' \in \Gamma$ *containing* p *and* q.

Proof. We have $\|pq\| \leq \mu(p) + \mu(q)$ by assumption. If $p \in T$ and $q \in T$, $\mu(p) = d(p)/3$ and $\mu(q) = d(q)/3$ and hence $\|pq\| \leq (d(p) + d(q))/3 \leq 2 \max(d(p), d(q))/3$. Thus $\{p, q\} \subseteq B^0(x, d(x))$ for one of the endpoints x of e.

If one of the endpoints does not belong to T, say, $p \notin T$, then $p \in B^0(s, \delta_s/5)$ for some sharp corner. If $q \in B^0(s, \delta_s)$ we are done. Therefore, assume otherwise. We have $\mu(p) \leq 2\delta_s/5$ and $\|pq\| \geq 4\delta_s/5$. If $q \in T$, then $\mu(q) = d(q)/3 \leq \|sq\|/3 \leq (\|pq\| + \delta_s/5)/3$ and hence $\|pq\| \leq (\|pq\| + \delta_s/5)/3 + 2\delta_s/5$ or $2\|pq\|/3 \leq 7\delta_s/15$ or $\|pq\| \leq 21\delta_s/30$, a contradiction to $\|pq\| \geq 4\delta_s/5$. If $q \notin T$, then $q \in B^0(t, \delta_t/5)$ for some sharp corner t different from s and hence $\|pq\| > 4(\delta_s + \delta_t)/5$. But $\mu(p) \leq 2\delta_s/5$ and $\mu(q) \leq 2\delta_t/5$, a contradiction. ∎

In order to show that the MST_μ coincides with the polygonal reconstruction, we show that the modified distance between two points p and r is either nonnegative, or the modified distance from p to r is greater than the modified distance from p to any point q between p and r. Since we also want to use these lemmas if we treat the problem with finite precision arithmetic, we quantify the change of the modified distance in the distance of q and r.

LEMMA 3.7. *Let* p, q, *and* r *be sample points on a regular subcurve with* $p < q < r$. *If* $\{p, q, r\} \in B(t, d(t))$ *for some point* $t \in \gamma$, *then* $c_\mu(p, r) - c_\mu(p, q) \geq \|qr\|/6$.

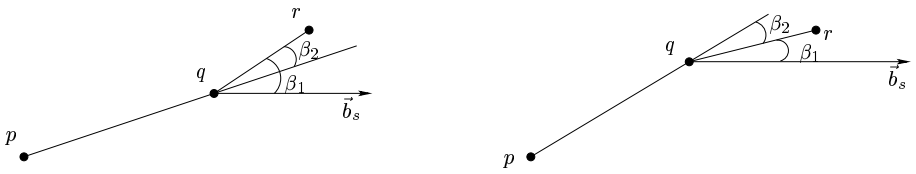


FIG. 3.9. *Illustration of the definition of β_1 and β_2, if p, q, and r are on the same leg.*

Proof. Let $t \in \gamma$ and $p < q < r \in B(t, d(t))$. Since the points are contained in a regular subcurve, we have (see Figure 3.8) $\angle(\vec{pq}, \vec{qr}) < \pi/3$ and hence $\|pr\| > \|pq\| + \|qr\| \cos \pi/3 = \|pq\| + \|qr\|/2$. Furthermore observe that $d(r) \leq d(q) + \|qr\|$. Thus

$$\begin{aligned} c_\mu(p,r) - c_\mu(p,q) &= \|pr\| - \mu(p) - \mu(r) - \|pq\| + \mu(p) + \mu(r) \\ &\geq \|qr\|/2 - \|qr\|/3 = \|qr\|/6. \qquad \square \end{aligned}$$

LEMMA 3.8 (property (P1) for regular regions). *MST_μ coincides with the polygonal reconstruction for regular subcurves.*

Proof. The lemma above for both possible orientations of the curve implies directly that Prim's minimum spanning tree algorithm finds the polygonal reconstruction. $\square$

LEMMA 3.9. *Let p, q, and r be sample points of γ with $p < q < r$. If $\{p, q, r\} \in B(s, \delta_s)$ for some sharp corner $s \in \gamma$, then $c_\mu(p,r) \geq 0$ or $c_\mu(p,r) - c_\mu(p,q) \geq \|qr\|(\sin \bar{\alpha}_s/4)^2/3$.*

Proof. We first argue that it suffices to prove the claim for the situations where either q and r both belong to T or neither of them does. Assume for the moment that those two cases have been dealt with. If exactly one of q and r belongs to T, there is a point u between q and r that belongs to the boundary of T. For this point, we have $d(u)/3 = c_s - \|su_s\|$ and hence u can be considered to be in T or outside T. The triples (p, q, u) and (p, u, r) are both in one of the special situations and hence we have $c_\mu(p,u) \geq 0$ or $c_\mu(p,u) - c_\mu(p,q) \geq \|qu\|(\sin \bar{\alpha}_s/4)^2/3$ and $c_\mu(p,r) \geq 0$ or $c_\mu(p,r) - c_\mu(p,u) \geq \|ur\|(\sin \bar{\alpha}_s/4)^2/3$. If $c_\mu(p,r) \geq 0$, we are done. Otherwise $c_\mu(p,r) < 0$ and hence $c_\mu(p,u) < 0$. Thus $c_\mu(p,r) - c_\mu(p,q) = c_\mu(p,r) - c_\mu(p,u) + c_\mu(p,u) - c_\mu(p,q) \geq (\|qu\| + \|ur\|)(\sin \bar{\alpha}_s/4)^2/3 \geq \|qr\|(\sin \bar{\alpha}_s/4)^2/3$, where the last inequality follows from the triangle inequality. We may from now on assume that q and r either both belong to T or neither of them does.

We need some further case distinctions. The first distinction is according to the sign of $c_\mu(p,q)$. The case $c_\mu(p,q) > 0$ is dealt with in the last paragraph of the proof.

We start with the assumption $c_\mu(p,q) \leq 0$. We make a further case distinction according to the position of s in the sequence $p < q < r$. In all four cases we employ a common strategy. We have $c_\mu(p,r) - c_\mu(p,q) = (\|pr\| - \|pq\|) - (\mu(r) - \mu(q))$. We bound $\|pr\| - \|pq\|$ from below and $\mu(r) - \mu(q)$ from above and estimate the difference of the bounds. In all cases we also use the estimates $\bar{\alpha}_s \leq 17\pi/24$, $\sin \bar{\alpha}_s/4 \leq 0.5281$, $(\sin \bar{\alpha}_s/4)^2/3 \leq 0.1$, and $\cos 2\pi/9 \geq 2/3$.

$s \leq p < q < r$: If $\{q, r\} \cap T = \emptyset$, $\mu(r) \leq \mu(q)$, and if $\{q, r\} \subseteq T$, $\mu(r) \leq \mu(q) + \|qr\|/3$. In either case[3], $\mu(r) \leq \mu(q) + \|qr\|/2$.

[3]In section 5.2 we will consider a modified potential function for which we know only $\mu(r) \leq \mu(q) + \|qr\|/2$. We want to reuse the proof there.

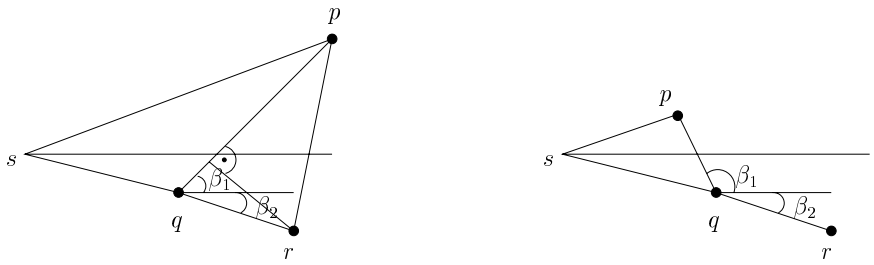


FIG. 3.10. *The case $p < s \leq q < r$: In the left part q_s is closer to s than p_s and in the right part the converse is true. In both situations the definitions of β_1 and β_2 are illustrated.*

Let β_2 be the angle between the vectors $\vec{pq}$ and $\vec{qr}$. Then $\|pr\| - \|pq\| \geq \|qr\| \cos \beta_2$. By the sampling condition, we have $\beta_2 \leq 2\pi/9$, since the angle between pr, respectively, qr, and the left tangent at s is at most $\pi/9$. Thus

$$c_\mu(p,r) - c_\mu(p,q) \geq \|qr\|(\cos(\beta_2) - 1/2) \geq \|qr\|/6 \geq \|qr\| \sin(\bar{\alpha}_s/4)^2/3 \ .$$

$p < q < r \leq s$: If $\{qr\} \subseteq T$, $\mu(r) \leq \mu(q) \leq \mu(q) + \|qr\|/2$. Let β_2 be the angle between the vectors $\vec{pq}$ and $\vec{qr}$. Then $\|pr\| - \|pq\| \geq \|qr\| \cos \beta_2$ and $\beta_2 \leq 2\pi/9$ and hence the argument used in the case $s \leq p < q < r$ applies.

Assume next that $\{q,r\} \cap T = \emptyset$. Let β_1 be the angle between the vectors $\vec{qr}$ and $\vec{b}_s$. Then $\mu(r) - \mu(q) = \|qr\| \cos \beta_1$. By the sampling condition, we have $\beta_1 \geq 3/4\bar{\alpha}_s$. Thus $\mu(r) - \mu(q) = \|qr\| \cos 3/4\bar{\alpha}_s$. Let β_2 as above be the angle between the vectors $\vec{pq}$ and $\vec{qr}$; see Figure 3.9 for an illustration. Then $\beta_2 \leq \bar{\alpha}_s/2$ (since the angle between pq, respectively, qr, and the right tangent at s is at most $\bar{\alpha}_s/4$) and $\|pr\| - \|pq\| \geq \|qr\| \cos \beta_2 \geq \|qr\| \cos \bar{\alpha}_s/2$.

Combining bounds we obtain

$$\begin{aligned} c_\mu(p,r) - c_\mu(p,q) &\geq \|qr\|(\cos(\bar{\alpha}_s/2) - \cos(\bar{\alpha}_s/2 + \bar{\alpha}_s/4)) \\ &\geq \|qr\|(\cos(\bar{\alpha}_s/2) - \cos(\bar{\alpha}_s/2)\cos(\bar{\alpha}_s/4) + \sin(\bar{\alpha}_s/2)\sin(\bar{\alpha}_s/4)) \\ &\geq \|qr\| \sin(\bar{\alpha}_s/4)^2. \end{aligned}$$

$p < s \leq q < r$: Assume first that $\{q,r\} \cap T = \emptyset$. If q_s is as least as far from s as p_s, we have $\|pr\| - \|pq\| \geq 0$ and $\mu(q) - \mu(r) \geq \|qr\| \cos(17\pi/48 + \pi/9) = \|qr\| \cos(\pi/2 - 5\pi/144) = \|qr\| \sin(5\pi/144) > 0.1 \cdot \|qr\| \geq \|qr\|(\sin \bar{\alpha}_s/4)^2/3$. The claim follows.

If q_s is closer to s than p_s, let β_1 be the angle between the vectors $\vec{qp}$ and $\vec{b}_s$ and β_2 be the angle between the vectors $\vec{qr}$ and $\vec{b}_s$; see the left part of Figure 3.10 for an illustration. We have $\mu(q) - \mu(r) = \|qr\| \cos \beta_2$ and $\|pq\| - \|pr\| \leq \|qr\| \cos(\beta_1 + \beta_2)$. By the sampling condition, we have $\beta_2 \geq \bar{\alpha}_s/4$ and $\beta_1 \geq \angle(\vec{sp}, \vec{b}_s) \geq \bar{\alpha}_s/4$, since $\angle(\vec{qp}, \vec{b}_s) \geq \angle(\vec{sp}, \vec{b}_s)$ (moving along the line $\vec{sq}$ increases the angle). Combining bounds we obtain

$$\begin{aligned} c_\mu(p,r) - c_\mu(p,q) &\geq \|qr\|(\cos \beta_2 - \cos(\beta_1 + \beta_2)) \\ &= \|qr\|(\cos \beta_2 - \cos \beta_1 \cos \beta_2 + \sin \beta_1 \sin \beta_2) \\ &\geq \|qr\| \sin(\bar{\alpha}_s/4)^2. \end{aligned}$$

We come to the case that $\{q,r\} \subseteq T$: If $p \in T$, we have $c_\mu(p,r) = \|pr\| - d(p)/3 - d(r)/3 > 0$, since $d(p) \leq \|pr\|$ and $d(r) \leq \|pr\|$.

So assume $p \notin T$. Since $q \in T$, we have $d(q)/3 \geq c_s - \|sq_s\|$ and hence $c_s \leq d(q)/3 + \|sq_s\|$. Thus[4] $0 \geq c_\mu(p,q) = \|pq\| - (c_s - \|sp_s\|) - d(q)/3 \geq \|pq\| - (\|sq_s\| - \|sp_s\|) - 2d(q)/3 \geq \|pq\|/3 - (\|sq_s\| - \|sp_s\|)$ and hence $\|sq_s\| - \|sp_s\| \geq \|pq\|/3$. In particular, q_s lies further away from s than p_s.

Let β_1 be the angle between the vectors $\vec{qp}$ and $\vec{b}_s$ and β_2 be the angle between the vectors $\vec{qr}$ and $\vec{b}_s$; see the right part of Figure 3.10 for an illustration. Then $\cos\bar{\beta}_1 \geq 1/3$, $\beta_2 \geq \bar{\alpha}_s/4$, and $\|pr\| - \|pq\| \geq \|qr\|\cos(\pi - \beta_1 - \beta_2)$.

Combining bounds we obtain $c_\mu(p,r) - c_\mu(p,q) \geq \|qr\|(\cos(\bar{\beta}_1 - \beta_2) - 1/3)$. If $\cos(\bar{\beta}_1 - \beta_2) - 1/3 \geq 4/27$, we are done since $4/27 \geq (\sin\bar{\alpha}_s/4)^2/3$. So assume $\cos(\bar{\beta}_1 - \beta_2) \leq 1/3 + 4/27 = 13/27 \leq 1/2$. Then $\bar{\beta}_1 - \beta_2 \geq \pi/3$ and hence $\bar{\beta}_1 \geq \pi/3$ and hence $(1 - \cos\bar{\beta}_1)/\bar{\beta}_1 \geq 1/2$. Since $\cos(\bar{\beta}_1 - x)$ is convex in $[0, \bar{\beta}_1]$ and hence is above the line through the points $(0,1)$ and $(\bar{\beta}_1, \cos\bar{\beta}_1)$, we have

$$\cos(\bar{\beta}_1 - \beta_2) - 1/3 \geq \cos\bar{\beta}_1 + (1 - \cos\bar{\beta}_1)(\beta_2/\bar{\beta}_1) - 1/3 \geq \beta_2/2 \geq (\sin\bar{\alpha}_s/4)^2/3.$$

$p < q < s < r$: We have $\|qr\| \leq \|qs\| + \|sr\|$. One of the previous cases applies to the triples (p,q,s) and (p,s,r) and hence $c_\mu(p,r) - c_\mu(p,q) = c_\mu(p,r) - c_\mu(p,s) + c_\mu(p,s) - c_\mu(p,q) \geq (\|qs\| + \|sr\|)(\sin\bar{\alpha}_s/4)^2/3 \geq \|qr\|(\sin\bar{\alpha}_s/4)^2/3$.

The discussion of the case $c_\mu(p,q) \leq 0$ is now completed. So let us assume $c_\mu(p,q) > 0$. If $c_\mu(p,r) < 0$ there is a point q' between q and r with $c_\mu(p,q') = 0$. The first case applies to the triple (p,q',r) and hence $c_\mu(p,r) > c_\mu(p,q')$, a contradiction. Thus $c_\mu(p,r) \geq 0$. □

LEMMA 3.10 (property (P1) for singular regions). *MST_μ coincides with the polygonal reconstruction for singular subcurves.*

Proof. The lemma above for both possible orientations of the curve implies directly that Prim's minimum spanning tree algorithm finds the polygonal reconstruction. □

3.6. Conditions on the thresholds. In the preceding sections we showed that properties (P1) and (P2) hold if one chooses the thresholds as in section 3.4. There are other possible choices for the thresholds that make the arguments work. We now collect the conditions on the thresholds. Note that the subtour-LP has an unique integral solution if there is a choice of the thresholds that make the MST_μ unique and equal to the polygonal reconstruction.

We introduced six thresholds:

θ_{max_sharp}: The minimum turning angle of a singularity, which we call sharp. We have chosen $\theta_{max\_sharp} = 7\pi/24$.

θ_{turn}: The maximal turning angle in a $B(p, d(p))$ ball. We have chosen $\theta_{turn} = \pi/3$.

f_{scale}: The factor by which we scaled the $d(p)$ value for the potential. We have chosen $f_{scale} = 1/3$.

$f_{wriggle}$: The factor by which we scaled the $\bar{\alpha}_s$ as maximal angle between a tangent and a segment in a sharp corner. We have chosen $f_{wriggle} = 1/4$.

$\theta_{wriggle}$: The maximal angle between a tangent and a segment in a sharp corner. We have chosen $\theta_{wriggle} = \pi/9$.

f_{shrink}: The factor by which we shrunk the δ_s ball to define the area where we use the potential function for sharp corners. We have chosen $f_{shrink} = 1/5$.

[4]Recall that we work under the assumption $c_\mu(p,q) \leq 0$.

First of all, we must guarantee that $\mu > 0$, which is equivalent to

$$\boxed{\theta_{turn} < \theta_{max\_sharp}.}$$

We start our investigations by looking at property (P2), i.e., to guarantee that all edges which are not contained in a region have positive modified cost. Look at any edge pq with negative reduced cost. If p and q are in T, we have to show that there is a point x with $\|px\| \leq d(x)$ and $\|qx\| \leq d(x)$. For simplicity we have assumed that x is either p or q. This is reasonable, since the conditions we get are weaker than the conditions we need later. Thus we require $\max(d(p), d(q)) \geq \|pq\|$. Since $\|pq\| \leq f_{scale}(d(p) + d(q))$ it suffices that

$$\boxed{f_{scale} < 1/2.}$$

If exactly one of p, q is not in T, we require that $2f_{shrink}\delta_s + f_{scale}\|sq\| \leq \|pq\|$. (The first summand is an upper bound for the potential of p, the second for the potential of q.) Using that $\|pq\| \geq 1 - f_{shrink}$ for all points outside the $B(s, \delta_s)$ ball, we conclude that it suffices that

$$\boxed{3f_{shrink} + f_{scale} < 1.}$$

Assume next that p and q are not in T. Then $p \in B(s_1, f_{shrink}\delta_{s_1})$ and $q \in B(s_2, f_{shrink}\delta_{s_2})$ for different sharp corners s_1 and s_2. Let without loss of generality (w.l.o.g.) $\delta_{s_1} \geq \delta_{s_2}$. It suffices that

$$\boxed{f_{shrink} < 1/3.}$$

Let us now turn to property (P1). In regular regions we require that "potential changes slower than the distance." Thus we need

$$\boxed{\cos(\theta_{turn}) > f_{scale}.}$$

For singular regions, we have to go through all cases of Lemma 3.9. If $s \leq p < q < r$ we need similarly to the case of regular cures

$$\boxed{\cos(\theta_{wriggle}) > f_{scale}.}$$

The same suffices in the case $p < q < r < s$ and q and r are in T.

If $p < q < r < s$ and q and r are not in T, we require that $\cos(1 - f_{wriggle}) < \cos(2f_{wriggle})$, which is equivalent to

$$\boxed{f_{wriggle} < 1/3.}$$

If $p < s \leq q < r$ and $q, r \notin T$, we additionally require that

$$\boxed{\theta_{max\_sharp}/2 + \theta_{wriggle} < \pi/2.}$$

If $q, r \in T$ we compute that $\cos(\bar{\beta}_1) \geq 1 - 2f_{scale}$ and require that $\cos(\bar{\beta}_1 - f_{wriggle}\alpha_s) > 1/3$. This is equivalent to

$$\boxed{f_{scale} \leq 1/3.}$$

Note that the last requirement is the only one where the condition can also be satisfied with equality, since we subtract a positive number in the derivation. This ends our discussion.

4. Open curves with unspecified endpoints. In the preceding sections we assumed that the first and last sample point (= endpoints of the traveling salesman path) are specified as part of the input. In this section we show that the subtour-LP can also reconstruct when the endpoints are not specified. Of course, the requirements on the sample will be stronger. The argument used in this section is a variant of the argument used in section 3.

We use the following formulation of the subtour-ILP. The goal is to select a total of $n-1$ edges such that at most two of them are incident to any node and such that no subset V' with $V' \neq \emptyset$ is "over-full."

$$\begin{aligned}
\min \quad & \sum_{u,v \in V} c_{uv} x_{uv} \\
\text{subject to (s.t.)} \quad & \sum_{v \in V} x_{uv} \leq 2 && \text{for all } u \in V, \\
& \sum_{u,v \in V'} x_{uv} \leq |V'| - 1 && \text{for } V' \subset V,\ V' \neq \emptyset, \\
& \sum_{u,v \in V'} x_{uv} = n-1, \\
& x_{uv} \in \{0,1\} && \text{for all } u, v \in V.
\end{aligned}$$

Selecting a total of $n-1$ edges such that at most two of them are incident to any node amounts to selecting a path and a set of cycles covering all nodes. The constraint that no set can be over-full implies that no cycles can be used and hence any solution must be a traveling salesman path. The subtour-LP is obtained by replacing the integrality constraints $x_{uv} \in \{0,1\}$ by the linear constraints $0 \leq x_{uv} \leq 1$.

As in the case of open curves with specified endpoints, we have to show that the separation problem can be solved in polynomial time. The separation algorithm works as follows. Let x^* be the optimal solution. We assign a capacity of x^*_e to edge e for every edge e. Furthermore we introduce a artificial vertex s and edges us with capacity $2 - \sum_{e \in \delta(u)} x^*_e$ for every node u of the graph. We compute a minimal cut in this graph and take as subset for the subtour elimination constraint the side of the cut that does not contain the artificial node s.

To see the correctness of this separation algorithm, we show that the subtour elimination constraint for S is violated iff the size of the cut is less than 2. Let $S \subset V$ be a subset of the nodes. First notice that the sum of the capacities of edges adjacent to a node is exactly 2 for every node in the above graph. Thus $2\sum_{e \in \gamma(S)} x^*_e + \sum_{e \in \delta(s)} x^*_e = 2\ S$. (For every node $u \in S$, we sum the capacities of the adjacent edges.) We conclude that $\sum_{e \in \delta(s)} x^*_e < 2$ iff $\sum_{e \in \gamma(S)} x^*_e >\ S\ - 1$.

We consider only nonpositive potential functions $\mu \leq 0$ in this section. Let a and b be fixed vertices and consider any traveling salesman path T with endpoints a and b. Its costs under c and c_μ are related by $c_\mu(T) = c(T) - 2\sum_{v \in V} \mu(v) + \mu(a) + \mu(b)$ since the path uses two edges incident to every vertex except for a and b. Observe that $c_\mu(T) - c(T)$ does not depend on T and hence the optimal traveling salesman path

for endpoints a and b is the same under both cost functions. However, the relative order of traveling salesman path with distinct endpoints is changed.

Let MST_μ be a minimum spanning tree with respect to the cost function c_μ and let $C_\mu = c_\mu(\mathrm{MST}_\mu)$ be its cost. Then $C_\mu \le c_\mu(T)$ for any traveling salesman path T.

FACT 3. *Let $\mu \le 0$ be any potential function. If MST_μ is a traveling salesman path and $\mu(a) = \mu(b) = 0$ for the endpoints of this path, it is an optimal traveling salesman path.*

Proof. Let T_0 be an optimal traveling salesman path, say, with endpoints u and v. Then $C_\mu \le c_\mu(T_0)$, since T_0 is a spanning tree, $c(\mathrm{MST}_\mu) = c_\mu(\mathrm{MST}_\mu) + 2\sum_{v\in V}\mu(v) - \mu(a) - \mu(b) = c_\mu(\mathrm{MST}_\mu) + 2\sum_{v\in V}\mu(v)$, since MST_μ is a path with endpoints a and b and a and b have potential zero, and $c(T_0) = c_\mu(T_0) + 2\sum_{v\in V}\mu(v) - \mu(u) - \mu(v) \ge c_\mu(T_0) + 2\sum_{v\in V}\mu(v)$, since T_0 is a path with endpoints u and v, and since the potentials of u and v are nonpositive. Thus

$$c(T_0) \ge c_\mu(T_0) + 2\sum_{v\in V}\mu(v) \ge c_\mu(\mathrm{MST}_\mu) + 2\sum_{v\in V}\mu(v) = c(\mathrm{MST}_\mu). \qquad \square$$

The inequality $C_\mu \le c_\mu(T_0) = c(T_0) - 2\sum_{v\in V}\mu(v) + \mu(u) + \mu(v) \le c(T_0) - 2\sum_{v\in V}\mu(v)$ is valid for every nonpositive potential function (the last inequality uses nonpositivity) and hence

$$\max_{\mu\le 0}\left(C_\mu + 2\sum_{v\in V}\mu(v)\right) \le c(T_0)\ .$$

The quantity on the left is called the Held–Karp bound. The following fact is crucial for our proof.

FACT 4. *The Held–Karp bound is equal to the optimal objective value of the subtour-LP.*

Proof. The proof follows from [7, p. 259]. Relaxing the degree constraints $\sum_{v\in V} x_{uv} \le 2$ in a Langrangian fashion, we obtain the problem

$$\begin{aligned}
\max_{\mu\le 0}\min_{x\ge 0} \quad & \sum_{u,v\in V} c(uv)x_{uv} + \sum_u \mu(u)\left(2 - \sum_{v\in V} x_{uv}\right) \\
\text{s.t.} \quad & \sum_{u,v\in V'} x_{uv} \le |V'| - 1 \quad \text{for } V' \subset V,\ V' \ne \emptyset, \\
& \sum_{u,v\in V} x_{uv} = n - 1, \\
& x_{uv} \le 1 \qquad \text{for all } u, v \in V.
\end{aligned}$$

Observe that we are only maximizing over nonpositive potential functions μ. This stems from the fact that in contrast to section 3, the degree constraints are now inequalities instead of equalities. The reformulation has the same objective value. The objective function of the LP can be reformulated as $\sum_{u,v\in V}(c(uv) - \mu(u) - \mu(v))x_{uv} + 2\mu(V)$. We conclude that the LP is simply a minimum spanning tree problem for the cost function c_μ. $\square$

We remark (but will not use) that the optimal choice of μ in the Held–Karp bound is given by the optimal solution of the linear programming dual of the subtour-LP; μ

corresponds to the dual variables for the degree constraints. We next draw a simple consequence from the two facts above.

LEMMA 4.1. *Let $\mu \leq 0$ be any potential function. If MST_μ is the unique minimum spanning tree with respect to c_μ, is a traveling salesman path, and $\mu(a) = \mu(b) = 0$ for its endpoints a and b, the subtour-LP has a unique optimal solution and this solution is integral.*

Proof. If MST_μ is a traveling salesman path, it is optimal (Fact 1) and hence $c(\mathrm{MST}_\mu) = c(T_0)$. The Held–Karp bound is therefore equal to $c(T_0)$ and the same holds true for the optimal objective value of the subtour-LP (Fact 2). The incidence vector of MST_μ is a feasible solution of the subtour-LP of cost $c(\mathrm{MST}_\mu)$ and hence is *an* optimal solution of the subtour-LP. We will next argue that it is the unique optimal solution. Assume that there is an optimal solution of the subtour-LP with $x_e > 0$ for some $e \notin \mathrm{MST}_\mu$. Since MST_μ is unique there is a $\eta > 0$ so that decreasing the cost of e by η will not change the minimum spanning tree and hence will not change the value of the Held–Karp bound. However, the objective value of the subtour-LP will decrease. This is a contradiction to the equality of the two bounds. □

It remains to define the appropriate potential function. We obtain it as a modification of the potential function defined in the preceding section. We use $\bar{\mu}$ to denote it. Let V be a set of sample points and let a and b be the first and the last sample point.

Let $m = \min(\bar{\mu}(a), \bar{\mu}(b))$ and set $c_s = \min(c_s, m)$ for all sharp corners. This changes $\bar{\mu}$. (It makes $\bar{\mu}$ smaller for some points near sharp corners.) Define new potential functions $\tilde{\mu}$ and μ by

$$\tilde{\mu}(p) = \min(\bar{\mu}(p), m) \quad \text{and} \quad \mu(p) = \tilde{\mu}(p) - m$$

for all $p \in \gamma$, i.e., first all potential values are capped at m and then m is subtracted. Then $\tilde{\mu}(p) \geq \mu(p)$ for all p and $\mu(p) \leq 0$ for all p. Also $\mu(a) = \mu(b) = 0$.

We strengthen the sampling condition and require $c_{\tilde{\mu}}(pq) = \|pq\| - \tilde{\mu}(p) - \tilde{\mu}(q) \leq 0$ for all edges in the reconstruction.

Since $c_s \leq m$ for all sharp corners, we have $\tilde{\mu}(p) = \bar{\mu}(p)$ for all $p \notin T$. Here $\bar{\mu}$ denotes the original potential function, but with the capped c-values.

Suppose that V satisfies the strengthened sampling condition. Then the minimum spanning tree with respect to $c_{\tilde{\mu}}$ is equal to the polygonal reconstruction. This requires us to check that Lemmas 3.6 to 3.10 stay true; we leave the straightforward but tedious check to the reader. The minimum spanning tree with respect to μ is the same as the minimum spanning tree with respect to $\tilde{\mu}$, since μ and $\tilde{\mu}$ differ only by a constant. We conclude that the minimum spanning tree with respect to μ is equal to the polygonal reconstruction and moreover unique. We finally observe that μ is nonpositive and that $\mu(a) = \mu(b) = 0$. Thus MST_μ is the unique optimal solution of the subtour-LP by Lemma 3.2.

5. Closed curves. We extend the result to closed curves in two steps.

- In section 5.2 we alter the potential function to μ' so that the two longest edges of the polygonal reconstruction have the same modified cost and so that the minimum spanning trees with respect to the new modified cost are precisely the polygonal reconstruction minus one of the edges of maximal modified cost.
- In section 5.1 we show that the preceding sentence implies our main theorem for closed curves.

Observe that, in our write-up, the second step is dealt with first.

Readers familiar with the Held–Karp bound for traveling salesman tours may wonder why we are not arguing about 1-trees. We tried but could not get the argument to work. A 1-tree is defined as follows. An arbitrary node $v \in V$ is fixed. A 1-tree consists of the two cheapest edges incident to v plus a minimum spanning tree of $V \setminus v$. We were unable to construct a potential function for which the optimal 1-tree coincides with the polygonal reconstruction. We were able to construct a potential function where the two cheapest edges incident to v were indeed the edges to the two neighbors in the polygonal reconstruction and were able to construct a potential function where the minimum spanning tree on $V \setminus v$ coincided with the polygonal reconstruction minus the two edges incident to v. We were unable to satisfy both conditions simultaneously.

5.1. The subtour-LP and the global reasoning. Assume that the potential function μ' has been constructed. The subtour-LP for the traveling salesman problem can be formulated as follows:

$$\begin{aligned}
\min \quad & \sum_{u,v \in V} c_{uv} x_{uv} \\
\text{s.t.} \quad & \sum_{v \in V} x_{uv} = 2 && \text{for } u \in V, \\
& \sum_{u \in V', v \in V'} x_{uv} \leq |V'| - 1 && \text{for } V' \subseteq V, \emptyset \neq V' \neq V, \\
& \sum_{u,v \in V} x_{uv} = |V|, \\
& 0 \leq x_{uv} \leq 1.
\end{aligned}$$

The last equality is redundant but helpful for our Lagrangian-relaxation. The length of the polygonal reconstruction is an upper bound for the objective value of the subtour-LP. We relax the set of degree equalities to the objective function and obtain the following problem with the same objective value [7, pp. 258–260]:

$$\begin{aligned}
\max \quad & 2 \sum_{u \in V} \mu(u) + \min \sum_{u,v \in V} c_\mu(uv) x_{uv} \\
\text{s.t.} \quad & \sum_{u \in V', v \in V'} x_{uv} \leq |V'| - 1 && \text{for } V' \subseteq V, \emptyset \neq V' \neq V, \\
& \sum_{u,v \in V} x_{uv} = |V|, \\
& 0 \leq x_{uv} \leq 1.
\end{aligned}$$

In this formulation the maximization is over all choices of μ. For fixed μ the inner minimization is over the choices for the x_{uv}. We will show that for $\mu = \mu'$ the polygonal reconstruction is the unique optimal solution for the minimization problem and hence the objective value of the maximization problem is at least the length of the polygonal reconstruction. It cannot be larger and hence the objective value of the maximization problem is equal to the length of the reconstruction. This proves that the polygonal reconstruction is an optimal solution to the subtour-LP. We still need to argue uniqueness. Assume that there is another optimal solution for the subtour-LP. Since the solution satisfies the degree constraints, it will give the same value to the

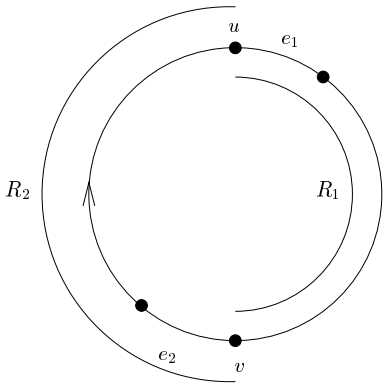


FIG. 5.1. *The notation used in the reformulations of the subtour-LP. We have* $e_i \in E_i$.

inner minimization problem as the polygonal reconstruction and hence the polygonal reconstruction is not the unique optimal solution to the inner minimization problem.

It remains to prove that for $\mu = \mu'$, the polygonal reconstruction is the unique optimal solution for the inner minimization problem. Orient γ arbitrarily, let e_1 and e_2 be edges in the polygonal reconstruction which have maximal modified cost, and let u and v be the starting nodes of e_1 and e_2, respectively, and let R_1 and R_2 be the sample points from u to v (both inclusive), respectively, from v to u (both inclusive) with respect to the order of the points along the curve; see Figure 5.1. Then $R_1 \cup R_2 = V$ and $R_1 \cap R_2 = 2$. Let E_i, $i = 1, 2$, be the set of edges having both endpoints in R_i and let C be the remaining set of edges. Then $e_i \in E_i$ and any edge $e \in C$ has a modified cost larger than e_1 (and hence e_2). Otherwise there would be a minimum spanning tree that is not contained in the polygonal reconstruction. In an optimal solution to the inner LP, the total weight of the edges in E_i is $|R_i| - 1 - o_i$ for some $o_i \geq 0$, $i = 1, 2$, and the total weight of the edges in C is $o_1 + o_2$. Thus the inner LP is relaxed by

$$
\begin{aligned}
\min_{o_1,o_2\geq 0} \min \quad & \sum_{u,v\in V} c_{\mu'}(uv)x_{uv} & & \\
\text{s.t.} \quad & \sum_{u\in R_1', v\in R_1'} x_{uv} \leq |R_1'| - 1 & & \text{for all } R_1' \subseteq R_1, \text{ with } R_1' \neq \emptyset, \\
& \sum_{u,v\in R_1} x_{uv} = |R_1'| - 1 - o_1, & & \\
& \sum_{u\in R_2', v\in R_2'} x_{uv} \leq |R_2'| - 1 & & \text{for all } R_2' \subseteq R_2, \text{ with } R_2' \neq \emptyset, \\
& \sum_{u,v\in R_2} x_{uv} = |R_2| - 1 - o_2, & & \\
& \sum_{uv\in C} x_{uv} = o_1 + o_2, & & \\
& 0 \leq x_{uv} \leq 1. & &
\end{aligned}
$$

Observe that we dropped some of the subtour elimination constraints.

Consider the inner minimization problem for fixed values of o_1 and o_2. The first two lines describe a partial minimum spanning tree for R_1 and the next two lines a

partial minimum spanning tree for R_2. More precisely, the system is minimized if one chooses the $\lfloor\ R_1\ -1-o_1\rfloor$ shortest edges of the minimum spanning tree and fills the fractional part with the next edge.[5] The same is true for R_2. The LP takes its minimum for $o_1 = o_2 = 0$ since any edge in C has higher cost than any edge in the minimum spanning tree. For $o_1 = o_2 = 0$, the system describes the minimum spanning trees for R_1 and R_2. Thus the polygonal reconstruction is the unique optimal solution of the inner minimization problem and hence of the subtour-LP.

5.2. The modified potential function and the local reasoning. We show how to alter the potential so that the two longest edges of the polygonal reconstruction have the same modified cost and so that all minimum spanning trees for V remain part of the polygonal reconstruction. Note that the new potential is defined according to a given sample set, whereas the original potential depends only on the curve.

Let e_{max} be the edge of the polygonal reconstruction with highest modified cost. We claim that one of the following cases arises:

(1) There is a sharp corner so that both endpoints of e_{max} are outside the ball $B(s, \delta_s/5)$.

(2) There is a point $v \in \gamma$ so that $\|vu\|/2 \geq \mu(u)$ for both endpoints u of e_{max} and so that v does not lie in the $B^0(s, \delta_s)$ ball of any sharp corner s.

The first case certainly arises when there are at least three sharp corners. Assume that the first case does not hold. We make a further case distinction: Either both endpoints of e_{max} lie in T (this will certainly be the case when there is no sharp corner) or some endpoint of e_{max} lies in $B(s, \delta_s/5) \setminus T$ for some sharp corner s. In the former case the curve must leave the union of the $B(u, d(u))$ balls of the two endpoints u of e_{max} (because the curve can turn by at most $2\pi/3$ within the union of these balls) and any curve point v outside the two balls (since $\|uv\| \geq d(u) = 3\mu(u)$ for any of the endpoints) and outside $\cup_{s\in S} B^0(s, \delta_s)$ will work; v exists since the regions $(B^0(s, \delta_s))_{s\in S}$ are pairwise disjoint and since γ turns by less than π in any such region. So assume that some endpoint of e_{max} lies in $B(s, \delta_s/5) \setminus T$ of some sharp corner s.

[5] For $o_1 = 0$ this is the characterization of minimum spanning trees by a linear program; see [7, Theorem 2.8]. The proof extends to nonzero values of o_1. We sketch the proof for the sake of completeness. The dual linear program has an unconstrained variable y_{R_1} and a nonpositive variable $y_{R'}$ for every proper subset of R_1. It reads $\max(|\,R_1\,|-1)y_{R_1} + \sum_{R';\emptyset\neq R'\neq R_1}(|\,R'\,|-1)y_{R'}$ subject to $y_{R_1} + \sum_{R';e\in\gamma(R')} y_{R'} \leq c_e$ for every edge e and $y_{R'} \leq 0$ for every R', $\emptyset \neq R' \neq R_1$. In this footnote we use $\gamma(R)$ to denote all edges having both endpoints in R. We use Kruskal's algorithm to construct a primal and dual solution simultaneously. The dual solution will always be feasible; the primal solution will always satisfy $x(\gamma(R')) \leq |\,R'\,| - 1$ for all nonempty R' (but will satisfy $x(\gamma(R_1)) = n - 1 - o_1$ only in the last step); and we will always have complementary slackness.

We start with the empty spanning forest, i.e., $x_e = 0$ for all e, and all dual variables equal to zero. We declare all singleton sets R' active. We then increase y_{R_1} and decrease $y_{R'}$ for all active R' at the same rate until the first constraint $y_{R_1} + \sum_{e;e\in\gamma(R)} y_R \leq c_e$ becomes tight. This will be for the minimum cost edge (let us assume for simplicity that all costs are distinct); observe that the $y_{R'}$ for singletons sets R' are irrelevant. We add e to the spanning forest, i.e., we change x_e to one. Observe that this preserves complementary slackness. The addition of e to the spanning forest combines two active sets R'_1 and R'_2 into a set R'. We declare R'_1 and R'_2 inactive and R' active. Observe that R' corresponds to a subtree of the spanning tree and hence the primal constraint $x(\gamma(R')) = |\,R'\,| - 1$ is tight. Thus $y_{R'}$ is a dual variable which complementary slackness allows to become nonzero. We proceed, i.e., we increase y_{R_1} and decrease $y_{R'}$ for every active set R' at the same rate. Observe what happens. For every edge f contained in an active set R' the left-hand side of $y_{R_1} + \sum_{e;e\in\gamma(R)} y_R \leq c_f$ will not change, because the increase of y_{R_1} is balanced by the decrease of $y_{R'}$. For any edge f connecting two active sets the left-hand side will increase. When the next edge becomes tight, we add it to the spanning tree, ... We finish when adding a partial edge makes the constraint $x(\gamma(R_1)) = n - 1 - o_1$ tight.

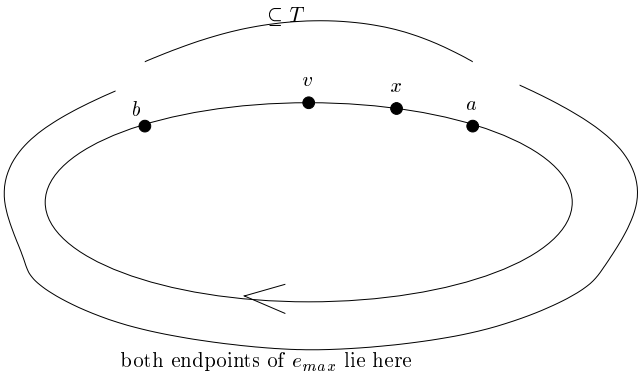


FIG. 5.2. *The construction of the potential function μ'.*

Consider the leg of s that does not contain the other endpoint of e_{max}. It contains no endpoint of e_{max} that lies in T. Let v be the point on this leg of s with distance δ_s from s and let u be an endpoint of e_{max}. If $u \in B(s, \delta_s/5)$, then $\|uv\| \geq 4\delta_s/5$ and $\mu(u) \leq \max(c_s, d(u)/3) \leq 2\delta_s/5$ and if $u \notin B(s, \delta_s/5)$, then $u \in T$ and hence u does not lie on the same leg as v does. Then $d(u) \leq \|uv\|$ by Lemma 3.3 and we are done. In either case we have shown that one of items above holds. We also need the following lemma.

LEMMA 5.1. *Let s be a sharp corner and let x and y be adjacent sample points so that $s \in \gamma[x, y]$. Then $x, y \in B(s, 3\delta_s/5)$ and either x or y lies in $B(s, \delta_s/5) \setminus T$.*

Proof. Since the regions $B^0(p, d(p)) \cap T$ do not contain any sharp corner and since the regions $(B^0(t, \delta_t))_{t \in S}$ are pairwise disjoint, we have $x, y \in B^0(s, \delta_s)$ by Lemmas 3.5 and 3.6. Assume $x \notin B(s, 3\delta_s/5)$. Then

$$c_\mu(x, y) \geq c_\mu(s, x) \geq \|sx\| - 2\delta_s/5 - \|sx\|/3 \geq 2\|sx\|/3 - 2\delta_s/5 > 2\delta_s/5 - 2\delta_s/5 = 0,$$

where the first inequality follows from the third claim in the proof of Lemma 3.10. Assume next that $x, y \in T$. Lemma 3.3 implies $d(x), d(y) \leq \|xy\|$ and hence $c_\mu(x, y) \geq \|xy\|/3 > 0$. □

We now turn to the definition of the modified potential function and the proof that all minimum spanning trees for V with respect to the modified potential function are subsets of the polygonal reconstruction.

Assume first that there is a sharp corner s so that both endpoints of e_{max} lie outside the $B(s, \delta_s/5)$ ball. We decrease c_s continuously down to zero. For $c_s = 0$, we have $\mu(x) = d(p)/3$ for all $x \in B(s, \delta_s)$ and hence the edge xy in the reconstruction that connects the two legs of s (it exists by Lemma 5.1) has positive modified cost: $c_{\mu'}(x, y) = \|xy\| - \mu'(x) - \mu'(y) \geq \|xy\| - \|xy\|/3 - \|xy\|/3 > 0$ since $d(x), d(y) \leq \|xy\|$ and hence $c_\mu(x, y) \geq \|xy\|/3 > 0$ by Lemma 3.3. The modified cost of the edge e_{max} is not affected by the change of c_s. Thus there must be a value of c_s for which the two largest modified costs in the reconstruction are the same. All edges in the reconstruction still have nonpositive modified cost (because e_{max} has) and the minimum spanning tree with respect to $c_{\mu'}$ remains the polygonal reconstruction by the remark in section 3.4 that only the fact that $0 < c_s \leq 2\delta_s/5$ is used in our proofs. This completes the discussion of the first case.

We come to the case that there is a point $v \in \gamma$ so that $\mu(u) \leq \|vu\|/2$ for both endpoints u of e_{max} and so that v lies outside the $B^0(s, \delta_s)$ balls of all sharp corners.

We split the curve at v and orient the resulting open curve arbitrarily. For any $l \in \mathbb{R}$, $0 \leq l \leq \mu(v)$, we define a as the first point with $\mu(a) = \|va\|/2 + l$ and b as the last point with $\mu(b) = \|vb\|/2 + l$ and define a potential μ' by

$$\mu'(p) = \begin{cases} \mu(p) & \text{if } a \leq p \leq b, \\ \|vp\|/2 + l & \text{otherwise.} \end{cases}$$

Observe that a, b, and μ' depend on l. For simplicity of notation we do not make this dependence explicit in the notation. Figure 5.2 illustrates the definition of μ'.

We need to argue that a and b exist for all choices of l, that the cost of e_{max} does not depend on l, that there is a choice of l for which the two largest modified costs are the same, and that every minimum spanning tree with respect to $c_{\mu'}$ uses only edges of the polygonal reconstruction.

We first show the existence of a and b for all choices of l. For $l = \mu(v)$ we have $a = b = v$. For $0 \leq l < \mu(v)$ we have $\mu(v) > l + \|vv\|/2$ and $\mu(u) \leq \|uv\|/2 \leq l + \|uv\|/2$ for either endpoint of e_{max}. From the continuity of μ we conclude that there is a point a between v and u for which $\mu(a) = \|va\|/2 + l$ and a point b between u and v for which $\mu(b) = \|vb\|/2 + l$. We have shown that a and b exist and that e_{max} does not depend on l. Furthermore we know that $\mu'(p) \leq \mu(p)$ for all p by the definition of a and b.

We next show that $a \leq r$, where r is the first point on γ with $\|sr\| = 3\delta_s/5$ for some sharp corner s, and hence $a \in T$. We have $\|rv\| \geq 2\delta_s/5$ and $\mu(r) = d(r)/3 \leq \|sr\|/3 = \delta_s/5 \leq \|rv\|/2 \leq \|rv\|/2 + l$. Continuity of μ implies that a lies between v and r. Similarly, b lies in T and after the last point r on γ with $\|sr\| = 3\delta_s/5$ for some sharp corner.

We next argue that there is a choice of l for which the two largest modified costs in the reconstruction are the same. From $a \leq u \leq b$ for any endpoint u of e_{max} we conclude that the modified cost of e_{max} does not depend on l. For $l = \mu(v)$, we have $a = b = v$ and hence $\mu = \mu'$. Thus e_{max} has the maximal modified cost $c_{\mu'}$ among all edges in the reconstruction. For $l = 0$, we consider the reconstruction edge between the last and the first sample points x and y, respectively, and show that it has positive modified cost $c_{\mu'}(xy)$. There must be a subcurve γ' containing $\gamma[x, y]$ and hence v. Since $v \notin B(s, \delta_s)$ for any sharp corner, γ' is a regular subcurve and hence $\gamma[x, y] \subseteq T$. We first show that $a < x$ and $y < b$ is impossible. Assume otherwise. Then $\mu(x) \leq \mu(a) + \|ax\|/3 = \|va\|/2 + \|ax\|/3$ and $\mu(y) \leq \mu(b) + \|by\|/3 = \|vb\|/2 + \|by\|/3$ and $\|xy\| > \|xa\|/2 + \|ab\| + \|by\|/2$ since γ' turns by less than $\pi/3$. Thus $c_\mu(x, y) > 0$, a contradiction to our sampling condition. Thus either $v \leq x \leq a$ or $b \leq y \leq v$ or both. We may assume w.l.o.g. that $v \leq x \leq a$. If $y < b$, we have $\mu(y) \leq \mu(b) + \|by\|/3 = \|vb\|/2 + \|by\|/3$, $\|xy\| > \|xv\|/2 + \|vb\| + \|by\|/2$, and $\mu'(x) = \|vx\|/2$ and hence $c_{\mu'}(x, y) > 0$. If $b \leq y$, we have $\|xy\| > \|xv\|/2 + \|vy\|/2$, $\mu'(y) = \|vy\|/2$ and $\mu'(x) = \|vx\|/2$ and hence $c_{\mu'}(x, y) > 0$. In either case we have shown that for $l = 0$ there is an edge in the reconstruction with positive modified cost. Continuity implies that there is a value of l for which the two largest modified costs in the reconstruction are the same. This completes the definition of the modified potential function in the second case.

In order to verify that all minimum spanning trees are subsets of the polygonal reconstruction is suffices to show that Lemma 3.6 holds for the new potential function and that $c_{\mu'}(p, r) > c_{\mu'}(p, q)$ for any three points with $p < q < r$ and $\{p, q, r\} \subseteq \gamma'$ for some $\gamma' \in \Gamma$.

From $\mu'(x) \leq \mu(x)$ for all x, we conclude $c_{\mu'}(pq) \leq c_\mu(pq)$ for all edges pq. Thus

Lemma 3.6 stays true.

If $\{p,q,r\} \subseteq B(u,d(u)) \cap T$ for some $r \in \gamma$, we have $\angle(\vec{pq},\vec{qr}) < \pi/3$ and hence $\|pr\| - \|pq\| > \|qr\|/2$ and $\mu'(r) \leq \mu'(q) + \|qr\|/2$. Thus $c_{\mu'}(p,r) > c_{\mu'}(p,q)$.

Assume next that $\{p,q,r\} \subseteq B(s,\delta_s)$ for some sharp corner s. If $c_{\mu'}(p,r) \geq 0$ we are done. Otherwise $c_\mu(p,r) < 0$, since the modified distance with respect to μ is at most the modified distance with respect to μ'. We make the same case distinction as in the proof of Lemma 3.9. In this proof we bounded $\|pr\| - \|pq\|$ from below and $\mu(r) - \mu(q)$ from above. We now need to bound $\mu'(r) - \mu'(q)$. Since $\mu'(x) = \mu(x)$ for $x \notin T$, we need only to reconsider the case that r and q are in T. We have $\mu'(r) \leq \mu'(q) + \|qr\|/2$ and hence the arguments used in the cases $p < q < r \leq s$, $s \leq p < q < r$, and $p < q < s < r$ stay valid. We need only to reconsider the case $p < s \leq q < r$.

If $\mu(p) \neq \mu'(p)$, we have $\mu(q) = \mu'(q)$ and $\mu(r) = \mu'(r)$ and are done. So assume $\mu(p) = \mu'(p)$. If $\mu(q) = \mu'(q)$ we have $c_{\mu'}(p,r) - c_{\mu'}(p,q) \geq c_\mu(p,r) - c_\mu(p,q) \geq 0$, since $\mu'(r) \leq \mu(r)$. Otherwise $\mu'(r) - \mu'(q) \leq 0 \leq \mu(r) - \mu(q)$, since r is closer to v than q and both q and r are in $T \cap B(s,\delta_s)$. Thus $c_{\mu'}(p,r) - c_{\mu'}(p,q) \geq c_\mu(p,r) - c_\mu(p,q) \geq 0$.

6. Solving the subtour-LP. The subtour-LP has an exponential number of constraints. The ellipsoid method [18] allows one to solve LPs with an exponential number of constraints in time polynomial in the number of variables if the following conditions are satisfied:

- The coefficients of the variables in the constraints are polynomially bounded. This is the case for the subtour-LP.
- The separation problem can be solved in polynomial time, i.e., given a vector x^*_{uv} of polynomially bounded values for the variables, one can decide in polynomial time whether the vector satisfies all constraints and, if not, exhibit a violated constraint. This is the case. It is trivial to check the degree constraints and the constraint that all values lie between zero and one. In order to check the subtour elimination constraints, we discuss the case of tours, having already discussed the case of path in sections 3 and 4. Consider the complete network on V and assign capacity x^*_{uv} to edge uv. Consider any subset V' of V with $\emptyset \neq V' \neq V$ and observe that $2|\,V'\,| = \sum_{u\in V'}\sum_{v\in V} x^*_{uv} = 2\sum_{u\in V', v\in V'} x^*_{uv} + \sum_{u\in V'}\sum_{v\notin V'} x^*_{uv}$. We conclude that the subtour elimination constraint for V' is satisfied iff the capacity of the cut $(V', V \setminus V')$ is at least two. Some subtour elimination constraint is satisfied if the minimum cut is less than two. A minimum cut can be computed in polynomial time.
- The coefficients in the objective function are polynomially bounded. This is not the case since the bit-representation of an Euclidean length is in principal infinite. Furthermore there are curves where the bit-representation of any point is infinite.

We show that it suffices to know the sample points and their Euclidean distances only approximately. More precisely, we show the following: Let S be a set of points. Let m be the minimal distance between any two points in S and for a point $p \in S$. Let p^* be a closest point of γ. If

1. $\|pp^*\| \leq \rho m/10$ for all $p \in S$, where ρ is a constant depending on γ, and
2. the set $S^* = \{p^* \mid p \in S\}$ satisfies a strengthened sampling condition,

then the subtour-LP has a unique optimal integral solution even when the distances between sample points are only known up to an error of $m\rho/10$. Moreover, the subtour-LP can be solved in polynomial time and the optimal solution is a tour connecting the points in S in the order in which the points S^* lie on γ. We also show

how to estimate ρ from the sample set and without knowledge of γ.

Now we make precisely what we mean by an approximate sample set of a curve γ. Recall that for an angle α we have defined $\bar{\alpha}$ by $\bar{\alpha} = \pi - \alpha$. Let $\alpha_0 = \max_{s \in S} \alpha_s$. We define $\bar{\alpha}_0 = 17\pi/24$, if there is no sharp corner.

DEFINITION 6.1. *Let γ be a benign semiregular curve. We call a set S of points $p \in \mathbb{Q}^2$ an* approximate sample set *of a curve γ, if for all $p \in S$*

$$\|p\gamma\| \leq (\sin \bar{\alpha}_0/4)^2 m/30.$$

For the following let $\rho = (\sin \bar{\alpha}_0/4)^2 m/3$. Then $m > 7\rho$ (since $\bar{\alpha}_0 < 17\pi/24$) and $\|pp^*\| \leq \rho/10$ for the points in an approximate sample. We need the following sampling condition.

Sampling condition for approximate sample sets.

(a) For any two adjacent (on γ) samples u^* and v^*: $\|u^*v^*\| \leq 9/10(\mu(u^*)+\mu(v^*))$.

(b) For any two adjacent samples u^* and v^*: $\gamma[u^*, v^*]$ turns by less than π. For two adjacent points p, q on the curve, $\gamma[p, q]$ denotes the subcurve of γ with endpoints p and q not containing another sample point.

Let γ be a benign semiregular curve and S an approximate sample set satisfying the sampling condition. For any two sample points p and q let $\|pq\|_{\approx}$ be a rational number, so that $\|pq\| - \|pq\|_{\approx} \leq \rho/10$. Then $\|p^*q^*\| - \|pq\|_{\approx} \leq 3\rho/10$. Note that for all $p, q \in S$ there exists a choice of $\|pq\|_{\approx}$ which has a bit representation of polynomial size in m and $\bar{\alpha}_0$. We consider the approximate subtour-LP of the approximate sample set S.

THEOREM 6.2. *Let γ be an open (closed) benign semiregular curve and S be an approximate sample set of γ satisfying the sampling condition above.*

- *The approximate subtour-LP for S has an unique optimal integral solution.*
- *The approximate subtour-LP can be solved in polynomial time.*

Proof. We define the potential function of an approximate sample point p as $\mu(p) := \mu(p^*)$. We call the new modified cost function $c_\mu^{\approx}$. We show that

1. if p and q are adjacent sample points, then $c_\mu^{\approx}(p, q) < -3\rho/10$ and hence there is a minimum spanning tree in which each edge has length less than $-3\rho/10$;
2. if p and q are sample points which are not contained in some $\gamma' \in \Gamma$, then $c_\mu^{\approx}(p, q) \geq -3\rho/10$ and hence no such edge belongs to a minimum spanning tree;
3. if $p < q < r$ are three sample points contained in some $\gamma' \in \Gamma$ and $c_\mu^{\approx}(p, r) < -3\rho/10$, then $c_\mu^{\approx}(p, r) > c_\mu^{\approx}(pq)$, and hence the minimum spanning tree reconstructs locally.

We turn to the first item. Let p and q be adjacent sample points. We have $6\rho < m - \rho \leq \|pq\| - \rho \leq \|p^*q^*\| + 2\rho/10 - \rho \leq \|p^*q^*\| \leq \mu(p^*) + \mu(q^*)$. Thus

$$\begin{aligned} c_\mu^{\approx}(p, q) &= \|pq\|_{\approx} - \mu(p) - \mu(q) \\ &\leq \|p^*q^*\| + 3\rho/10 - 1/10(\mu(p^*) + \mu(q^*)) - 9/10(\mu(p^*) + \mu(q^*)) \\ &< \|p^*q^*\| - 3\rho/10 - 9/10(\mu(p^*) + \mu(q^*)) \\ &\leq -3\rho/10. \end{aligned}$$

For the second item, consider points p and q that are not contained in any common subcurve $\gamma' \in \Gamma$. We have

$$\begin{aligned} c_\mu^{\approx}(p, q) &\geq \|p^*q^*\| - 3\rho/10 - \mu(p^*) - \mu(q^*) \\ &= c_\mu(p^*, q^*) - 3\rho/10 \\ &\geq -3\rho/10. \end{aligned}$$

We come to the third item. Consider three sample points $p < q < r$ that are contained in some $\gamma' \in \Gamma$ and for which $c^{\approx}_{\mu}(pr) < -3\rho/10$. Then $c_\mu(p^*, r^*) < 0$ and hence (using Lemmas 3.7 and 3.9 and the fact that $(\sin\bar{\alpha_0}/4)^2/3 \leq 1/6$)

$$\begin{aligned} c^{\approx}_{\mu}(p,r) - c^{\approx}_{\mu}(p,q) &\geq c_\mu(p^*, r^*) - c_\mu(p^*, q^*) - 6\rho/10 \\ &\geq \|qr\|(\sin\bar{\alpha}_0/4)^2/3 - 6\rho/10 \\ &\geq \rho - 6\rho/10 > 0. \qquad \square \end{aligned}$$

What have we achieved at this point? We have shown that the subtour-LP reconstructs provided our sample set S satisfies the sampling condition for approximate sample sets *and* we are given approximate distances of polynomial size that differ by at most $\rho/10$ from the true distances. We could compute approximate distances with the required property, if we were given ρ or alternatively α_0 as an additional input. We now show how to compute a lower bound on α_0, which leads to a polynomial precision in the input size, without any additional knowledge of the curve.

LEMMA 6.3. *Let m and M be the minimal, respectively maximal, distance between two sample points. Then $sin(\bar{\alpha}_0/4) \geq m/(15M)$.*

Proof. If γ has no sharp corners, $\alpha_0 = 17\pi/24$ and there is nothing to show. So assume otherwise and let s be any sharp corner. We prove that there is a sample point p in $B(s, \delta_s) \cap T$ on each leg of the sharp corner and then use this fact to bound $\bar{\alpha}_s$ from below.

We look at an arbitrary order of the two orders obtained by splitting the curve at s and prove that there is a sample point behind s in $B(s, \delta_s) \cap T$. Assume otherwise. Let x be the first sample point behind s outside $B(s, \delta_s)$ and let y be the sample point preceding x. Since every edge of the polygonal reconstruction must lie in at least one subcurve $\gamma' \in \Gamma$, y must lie behind s. By assumption y does not lie in T. Assume first that $x \in T$. Then $\|xy\| \geq 4\delta_s/5$ and $\mu(y) \leq 2\delta_s/5$. Thus $c_\mu(x, y) = \|xy\| - \mu(x) - \mu(y) \geq \|xy\| - 2\delta_s/5 - \|xy\|/3 \geq 2\|xy\|/3 - 2\delta_s/5 \leq 8\delta_s/15 - 2\delta_s/5 > 0$, a contradiction. Assume now $x \notin T$. Let s' be the corner so that $x \in B(s', \delta_{s'})$ and assume w.l.o.g. $\delta_s \geq \delta_{s'}$. Then $\mu(x) \leq 2\delta_s/5$, $\mu(y) \leq 2\delta_s/5$ and $\|xy\| \geq 2\delta_s - \delta_s/5 - \delta_{s'}/5 \geq \delta_s$. Thus $c_\mu(x, y) > 0$, a contradiction.

Let p be the first sample point behind s in $B(s, \delta_s) \cap T$ and let q be the adjacent sample point behind p. Then $q \in T$; if $q \notin T$, then $q \in B(s', \delta_{s'})$ for some sharp corner s' and hence s' would have no sample point in $B(s', \delta_{s'}) \cap T$. The distance between p and q is at least m. Also $\|pq\| \leq d(p)/3 + d(q)/3 \leq d(p)/3 + (d(p) + \|pq\|)/3$ and so $d(p) \geq \|pq\| \geq m$. Consider the intersections of the two legs of s with the boundary of the δ_s-ball centered at s. The intersections have distance at least m (since d-values grow along each leg) and s sees the intersections under an angle of at least $\bar{\alpha}_0/2$. Thus $\sin\bar{\alpha}_0/4 \geq m/(2\delta_s)$. Since there is at least one sample outside the ball $B(s, \delta_s)$ and at least one sample inside the ball $B(s, \delta_s/5)$, we have $M \geq 4\delta_s/5$. Thus $\sin\bar{\alpha}_0/4 \geq 4m/(10M)$. $\square$

7. Collections of closed curves. In the preceding sections we showed that the subtour-LP formulation of the traveling salesman problem is able to reconstruct *single* closed and open curves. In this section we extend the algorithm so that is can handle collections of closed curves. We do not know how to handle collections of open and closed curves. Please note that the algorithms [9, 13] can handle open and closed curves.

The algorithm works in rounds. The first round constructs an initial partition of the sample points and subsequent rounds merge blocks of the partition. The construction of the initial partition and the merging is done conservatively, i.e., all points

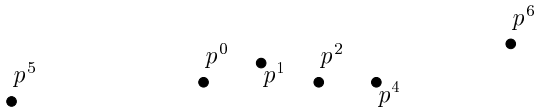


FIG. 7.1. *The points p^1 up to p^5 are joined with p^0, but p^6 is not joined.*

in the same block provably belong to the same curve. In the first round, every point is joined to points close to it; section 7.1 gives the details. In later rounds (see section 7.2), we solve the subtour-LP for each block and then analyze the solution. If the subtour-LP fails on a block or if curves constructed for different blocks interfere, some blocks are merged.

Throughout this section we assume our set of sample points to satisfy a somewhat strengthened sampling condition. The strengthened sampling condition leads to denser sampling near sharp corners. We change μ to μ' by decreasing the δ- and c-values of sharp corners. We set $\delta'_s \le \delta_s$ such that for any two points p and q in $B(s, \delta'_s/7)$ the angle between the segment pq and the corresponding tangent in s is at most $\pi/40$. Furthermore we decrease the value c_s to c'_s such that $c'_s - \|sp_s\| \le d(p)/3$ for every $p \notin B(s, \delta'_s/60)$. This enlarges the region where the potential is defined by $d(p)/3$ from T to T'. Recall that the choice of c_s guaranteed only that the points outside $B(s, \delta_s/5)$ belonged to T, whereas T' contains all points outside the balls $B(s, \delta'_s/60)$.

7.1. The initial partition. We define a graph on our set of sample points. The connected components of this graph form the initial partition. For a sample point $p = p^0$ let p^1, p^2, ... be the other sample points in order of increasing distance (ties are broken arbitrarily). We always join p^0 with p^1 and p^2. We join p^0 and p^i, $i \ge 3$, if $\angle\overrightarrow{p^{k-1}p^{k-2}}, \overrightarrow{p^{k-1}p^k} \ge 2\pi/3$ for all k with $2 \le k \le i-1$. Observe that the decision whether p^i is joined to p^0 depends only on the points p^0 up to p^{i-1}, but not on the point p^i itself. This is essential for making connections between the points on different legs of a sharp corner but also hinders the extension to open curves. Figure 7.1 illustrates the definition.

LEMMA 7.1. *If p and its two adjacent sample points are in T', then p is only joined with points in $B(p, d(p))$ and is joined with both adjacent sample points.*

Proof. Since p is in T', $\gamma \cap B(p, d(p))$ consists of a single component and turns by less than $\pi/3$. Also the two sample points q, r adjacent to p on γ lie in $B(p, d(p))$. We show this for q. We have $d(q) \le d(p) + \|pq\|$, $\mu(q) = d(q)/3$ since $q \in T'$, and, by our sample condition, $\|pq\| \le d(p)/3 + d(q)/3$. Thus $\|pq\| \le d(p)/3 + (d(p) + \|pq\|)/3$ and hence $\|pq\| \le d(p)$.

Assume w.l.o.g. that q is considered before $r = p^i$. Orient $\gamma \cap B(p, d(p))$ such that $r < p < q$. Then $q = p^1$ and $p^1 < p^2 < \cdots < p^{i-1}$. Since $\gamma \cap B(p, d(p))$ turns by less than $\pi/3$, we have $\angle\overrightarrow{p^{k-1}p^{k-2}}, \overrightarrow{p^{k-1}p^k} \ge 2\pi/3$ for all k with $2 \le k \le i-1$ and $\angle\overrightarrow{p^{i-1}p^{i-2}}, \overrightarrow{p^{i-1}p^i} \le \pi/3$. Thus p is joined with p^1 up to p^i, but not with p^{i+1}. ∎

We turn to the nonsmooth parts of the curve. We first show that our sampling condition implies that each leg of a sharp corner must contain several sample points.

LEMMA 7.2. (a) *Let p be a sample point in $B(s, \delta'_s) \setminus B(s, 2\delta'_s/60)$. Then both adjacent sample points lie on the same leg as p, the one closer to s has distance at most $\|ps\|/2$ from p, and the one further from s has distance at most $\|ps\|$ from p.*

(b) *For every leg ℓ of a sharp corner s we have at least one sample point in $B(s, 2 \cdot 2^j \delta'_s/60) \setminus B(s, 2^j \delta'_s/60)$ for $j = 1, 2, 3, 4$.*

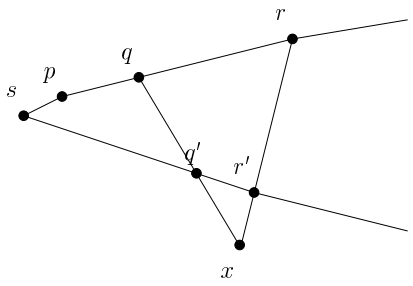


FIG. 7.2. *The angle between $q'q$ and $r'r$ must be large that the region is grown further. This contradicts the fact that the distance between q and r is short.*

Proof. (a) We have $\mu'(p) \leq \|sp\|/3$. The point q on the same leg as p with distance exactly $\|sp\|/2$ to s lies in T' and hence $\mu'(q) \leq \|sp\|/6$. Thus $c_{\mu'}(p,q) \geq \|sp\|/2 - \|sp\|/6 - \|sp\|/3 = 0$. Lemma 3.9 implies that $c_{\mu'}(p,x) \geq 0$ for any point x between s and q and for any point x on the other leg. Thus there must be a sample point between p and q. We conclude that both adjacent sample points lie on the same leg as p and that the one closer to s satisfies the distance constraint stated. For the one further from p we consider the point q on the same leg as p, further away from s than p, and having distance $\|sp\|$ from p. Then $\mu'(q) \leq 2\|sp\|/3$ and hence $c_{\mu'}(p,q) \geq \|sp\| - 2\|sp\|/3 - \|sp\|/3 = 0$. We now argue as above.

We turn to part (b). Part (a) implies that if one of the annuli contains a point, the adjacent annuli do also and hence all annuli do. We conclude that either all annuli contain a sample point or none does. Assume the latter and let p be the first sample point on ℓ outside $B(s,\delta'_s)$. The sample point q preceding p must lie in $B(s,2\delta'_s/60)$. Thus $\|pq\| \geq \|ps\| - 2\delta'_s/60$, $\mu'(p) \leq \|ps\|/3$, $\|ps\| \geq \delta'_s$, and $\mu'(q) \leq \max(2\delta'_s/60, c_s) \leq 2\delta'_s/5$, a contradiction. □

LEMMA 7.3. *A sample point p is joined only with points of the same curve.*

Proof. Consider a sample point p. If p and both adjacent sample points of p are in T', the claim follows from Lemma 7.1.

Otherwise $p \in B(s,2\delta'_s/60)$ by Lemma 7.2, part (a). Let $p = p^0, p^1, \ldots, p^j$ be the sample points in $B(s,\delta'_s)$ ordered according to their distance from p. Since both legs of the sharp corner contain sample points in the annuli $B(s,2\cdot 2^j\delta'_s/60) \setminus B(s,2^j\delta'_s/60)$ for $j = 2,3,4$, we must have a subsequence of length at least three such that the first and the last element of the subsequence, call them q and r, respectively, lie on the same leg as p and further away from s than p, and the points in between (there is at least one) lie on the other leg. Figure 7.2 illustrates the situation. We show that no point after r is joined to p. Assume otherwise. Let q' be the point added directly after q and r' be the point added directly before r.

We want to bound the angle between the segments $q'q$ and $r'r$. If q' is equal to r' this angle is at least $2\pi/3$. Otherwise the angle between $q'q$ and the segment between q' an the sample q'' added after q' is at least $2\pi/3$ and the angle between $r'r$ and the segment between r' an the sample r'' added before r' is at least $2\pi/3$ (since r is not the last point joined to p). Also q', q'', r'', and r' all lie on the same leg and hence the angle between $q'q$ and $r'r$ is at least $\pi/3 - 4\pi/40$. Here we use the strengthened sampling condition.

Let x be the intersection between the lines supporting qq' and rr'. We have $d(q) < d(r) \leq d(q) + \|qr\|$, $\|rr'\| \geq d(r)$, and $\|qq'\| \geq d(q)$ by Lemma 3.3, $\|qr\| \leq$

$(d(q) + d(r))/3$ by our sampling condition, and hence $\|rx\| \geq \|rr'\| \geq d(r) \geq (d(r) + d(q))/2 \geq 3\|qr\|/2$ and $\|qx\| \geq \|qq'\| \geq d(q) \geq (d(q) + d(r) - \|pq\|)/2 \geq \|qr\|$. Application of the "theorem of cosines[6]" with $D = \|qr\|$ yields $\cos(\angle(q'q, r'r)) \geq [(3D/2)^2 + D^2 - D^2]/(2 \cdot D \cdot 3D/2) = 3/4$ and hence $\angle(q'q, r'r) < \pi/3 - \pi/10$, a contradiction. □

LEMMA 7.4. *For any curve γ_j all sample points in $T' \cap \gamma_j$ belong to the same component.*

Proof. Breaking γ_j at its sharp corners gives us a collection of subcurves γ_{jk}. Consider any subcurve γ_{jk}. The sample points on γ_{jk} come in three groups: first a group of points outside T', then a group of points in T', and finally a group of points outside T'. Lemma 7.1 implies that all sample points in $\gamma_{jk} \cap T'$ belong to a single component. Consider now two adjacent subcurves incident to a sharp corner s. In both subcurves all points in T' belong to the same component. Let p and q be the points in the components containing the points T' which are closest to s and do not connect to both neighbors. We claim that p and q are connected to points on the other leg. Assume otherwise; say p is not connected to a point on the other leg. Since p is not connected to both neighbors we have $p \in B(s, \delta'_s/60)$. Let $p = p^0, p^1, \ldots, p^i$ be the sample points connected to p and ordered according to their distance from p. Then $s < p < p^1 < \cdots < p^i$ since p is not connected to both neighbors and since p is not connected to a sample point on the other side. Also we have shown in the proof of Lemma 7.3 that $p^i \in B(s, \delta'_s)$. Thus $\angle(p^{i-1}p^{i-2}, p^{i-1}p^i) \geq 2\pi/3$, a contradiction to the fact that p^i is the last point joined with p.

Thus p connects to a point u on the other leg and q connects to a point v on the other leg. If either u or v belong to the component containing the points in T' we are done. So assume otherwise. Then u is closer to s than q, and v is closer to s than p, and hence $pvuq$ builds a convex quadrangle. The two segments pu and qv are crossing. Thus either $\|pv\| < \|pu\|$ or $\|qu\| < \|qv\|$ since $\|pv\| + \|qu\| < \|pu\| + \|qv\|$ and hence either p or q will be joined with a sample closer to s, a contradiction. □

We call the component containing all sample points in $\gamma_j \cap T'$ the main component of γ_j.

LEMMA 7.5. *The subtour-LP applied to the main component of γ_j reconstructs γ_j.*

Proof. We show that the sample points in $\gamma_j \cap T'$ satisfy our original sampling condition. Consider two points p and q in $\gamma_j \cap T'$ that are adjacent along γ_j. If they are also adjacent in the full sample, we are done. Assume otherwise. Then $\{p, q\} \subseteq B(s, 2\delta'_s/60)$ for some sharp corner s. Let t be the point such $c_s = d(t)/3 + \|st_s\|$; see Figure 7.3. We have $\|pq\| \leq 2 \cdot (2\delta'_s/60) \cdot \sin 3\overline{\alpha}_s/4 + \|sp_s\| - \|sq_s\|$, since $\angle(sp, sp_s) \leq 3\overline{\alpha}_s/4$ and similarly for q, $d(t) \geq \delta'_s/5 \sin \overline{\alpha}_s/2$, since the ball around t with radius $\delta_s/5 \sin \overline{\alpha}_s/2$ does not intersect the other leg, $\mu(p) \geq c_s - \|sp_s\| = d(t)/3 + \|st_s\| - \|sp_s\| = d(t)/3 + \|t_sp_s\| \geq \frac{\delta'_s}{15} \sin \overline{\alpha}_s/2 + \|t_sp_s\|$ and, by the same argument, $\mu(q) \geq \frac{\delta'_s}{15} \sin \overline{\alpha}_s/2 + \|t_sq_s\|$. Thus $c_\mu(p, q) \leq \frac{4\delta'_s}{60} \sin 3\overline{\alpha}_s/4 + \|sp_s\| - \|sq_s\| - 2\frac{\delta'_s}{15}\frac{2}{3} \sin 3\overline{\alpha}_s/4 - \|t_sp_s\| - \|t_sq_s\| \leq 0$. □

7.2. Merging components. The initial partition may contain too many components. For every curve γ_j, there is a main component which contains all sample points in $\gamma_j \cap T'$ and maybe other components. We call them minor components. Each minor component is contained in $B(s, 2\delta'_s/60)$ for some sharp corner s. The

[6] $\|qr\|^2 = \|qx\|^2 + \|rx\|^2 - 2\|qx\| \cdot \|rx\| \cdot \cos(\angle(q'q, r'r))$.

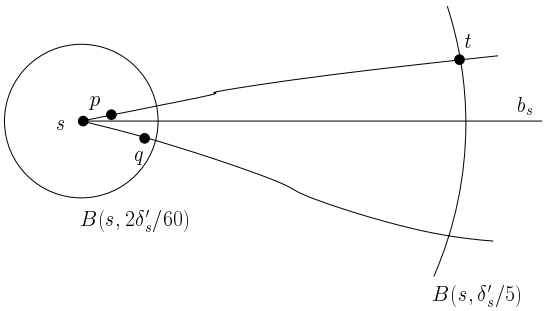


FIG. 7.3. *The definition of t.*

reconstruction based on the subtour-LP is guaranteed to succeed for the main component of every curve. For the minor components it may or may not produce a tour. In this section we describe a strategy for merging components.

We define a region R_{pq} for every edge pq of the computed reconstruction. If the region R_{pq} for an edge pq of the computed reconstruction contains a sample point from another component, we join the component containing p and q with the component of the point closest to pq and lying in another component. We continue until the components stabilize.

Before we define the regions R_{pq} we draw an important consequence of the merging rule. For an edge in a minor component the point closest to it and in another component is guaranteed to lie on the same curve. This follows from the fact that a minor component is contained in $B(s, 2\delta'_s/60)$ for some sharp corner, that the corresponding major component has a point in $B(s, 4\delta'_s/60)$, and that any point within $B(s, \delta'_s)$ belongs to the same curve.

In the arguments to follow we can therefore concentrate on edges in the reconstruction of the main component. In particular, we can use the fact that the subtour-LP correctly reconstructs the main component. We come to the definition of the regions R_{pq}. We define R_{pq} as the union of a region R'_{pq} and the circumcircle of the segment pq. For every sample point p we define β_p to be the angle between the two segments incident to p. The following paragraph motivates our definition of the region R'_{pq}.

Assume pq is the segment in the main component connecting the two legs of a sharp corner s (we say that the edge straddles the sharp corner) and let $\bar{\alpha}_s$ be the angle between the two tangents at s. Let s_p be a segment defined by two adjacent sample points on the leg of p; let s_q be a segment defined by two adjacent sample points on the leg of q, both lying in $B(s, \delta'_s/7)$. Let θ_s be the angle formed by the segments. Since either segment forms an angle less than $\bar{\alpha}_s/4$ with the corresponding tangent at s, we have $\bar{\alpha}_s/2 \leq \theta_s \leq 3\bar{\alpha}_s/2$ or $2\theta_s/3 \leq \bar{\alpha}_s \leq 2\theta_s$, i.e., θ_s is a good estimator for $\bar{\alpha}_s$. Since again the angle between any tangent on a leg and the appropriate tangent in the corner is at most $\alpha_s/4$, we know that the angle between the corner point and the points p and q is between $\bar{\alpha}_s/2$ and $3\bar{\alpha}_s/2$, thus between $\theta_s/3$ and $3\theta_s$.

We come to the definition of R'_{pq}. For an edge pq let p^+ and q^+ be the other neighbors of p and q, respectively, and let θ_s be the angle between the segments pp^+ and qq^+, i.e., $\theta_s = \beta_p + \beta_q - \pi$. We define R'_{pq} as the set of all points r with

- $\angle(\overrightarrow{rp}, \overrightarrow{rq}) \geq \theta_s/3$,
- $\angle(\overrightarrow{pr}, \overrightarrow{pq}) \leq \pi - \beta_p + \min(\pi/20, \theta_s)$ and r lies on the opposite halfspace with respect to the line pq as p^+, and

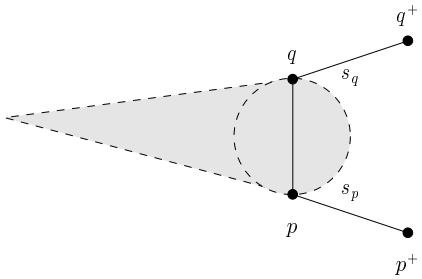


FIG. 7.4. *The definition of R_{pq}.*

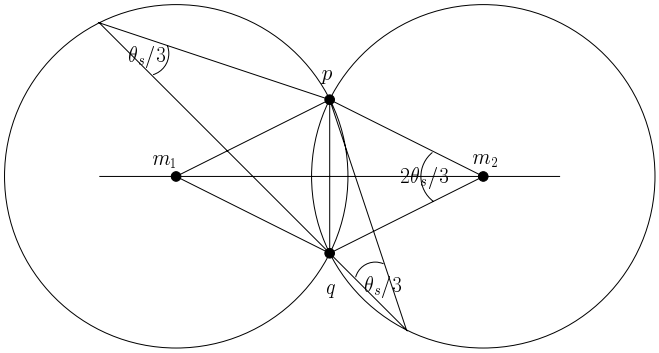


FIG. 7.5. *The inscribed angle theorem: The central angle is equal to twice the inscribed angle.*

- $\angle(\overrightarrow{qr}, \overrightarrow{pq}) \geq \pi - \beta_q + \min(\pi/20, \theta_s)$ and r lies on the opposite halfspace with respect to the line qp as q^+.

For an illustration of this definition see Figure 7.4. Note that $\pi - \beta_q + \min(\pi/20, \theta_s) \leq \pi - \beta_q + \theta_s = \beta_p \leq \pi$.

LEMMA 7.6. *If the main component of a curve does not yet contain all sample points from the curve, it will grow.*

Proof. The main component contains all points in T'. Consider a sample point on γ_j which does not belong to the main component and let p and q be its adjacent sample points in the main component. Then $\{p, q\} \subseteq B(s, 2\delta'_s/60)$ for some sharp corner. If p and q lie on the same leg of s the centerball of pq contains the subcurve between p and q and if pq straddles the sharp corner, the region R'_{pq} contains the subcurve. ∎

To prove that we do not merge components that do not belong to the same curve, we need the following three lemmas.

LEMMA 7.7. *Every point r in R'_{pq} has distance at most $\|pq\|/\sin(\theta_s/3)$ from p and q.*

Proof. Let m_1 and m_2 be the points on the perpendicular bisector of pq with distance $\|pq\|/(2\sin(\theta_s/3))$ from p and q (see Figure 7.5).

By the inscribed angle theorem, every point which sees pq under an angle of at least $\theta_s/3$ lies inside the union of the the balls with center m_1 or m_2 through p and q. Thus at any point in the region R_{pq} that distance is at most $\|pq\|/\sin(\theta_s/3)$ from p and q. ∎

LEMMA 7.8. $\theta_s \geq \alpha_s/2$.

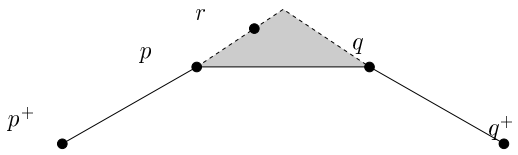


FIG. 7.6. *The segments rp and rq form a small angle with the segment pq. Thus $\|pr\|$ cannot be much larger than $\|pq\|$.*

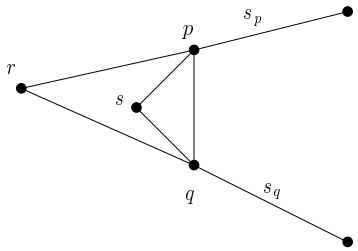


FIG. 7.7. *The two triangles pqr and pqs.*

Proof. If the segment pq straddles the corner, $\theta_s \geq \alpha_s/2$, since pp^+ lies completely on one leg and qq^+ lies completely on the other leg. If p^+ and q^+ are on the same leg, this follows directly from the sampling condition. If the segment pp^+ straddles the corner, the angle formed by the segments pp^+ and qq^+ is smaller than the angle formed by the segments sp^+ and qq^+, which is at least $\alpha_s/2$. □

LEMMA 7.9. *Let pq be a segment of the polygonal reconstruction, with $p, q \in T$ and $d(p) \geq d(q)$. Then $d(p) \geq 3\|pq\|/2$.*

Proof. Assume otherwise. Than $c_\mu(p,q) = \|pq\| - d(p)/3 - d(q)/3 \geq \|pq\| - 2d(p)/3 > \|pq\| + \|pq\| \geq 0$. □

We now turn to the proof that we do not merge a curve with points from another component.

LEMMA 7.10. *The region R_{pq} of an edge pq in the polygonal reconstruction of the main component contains no sample point of another component.*

Proof. This is obvious for the center ball of pq. We turn to the region R'_{pq}.

Assume first p and q are in T' and w.l.o.g. $d(p) \geq d(q)$. Assume there is a point r outside the the $B(p, d(p))$ ball in R'_{pq}. Then $\|pr\| \geq d(p) \geq 3\|pq\|/2$ by Lemma 7.9.

We know $\angle(\overrightarrow{pq}, \overrightarrow{pp^+}) \geq 2\pi/3$ and $\angle(\overrightarrow{pp^+}, \overrightarrow{pr}) \geq 19\pi/20$. Hence $\angle(\overrightarrow{pq}, \overrightarrow{pr}) \leq (\pi - 2\pi/3) + (\pi - 19\pi/20) = 23\pi/60$; see Figure 7.6. Analogously $\angle(\overrightarrow{qp}, \overrightarrow{qr}) \leq 23\pi/60$, thus $\angle(\overrightarrow{rp}, \overrightarrow{rq}) \geq 7\pi/30$. So[7] $\|pr\| \leq \sin(23\pi/60)/\sin(7\pi/3)\|pq\| < 3\|pq\|/2$.

Next assume that one of p and q does not belong to T'.

We will show that $R'_{pq} \subseteq B(s, \delta'_s)$. Let r be any point in R'_{pq} and let θ be the angle under which r sees the segment pq. Then $\theta \geq \theta_s/3$ by the definition of R'_{pq}.

First assume $\theta_s \geq \pi/6$. We know $\|pr\| \leq \|pq\|/\sin(\theta_s/3) \leq 6\|pq\|$. Thus $\|sr\| \leq 6(8\delta'_s/60) + 2\delta'_s/60 < \delta'_s$.

Now assume $\theta_s < \pi/6$. Look at the triangle $\triangle pqr$ and assume w. l. o. g. $\pi - \beta_p > \pi/2$ (see Figure 7.7). We show that the corner point s is almost as far away from p as r. The angle at r is at least $\theta_s/3$; the angle at p is at most $\pi + \theta_s - \beta_p$. Thus

[7] In this proof we make frequent use of the fact that $a/\sin\alpha = b/\sin\beta = c/\sin\gamma$ for a triangle with sides a, b, c and corresponding angles α, β, and γ.

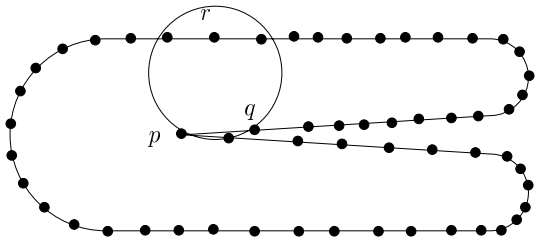


FIG. 8.1. *The edge pq does not belong to the Delaunay triangulation; p is a sharp corner with* $\delta_p = \|pr\|/2$. *Also* $c_p \approx 2\delta_p/5 = \|pr\|/5$. *If* $\|pq\| < \|pr\|/5$, *the edge pq has negative reduced cost. Thus our sampling condition is satisfied.*

$\|pr\| \leq \|pq\| \sin(\pi + \theta_s - \beta_p)/\sin(\theta_s/3)$.

The sharp corner s forms an angle of at most $3\theta_s$ with points p and q, since $\bar{\alpha}_s \leq 2\theta_s$ by Lemma 7.8. The angle at p of the triangle $\triangle pqs$ is between $\pi + \theta_s - \beta_p$ and $\pi - \theta_s + \beta_p$.

We have to distinguish two cases according to β_p. First assume $\beta_p > \pi - 2\theta_s$. We conclude $\|rs\| \leq \|ps\| + \|pr\| \leq 2\delta'_s/60 + \|pq\| \sin(\pi + \theta_s - \beta_p)/\sin(\theta_s/3) \leq 2\delta'_s/60 + \|pq\| \sin(3\theta_s)/\sin(\theta_s/3) \leq 2\delta'_s/60 + 10\|pq\| \leq 2\delta'_s/60 + 40\delta'_s/60 \leq \delta'_s$.

Now assume $\beta_p < \pi - 2\theta$. We conclude $\|rs\| \leq \|ps\| + \|pr\| \leq 2\delta'_s/60 + \|pq\| \sin(\pi + \theta_s - \beta_p)/\sin(\theta_s/3) \leq 2\delta'_s/60 + \|ps\|(\sin(3\theta)/\sin(\pi - \theta_s + \beta_p))(\sin(\pi + \theta_s - \beta_p)/\sin(\theta_s/3)) \leq 2\delta'_s/60 + \|ps\|(\sin(3\theta_s)/\sin(\theta_s))(\sin(3\theta_s)/\sin(\theta_s/3)) \leq 2\delta'_s/60 + 27\|ps\| \leq 2\delta'_s/60 + 52\delta'_s/60 < \delta'_s$. □

8. Curve reconstruction and the Delaunay diagram. Most previous curve reconstruction algorithms use sampling conditions that guarantee that the polygonal reconstruction is a subset of the Delaunay diagram. Our sampling condition does not imply that the traveling salesman tour is a subgraph of the Delaunay triangulation; see Figure 8.1. This fact can be interpreted positively and negatively: positively, as an indication of the strength of the traveling salesman-based reconstruction, and negatively, since the optimal traveling salesman tour must be searched for in the complete graph on the sample set. In this section we show that a slight strengthening of our sample condition implies that the polygonal reconstruction is contained in the Delaunay diagram.

Additional condition on the sample set. An edge pq of the polygonal reconstruction with an endpoint not in T has length at most $4\delta_s \sin(\alpha_s/2)/5$, where s is the sharp corner with $\{p, q\} \subseteq B(s, \delta_s)$.

LEMMA 8.1. *If the sample set V satisfies the strengthened sampling condition, the polygonal reconstruction is contained in the Delaunay diagram of V.*

Proof. Let pq be an edge of the polygonal reconstruction. We construct a Delaunay ball B for it.

First assume that $\{p, q\} \subseteq T$. The center ball of pq is contained in $B(p, d(p))$ and hence empty of sample points; otherwise, γ would either intersect the ball in more than one component or turn by more than $\pi/3$ within the ball.

Next assume that one of the endpoints of the edge, say, p, does not lie in T. Let s be the sharp corner with $p \in B(s, \delta_s)$. We distinguish the cases whether p and q are on the same leg of s or not.

First assume that p and q lie on different legs; see Figure 8.2. We put the center m of B into the halfspace containing s and having p and q in its boundary and set

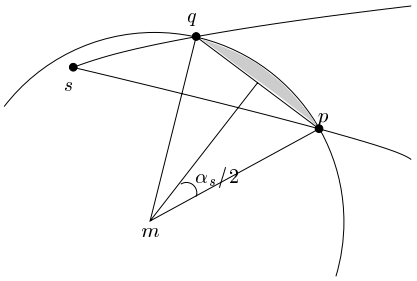


FIG. 8.2. *The sample x must lie on the upper leg behind q and in the shaded lune.*

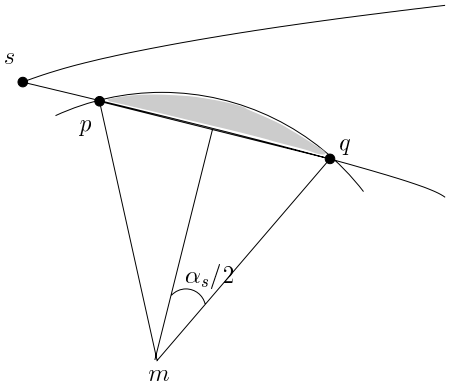


FIG. 8.3. *The sample x must lie on the upper leg and in the shaded lune.*

the radius r of B to $(\|pq\|/2)/\sin(\bar{\alpha}_s/2) \leq 2\delta_s/5$; the upper bound on the radius follows from the strengthened sampling condition. Since one of p or q is contained in $B(s, \delta_s/5)$, we conclude $B(m, r) \subseteq B(s, \delta_s)$.

Now assume that there is a sample point $x \in B(m, r)$. We discuss the case that x lies on the same leg as q and leave the other case to the reader. Since p and q are adjacent samples, x cannot lie on the segment sq and hence $\angle(\vec{sp}, \vec{qx}) = \angle(\vec{sp}, \vec{sq}) + \angle(\vec{sq}, \vec{qx}) \leq \angle(\vec{qp}, \vec{sq}) + \angle(\vec{sq}, \vec{qx}) = \angle(\vec{qp}, \vec{qx})$. (We have $\angle(\vec{sp}, \vec{sq}) < \angle(\vec{qp}, \vec{sq})$, since moving along the segment $\vec{sq}$ increase the angle.) Since x lies in the lune of $B(m, r)$ defined by pq, we have $\angle(\vec{qx}, \vec{qp}) \leq \bar{\alpha}_s/2$. Thus $\angle(\vec{sp}, \vec{qx}) \leq \bar{\alpha}_s/2$. On the other hand, the angle between the two tangents at s is $\bar{\alpha}_s$ and hence $\angle(\vec{sp}, \vec{qx}) > \bar{\alpha}_s - 2\bar{\alpha}_s/4 = \bar{\alpha}_s/2$, a contradiction.

Next assume that p and q lie on the same leg; see Figure 8.3. The center m of B lies on the same side of the angular bisector of the cone defined by the tangents $\mathrm{t}_r(s), -\mathrm{t}_l(s)$ as p and q and sees the segment pq under an angle of $\bar{\alpha}_s$. As above, we conclude that $B(m, r) \subseteq B(s, \delta_s)$. Then x must be contained in the lune of $B(m, r)$ defined by the segment pq. Since p and q are adjacent sample points, x must lie on the other leg. Assume w.l.o.g. that p is closer to s than q. We have $\angle(\vec{pq}, \vec{px}) \leq \bar{\alpha}_s/2$ and hence $\bar{\alpha}_s = \angle(\mathrm{t}_r(s), -\mathrm{t}_l(s)) < \angle(\vec{ps}, \vec{px}) + 2(\bar{\alpha}_s/4) \leq \bar{\alpha}_s$, a contradiction. □

9. Monotonicity. Intuitively, a larger sample set makes the reconstruction task simpler. We discuss how various sampling conditions and reconstruction algorithms behave with respect to larger sample sets.

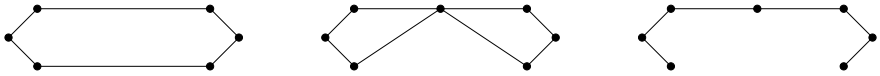


FIG. 9.1. *For the sample set in the left figure, the algorithms in* [3, 8, 15] *produce the hexagon, as shown. If one adds a sample point in the middle of one of the long segments, the algorithm in* [8] *produces the output of the middle figure, and the algorithms in* [3, 15] *produce the output of the right figure. Thus none of these algorithms is self-consistent.*

A sampling condition is called *monotone*[8] if any superset of a set satisfying the sampling condition also satisfies the sampling condition.

For closed curves and open curves with specified endpoints, our sampling condition is monotone. For open curves with unspecified endpoints the superset must satisfy the additional constraint that the additional points must lie in $\gamma[a, b]$, where a and b are the extreme sample points in the old sample. The sampling conditions used in the papers [3, 8, 15, 9] are also monotone; again the additional constraint is needed for open curves. The conditions in [13] are not monotone.

All algorithms mentioned in the present paper come with a guarantee: if the curve γ is from a certain class of curves and the sample set V is sufficiently dense, the algorithm will reconstruct γ. It is not specified what the algorithm does, if the hypothesis of the theorem is not satisfied. The algorithm may either fail, i.e., indicate that it could not find a curve, or "invent" a curve. From a practical point of view this situation is unsatisfactory as a user has in general no way of distinguishing reconstruction from invention. The situation is aggravated by the fact that the sampling densities required by the theorems are quite high and that the algorithms tend to work for smaller densities and hence are likely to be used in situations not covered by the theorems. *It would be nice to have algorithms that never invent curves.*

A reconstruction algorithm is called *self-consistent* if it has the following property. On an input V it either outputs `FAILURE` or `SUCCESS`. In the latter case it also outputs a curve Γ passing through V such that for any sample V' from Γ with $V \subseteq V'$, it will also output Γ. A reconstruction algorithm that is not self-consistent can change its mind if given additional sample points that seem to confirm the output of the algorithm.

THEOREM 9.1. *The algorithm in* [9] *and the traveling salesman-based algorithm are self-consistent; the algorithms in* [3, 8, 15, 13] *are not self-consistent.*

Proof. The algorithm in [9] is constructed to be self-consistent. For the algorithms in [3, 8, 15, 13] it is easy to come up with examples that show non-self-consistency (see Figure 9.1).

It remains to show self-consistency for the traveling salesman-based algorithm. We show that if the solution for the subtour-LP for a set V is unique and integral, then the same is true for the subtour-LP for $V \cup \{z\}$ for any z on an edge of the integral solution of the subtour-LP of V (and different from all points in V).

The claim is a simple consequence of the so-called splitting-off lemma (see [17, Problem 6.53]). Consider an optimal solution of the subtour-LP for $V \cup \{z\}$. The value C of this solution is at most the length C_0 of the optimal tour of V (since the tour for V is also a tour for $V \cup \{z\}$). The splitting-off lemma allows us to construct a solution for the subtour-LP for V from the solution for $V \cup \{z\}$. It implies the existence of a set of triples (e, f, r), where e and f are edges incident to z and r is a nonnegative real, such that

[8] Be aware, this choice of name reflects a bias on behalf of the authors.

- for each edge e incident to z the sum of the third components of all triples containing e is equal to x_e (the value of the edge e in the subtour-LP for $V \cup \{z\}$) and
- a solution for the subtour-LP for V can be obtained by modifying the solution for the subtour-LP for $V \cup \{z\}$ as follows: for each edge uv with $u \neq z$ and $v \neq z$ increase x_{uv} by the sum of the third components of all triples (uz, zv, r). Delete all edges incident to z.

For any edge $e = uv$ let y_e be the increase of x_e in this construction. The cost of the obtained solution is $C + \sum_{uv} y_{uv}(c_{uv} - c_{uz} - c_{zv})$. This cost is at most C (and hence at most C_0) since $c_{uv} \leq c_{uz} + c_{zv}$ by the triangle inequality. Since the solution for the subtour-LP for V is unique and is equal to the tour for V, the cost of the solution cannot be smaller than C_0 and hence for any edge $e = uv$ with $y_{uv} > 0$ we must have $c_{uv} = c_{uz} + c_{zv}$, i.e., z lies on the line segment $\overline{uv}$. Moreover, $y_e + x_e$ must be integral for every edge $e = uv$.

In the tour for V there is only one edge passing through z (since optimal tours are non-self-intersecting) and hence there can be only one edge uv with $y_{uv} > 0$. Thus our set of triples consists of a single triple (uz, zv, r) and since the degree constraint at z must be satisfied for the optimal solution of the subtour-LP for $V \cup \{z\}$ we conclude that $r = 1$. We conclude that the optimal solution of the subtour-LP for $V \cup \{z\}$ is unique and integral. □

10. Our sample condition and the local feature size. The papers [3, 8, 15, 9] investigated the reconstruction problem for smooth curves. A curve is smooth if it is twice-differentiable. They expressed the sampling condition in terms of the so-called *local feature size*. The local feature size $f(p)$ at a curve point p is the distance of p from the medial axis of γ. The *medial axis* of a curve is the closure of the set of points in the plane which have at least two nearest (with respect to the Euclidean metric) points on the curve. They required a sampling condition of the form: For any $p \in \gamma$ there must be a sample point $v \in V$ with $\|pv\| \leq \epsilon \cdot f(p)$; here ϵ is a parameter which depends on the algorithm. All algorithms require $\epsilon \leq 1/2$.

The experimental results in [2] suggest that the traveling salesman-based algorithms works for sparser sample sets than the algorithms mentioned above. We do not know whether this observation is a fact and can prove only a much weaker result.

LEMMA 10.1. *Let γ be a smooth curve and $\epsilon < 1/10$. If for any $p \in \gamma$ there is a sample point $v \in V$ with $\|pv\| \leq \epsilon \cdot f(p)$, then V satisfies our sampling condition.*

Before we prove Lemma 10.1, we show the following lemma.

LEMMA 10.2.

$$f(p) < 3d(p).$$

Proof. The following fact was shown in [3].

FACT 5. *Let r be a point of a smooth curve γ. Furthermore let q and s be points on γ with distance less than $f(r)$ from r with $q < r < s$. Then $\angle(\vec{rq}, \vec{rs}) > \pi/3$.*

By the definition of $d(p)$, either $B(p, d(p)) \cap \gamma$ is not connected or $B(p, d(p))$ contains three points turning by $\pi/3$.

In the first case, there is a medial axis point in $B(p, d(p))$ by Lemma 1 of [3] and hence $f(p) \leq d(p)$.

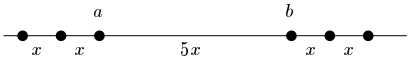


FIG. 11.1. *The underlying part of the curve is a line segment. Hence, there is no condition on the sample set. In a necklace tour the disks centered at a and b have radius at most* $2x$. *Thus the two disks will not intersect.*

We turn to the second case. Let $q < r < s \in B(p, d(p)) \cap \gamma$ forming an angle of $\pi/3$. By Fact 5 we conclude $f(r) \leq \max(\|qr\|, \|rs\|)$. Thus $f(p) \leq \|pr\| + \max(\|qr\|, \|rs\|) < 3d(p)$. □

Proof of Lemma 10.1. We have to show that for an ϵ-sampled curve, with $\epsilon < 1/10$, the modified cost of the edge between two adjacent sample points is less then 0.

Let p and q be adjacent sample points. Let w.l.o.g. $d(p) \leq d(q)$. Lemma 3.4 of [8] states $\|pq\| \leq 2\epsilon f(p)/(1-\epsilon) = 1/5 f(p)/(1-1/10)$. Thus $c_\mu(p,q) = \|pq\| - d(p)/3 - d(q)/3 < 1/5 f(p)/(1-1/10) - f(p)/9 - f(p)/9 \leq 0$. □

Our sample condition depends on several parameters and we have set these parameters to particular values in section 3. In section 3.6 we discussed the dependency between the parameters. For smooth curves we can set f_{scale} to $1/2$ and then the argument above works for $\epsilon = 1/7$.

11. Necklace tours. We have shown that curve reconstruction gives rise to a polynomially solvable case of the Euclidean traveling salesman problem. In this section we relate our results to a known solvable case, the so-called *necklace tours*. Let V be a set of points in the plane and assume that there is a set of disks centered at the points in V so that each disk intersects with exactly two other disks and so that the intersection graph of the disks is connected. The intersection graph of the disks defines a tour on V; the two neighbors of a point v correspond to the two disks that intersect the disk associated with v. The tour is called a necklace tour and is known to be an optimal traveling salesman tour of V; see [6].

We cannot claim that necklace tours are a special case of our result, since there is no curve underlying a necklace tour. The optimality proof for necklace tours is a special case of our argument. We simply define the potential of any point v as the radius of the disk associated with v. Then exactly the edges in the tour have nonpositive cost. Any tour which uses an edge outside the necklace tour must include edges of positive cost and can include only a subset of the edges in the necklace tour. This implies that the necklace tour is optimal.

Figure 11.1 shows an example of a traveling salesman problem which is covered by Theorem 2.3 and whose solution is not a necklace tour.

12. Conclusions. We have shown that traveling salesman-based curve reconstruction permits a polynomial time algorithm. In the companion paper [2] we show that traveling salesman-based curve reconstruction is reasonably efficient and works for smaller sampling density than other reconstruction algorithms. We have also shown how to extend traveling salesman-based reconstruction to collections of closed curves. The generalization to collections of open and closed curves stays open.

Surface reconstruction is the three-dimensional analogue of curve reconstruction. There are no surface reconstruction algorithms that can handle nonsmooth surfaces. Whether global minimization can also be applied to surface reconstruction remains an open question. A first step in this direction was recently taken by [1].

REFERENCES

[1] U. Adamy, J. Giesen, and M. John, *New techniques for topologically correct surface reconstruction*, in IEEE Visualization 2000, to appear.

[2] E. Althaus, K. Mehlhorn, S. Näher, and S. Schirra, *Experiments on curve reconstruction*, in ALENEX, 2000, pp. 103–114; also available online from www.mpi-sb.mpg.de/mehlhorn/ftp/exp-curve.ps.

[3] N. Amenta, M. Bern, and D. Eppstein, *The crust and the β-skeleton: Combinatorial curve reconstruction*, Graphical Models and Image Processing, 60 (1998), pp. 125–135.

[4] D. Attali, *r-regular shape reconstruction from unorganized points*, in Proceedings of the 13th Annual ACM Symposium on Computational Geometry (SCG'97), Nice, France, 1997, pp. 248–253.

[5] F. Bernardini and C. Bajaj, *Sampling and reconstructing manifolds using α-shapes*, in Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG'97), Queen's University, Kingston, ON, Canada, 1997, pp. 193–198.

[6] R. E. Burkard, V. G. Deineko, R. van Dal, J. A. A. van der Veen, and G. J. Woeginger, *Well-solvable special cases of the traveling salesman problem: A survey*, SIAM Rev., 40 (1998), pp. 496–546.

[7] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley, New York, 1998.

[8] T. Dey and P. Kumar, *A simple provable algorithm for curve reconstruction*, in Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99), SIAM, Philadelphia, 1999, pp. 893–894.

[9] T. Dey, K. Mehlhorn, and E. Ramos, *Curve reconstruction: Connecting dots with good reason*, Comput. Geom., 15 (2000), pp. 229–244; also available online from www.mpi-sb.mpg.de/mehlhorn/ftp/cure.ps.gz.

[10] H. Edelsbrunner, *Shape reconstruction with the Delaunay complex*, in LATIN'98: Theoretical Informatics, Lecture Notes in Comput. Sci. 1380, Springer-Verlag, 1998, pp. 119–132.

[11] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, *On the shape of a set of points in the plane*, IEEE Trans. Inform. Theory, 29 (1983), pp. 71–78.

[12] L. Figueiredo and J. Gomes, *Computational morphology of curves*, Visual Computer, 11 (1994), pp. 105–112.

[13] S. Funke and E. Ramos, *Reconstructing collections of curves with corners and endpoints*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01), SIAM, Philadelphia, 2001, pp. 344–353.

[14] J. Giesen, *Curve reconstruction, the TSP, and Menger's theorem on length*, in Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SCG'99), Miami, FL, 1999, pp. 207–216.

[15] C. Gold, *Crust and anti-crust: A one-step boundary and skeleton extraction algorithm*, in Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SCG'99), Miami, FL, 1999, pp. 189–196.

[16] D. Kirkpatrick and J. Radke, *A framework for computational morphology*, in Computational Geometry, G. Toussaint, ed., North-Holland, Amsterdam, 1985, pp. 217–248.

[17] L. Lovász, *Combinatorial Problems and Exercises*, North-Holland, Amsterdam, 1979.

[18] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, Chichester, 1986.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 67–85

TREE SEARCH ON AN ATOMIC MODEL FOR MESSAGE PASSING*

PANGFENG LIU†, WILLIAM AIELLO‡, AND SANDEEP BHATT§

Abstract. This paper presents a simple atomic model of message-passing multicomputers. Within one synchronous time step each processor can receive one atomic message, perform local computation, and send one message. When several messages are destined to the same processor, then one is transmitted and the rest are blocked. Blocked messages cannot be retrieved by their sending processors; each processor must wait for its blocked message to clear before sending more messages into the network. Depending on the traffic pattern, messages can remain blocked for arbitrarily long periods.

The model is conservative when compared with existing message-passing systems. Nonetheless, we prove linear message throughput when destinations are chosen at random; this rigorously justifies an instance of folklore. Based on this result we also prove linear speedup for backtrack and branch-and-bound searches using simple randomized algorithms.

Key words. parallel tree search, parallel communication model, randomized algorithm

AMS subject classifications. 68W10, 68W20, 68Q10

PII. S0097539799358070

1. Introduction.

The message-passing style of programming is widely used on almost all parallel computers. The primitives to *send* and *receive* messages hide low-level architectural details and are ideal for programming many large applications. While message-passing systems have been in use for over a decade, relatively few results concerning the complexity of message-passing protocols are available. One reason for this discrepancy is the lack of theoretical models that appropriately capture issues related to communication; as stated in [5], most theoretical models "encourage exploitation of formal loopholes, rather than rewarding development of techniques that yield performance across a range of current and future parallel machines."

We propose an atomic model [18] to study the performance of message-passing programs. The model is simple and much more restricted in its capabilities in comparison with existing systems. Nevertheless, we show that it allows simple and efficient solutions (linear speedup) for message scattering and backtrack and branch-and-bound searches.

1.1. Message-passing systems.

Message-passing instructions appear in two varieties: *blocking* and *nonblocking*. Blocking instructions require synchronization between the sender and receiver: a send instruction terminates only when the corresponding receive is executed by a remote process. One advantage of blocking instructions is that no system buffering is required. However, the delay in waiting for a send

*Received by the editors June 30, 1999; accepted for publication (in revised form) November 15, 2000; published electronically May 31, 2001. A preliminary version of this paper appeared in the 5th Annual ACM Symposium on Parallel Architectures and Algorithms, Velen, Germany, 1993, pp. 154–163. The first and third authors were supported in part by NSF/DARPA grant CCR-89-08285, DARPA contract DABT 63-91-C-0031 monitored by Army DOC, and Air Force grant AFOSR-89-0382 at Yale University.

http://www.siam.org/journals/sicomp/31-1/35807.html

†Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106, Republic of China (pangfeng@cs.ccu.edu.tw).

‡AT&T Labs Research, AT&T Shannon Center, 180 Park Avenue, Florham Park, NJ 07932 (aiello@research.att.com).

§Akamai Technologies, Inc., Cambridge, MA 02139 (bhatt@akamai.com). The work of this author at Rutgers University was supported by ONR grant N00014-93-1-0944.

instruction to complete means that computation and communication cannot overlap; this can reduce overall performance significantly. Another disadvantage is that the programmer must carefully arrange send/receive instruction pairs to avoid deadlock.

Nonblocking instructions allow a process to execute multiple send instructions before any of the corresponding receive instructions is executed. This allows for the possibility of increased efficiency since communication and computation can overlap. However, more system resources, buffering and bandwidth, for example, are required for a nonblocking scheme; otherwise *pending* messages (those sent but not yet received) will be excessively delayed or potentially lost. Moreover, since system resources are finite, the programmer must ensure that the number of pending messages is bounded at all times.

Underneath the message-passing abstractions, a message goes through several phases before it is absorbed at its destination. During each phase it requires some critical system resource to continue its journey. For example, a memory buffer is required to compose a message. When a message buffer is sent, it goes through the network interface connecting the processor to the network. Before the message arrives at the destination, it travels in the network and occupies network buffers. On reaching its destination the message occupies a buffer at the network interface before it is removed and processed. Whenever a message cannot get the critical resource it needs, it must wait. When messages wait a long time, there is the danger that communication delays can cause processor idling, thereby reducing overall performance greatly.

In many applications it is also common practice to reduce communication costs (primarily due to system overheads) by aggregating data into fewer, but longer, atomic messages [3].[1] First, the sender notifies the receiver of the message length. Upon receipt of this notification, the receiver allocates sufficient buffer space and sends back an acknowledgment. This establishes a link between the sender and the receiver and the message is transmitted in the third step. Once again, there is ample opportunity for delay from the time the protocol is initiated until the time the data is actually transferred.

Given the limited resources of multicomputer systems, it is natural to ask whether the efficiency gained by using nonblocking instructions is lost if the number of pending messages is limited. Nevertheless, we show that nonblocking communication can still achieve high performance, even with very limited communication resources.

We investigate this question formally within the atomic model which permits only one pending message per processor. In brief, each processor is given one send buffer and one receive buffer, each capable of holding one atomic message. The system alternates between message transmission and computation cycles. During a computation cycle a processor retrieves a message from its receive buffer, performs a computation, enqueues newly generated messages into a message queue, and writes the first message in the queue into the send buffer if the send buffer is empty. During the transmission cycle, the network attempts to transmit every message in each send buffer to the receive buffer of the destination. If more than one message is destined for the same processor, exactly one is successfully transmitted. The rest remain in their send buffers. The one which is transmitted is chosen by a *network arbiter*. The *worst-case* arbiter makes choices to maximize the running time. The first in first out (FIFO) arbiter gives priority to messages with smaller time-stamps; messages with the same time-stamp can be delivered in arbitrary order.

[1] Atomic messages travel through a critical resource as a single entity; different messages do not coexist inside the critical section.

The atomic model is motivated by the desire to analyze the performance of message-passing programs in an architecture-independent manner. For this reason, we have chosen to abstract the network as an arbiter which takes one unit of time to transfer messages from send buffers to receive buffers at the destination. We believe this is reasonable in applications that involve the atomic transfer of large data sets. Unit-delay assumptions are also made in the literature on PRAMs and complete networks [13, 14]. Unlike these models, however, we explicitly account for message contention and do not allow multiple messages to be received in one step by a processor. The issue of contention at the receiving module is also addressed in models for optical communication [8] and module parallel computers [12, 20]. A key feature which distinguishes the atomic model is that once a message has been sent it cannot be retrieved; the sending processor must wait for the network to clear the send buffer after the message has been copied into the receive buffer at the destination. Finally, the atomic model can be viewed as the limiting version of the LogP model [5]; with long messages of equal length, the latency, overhead, and gap parameters of the LogP model can be lumped into a single unit time delay.

Communication contention is an important issue in modeling parallel computations. Valiant's bulk synchronous parallel (BSP) model [24] divides the parallel computation into supersteps in which processors perform local computation and exchange data. All the outstanding communication requests will be serviced before the next superstep starts. The memory contention is characterized by the length of the time interval a processor must wait before sending the next message. To model the fact that many shared memory machines now have a large number of memory banks in order to serve relatively much faster processors, Blelloch et al. [4] extended BSP into (d, x)-BSP by adding two parameters—the memory bank delay (the minimum interval length a memory bank can serve memory requests) and the ratio of the number of memory banks to processors. The QSM model [9] also characterizes the contention problem by a bandwidth parameter g so that a processor can issue memory requests only once every g steps. All these models characterize the contention by limiting the communication capability on a *per processor* basis. In contrast, the QSM(m) model by Adler et al. [1] added another parameter to describe the limitation on the communication capability for the *entire* system.

Besides characterizing the memory contention by a memory bandwidth parameter, it is also possible to model the contention by allowing atomic access to a shared resource. Dwork, Herlihy, and Waarts [6] proposed a model for shared memory access in which simultaneous accesses to a single memory location are serialized and only one will succeed at a time. However, a process may have multiple pending operations due to trying to access different memory locations. Gibbons, Matias, and Ramachandran [11] proposed a queue-read, queue-write (QRQW) model that allows concurrent reading and writing to the same memory location to be queued. Similar to BSP, the computation is divided into supersteps and all the queued memory requests in one superstep will be served before the next one. The asynchronous variant of QRQW (AQRQW) [10] relaxes the bulk synchronous requirement of QRQW and BSP and more accurately captures the memory contention phenomenon in modern shared-memory parallel computers [10]. The atomic model we propose differs from all the previous *message-passing* models in that it allows only one pending outstanding "request" *per processor.*[2]

[2]The SIMD-QRQW model in [11] also allows only one pending request per processor in shared memory.

Despite the restriction on the rate at which the network can deliver messages to a destination, as well as the adversarial nature of the arbiter, we show that simple randomized algorithms can attain linear speedup for branch-and-bound and backtrack tree searching. Furthermore, all-to-some message passing can finish within a constant factor of the optimal time with high probability if the destinations are uniformly distributed among the processors.

2. The atomic message-passing model. We model a message-passing multicomputer as a collection of p nodes connected via an interconnection network [23]. For convenience of analysis we require that the system be synchronous and operate in discrete time steps.[3] Each time step is divided between one communication step and one node computation step.

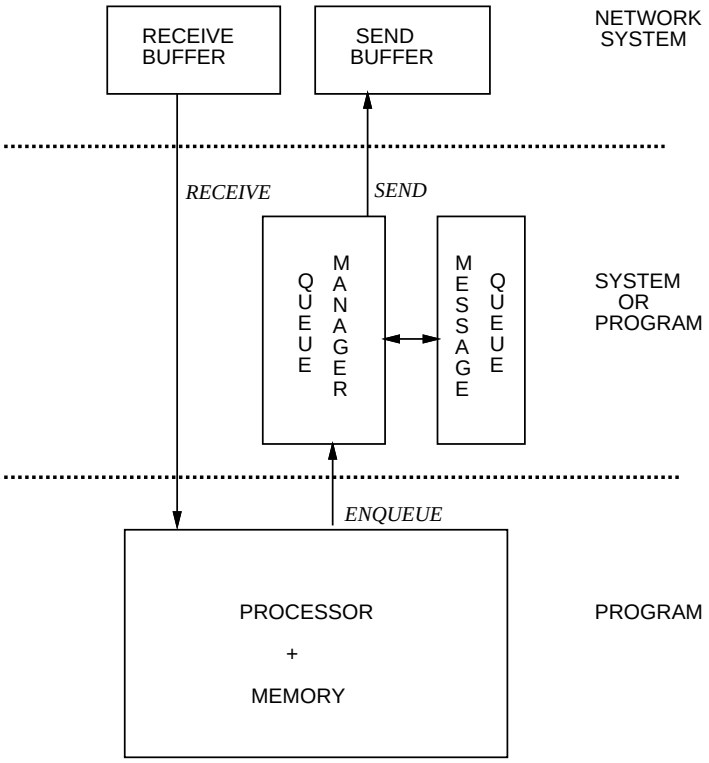


FIG. 1. *The structure of a node.*

Each node consists of a receive buffer, a processor, local memory, a queue manager, a message queue, and a send buffer (Figure 1). Each buffer can hold one atomic message. Every node can perform local computation using its processor and local memory. It can also receive a message using the receive buffer and enqueue messages into the message queue. The message queue is maintained by a queue manager which may be under the control of the processor or the system. A message from the message queue is injected into the network by placing it into the send buffer. For our purposes, it is convenient to model the actions at a node as repeated executions of the following *reactive cycle* which occurs during one synchronous time step. Notice that we consider

[3]This assumption is not required for termination but simplifies the analysis of throughput.

the communication as a sequence of time steps, and the timing of an event is expressed as the time step number at which the event occurs, not a particular point in the absolute time line. Similarly, a time interval refers to a contiguous set of time steps, not an interval in terms of wallclock time.

1. *The send phase* (performed by the processor).
   Transfer the message at the head of the queue into the send buffer if it is empty.
2. *The transmission phase* (performed by the network system).
   Take messages from send buffers to receive buffers according to message destinations. If more than one message is destined for the same receive buffer, the one which succeeds is selected by the network arbitration policy.
3. *The receive phase* (performed by the processor).
   Probe the receive buffer to receive an incoming message, if any, into local memory.
4. *The local computation phase* (performed by the processor).
   - COMPUTE. Perform local computation, possibly on the newly received message, and generate new messages.
   - ENQUEUE. Pass the newly generated messages to the queue manager, which will place it into an appropriate place in the message queue.

Observe that there are two ways a message can be delayed. First, a message may have to wait in the message queue until it is selected to be placed in the send buffer. Second, once a message is in the send buffer, it may be delayed in the network. We call the second kind a *receive-delay*.

When more than one message, occupying send buffers of different nodes, are simultaneously destined for the same node, the network must deliver one message. Since every node executes a RECEIVE instruction during its reactive cycle, this requirement of the network satisfies the network contract of the CM-5 [17]: "The data network promises to eventually accept and deliver all messages injected into the network by the processors as long as the processors promise to eventually eject all messages from the network when they are delivered to the processors." With the reactive cycle and the network contract we are assured that deadlocks cannot occur.

We wish to make as few assumptions as necessary on the message queue. Our results for backtrack search are independent of queue maintenance. Our result for branch-and-bound depends on maintaining the message queue as a priority queue.

We also wish to make as few assumptions as necessary about the network arbitration policy when multiple messages are destined for the same node. We will consider two different network arbitration policies. The *worst-case* policy selects the message which maximizes the overall time to complete the task at hand. The FIFO policy dictates that, for any pair of messages with the same destination, they will be accepted in the order of earliest occupancy of their respective send buffers. In other words, if the messages reach their send buffers at different time steps, then the earlier one will be delivered first. If two messages reach their send buffers at the same time, then the order of delivery is arbitrary. Optimal speedup for backtrack search can be achieved even with worst-case arbitration, whereas we require FIFO arbitration to prove optimal speedup for branch and bound search.

3. Overview of results. We will study three problems under the atomic message setting: all-to-some message scattering, backtrack search, and branch-and-bound search. For each of these problems we analyze the case when all messages are destined for independently chosen random nodes. Our intuition is that when messages

are headed for random destinations, the number of conflicting messages is unlikely to become too large. However, when the size of the computation is much larger than the number of processors, this is not always true and one has to prove that the effects of the conflicts do not add up significantly.

The message scattering problem is informally stated as follows: suppose that each node has a list of m messages to send (in order) to remote nodes. How much time does it take, under the worst-case (adversarial) arbitration policy, until all messages are received at their destinations? This problem arises naturally in several applications. In fact, the message scattering problem and the atomic message model are motivated by the "all-to-some" communication in our parallel N-body implementation [19].

In the backtrack search problem, each internal vertex of a search tree, $\mathcal{T}$, corresponds to a partial solution to a problem while each leaf represents a solution with a certain cost. The goal of backtrack search is to find the minimum cost leaf in the search tree. The search tree is not given in advance; it is instead spawned on-line as the search proceeds. The search begins with the root of the tree in a given node; when each internal vertex is expanded, two (or any bounded number of) children are spawned and must each be examined. When a leaf is examined, the cost is calculated and no further expansion along that branch is possible. If the total number of vertices in the search tree is n, and the maximum depth of any leaf is h, it is easy to see that the time to examine all leaves is at least $\Omega(n/p + h)$, where p denotes the number of processors.

Branch-and-bound search is similar to backtrack search, except that only a subtree of the search tree must necessarily be explored. Following Karp and Zhang [13, 14], we model a branch-and-bound tree as a binary search tree, each of whose vertices has an associated cost. The cost of each vertex is strictly less than the cost of each of its children (for simplicity we assume that all vertex costs are distinct). The problem is to find the leaf with minimum cost in the tree. Clearly, every tree vertex whose cost is less than the minimum cost leaf must be expanded because one of its children could potentially be the minimum cost leaf. These vertices form a critical subtree; call it $\mathcal{T}$ of the overall search tree.

As before, the time to complete the search is $\Omega(n/p + h)$, where n is the number of vertices in the *critical* subtree and h is the height of the critical subtree. Noncritical vertices can, in principle, be pruned by the search process and need not be explored.

Tight upper bounds for branch-and-bound, and hence for backtrack search, were given by Karp and Zhang [13] on the complete network which allows multiple messages to be simultaneously received at each node and on the concurrent PRAM which essentially allows unsuccessful writes to be detected. The basic idea was to send each node to a random processor for further exploration. Ranade [21] gave an elegant alternative proof of the Karp–Zhang result. By extending Ranade's techniques we show that the random destination strategy yields linear speedup for backtrack search in the atomic model.

THEOREM 3.1. *Using random destinations, the probability that a binary backtrack search tree of size n and depth h takes time more than $k(n/p + h)$ in the atomic transmission model with worst-case arbiter is polynomially small in n for k sufficiently large.*

Achieving linear speedup for branch-and-bound search in the atomic model is a little harder. The subtle distinction is that pending noncritical vertices can delay pending critical vertices. In the Karp–Zhang model this can never happen. Since we have no control over the number of noncritical vertices, and we do not know the

shape of the critical subtree, it is conceivable that the delays can become arbitrarily large under the worst-case arbiter which consistently favors noncritical vertices over critical vertices. However, under a FIFO arbiter we establish the following result.

THEOREM 3.2. *Let the critical subtree, $\mathcal{T}$, of a branch-and-bound search tree have size n and depth h. Using randomized destinations, the probability that the time, in the atomic model with FIFO arbiter, exceeds $k(n/p + h)$ is polynomially small in n when $n > p^2 \log p$ and k is sufficiently large.*

We present the proof of Theorems 3.1 and 3.2 in sections 6 and 7.

4. Message scattering. The off-line version of the message scattering problem in which the lists can be reordered is easily solved using standard bipartite graph edge-coloring techniques [2, 7]. If r and m are the maximum numbers of messages received and sent by any node, respectively, then $\max\{r, m\}$ steps are necessary and sufficient.

However, the distributed version of the problem, without reordering, is not as simple. We show that with each of p nodes sending m messages (m can be arbitrarily larger than p) the worst-case time is $\Omega(mp)$. In other words, the average throughput of the system is $O(1)$ messages received per time step, independent of the size of the system.

On a positive note, we show that when each of the messages is destined for a randomly chosen node (all destinations independent and uniformly drawn), then, with high probability, the time to completion is $O(m)$. As a result the average throughput is $\Omega(p)$ messages received per time step, asymptotically the maximum possible.

4.1. Lower bounds. Suppose every processor sends n messages to every processor in ascending processor index order. We show that a simple FIFO network arbiter increases the communication time to $\Omega(np^2)$ so that on average only a constant number of messages are received in one time step. The network arbiter ensures that the messages are received in FIFO order; the message sent first is received first. The messages sent at the same time are received in increasing processor index order.

Figure 2 shows the history of four processors sending two messages to each destination in ascending processor index order. A square in the intersection of row i and column j indicates that processor p_i successfully sends a message at time step j. The numbers in the squares are the processor index of the destination. Notice that two successful sends to the same destination are p time steps apart because the messages are received in FIFO order. The total number of time steps is therefore $((n-1)p+1)p + (p-1) = \Omega(np^2)$.

4.2. Randomized scattering. Formally, we establish the following theorem.

THEOREM 4.1. *Suppose that each node sends m messages and that for each message all destinations are equally likely and independent of choices of all other messages. The probability that the time until all messages have been received exceeds km is bounded by $O(e^{-m})$ for sufficiently large k and $m > \log p$.*

Proof. We adapt Ranade's proof [21] of the result of Karp and Zhang [13].

Let T be the completion time of the protocol, the last time step at which a message is received. Let message $\mathcal{M}_m$ be a message received at time step T and let S be the node which was the source of message $\mathcal{M}_m$. Let $\mathcal{M}_i$ denote the ith message sent by node S and let T_i denote the time step at which $\mathcal{M}_i$ was received at its destination Q_i.

DEFINITION. *Suppose that message $\mathcal{M}$ is selected for transmission, i.e., $\mathcal{M}$ enters the send buffer at time step τ and is destined for node q. Then we say that $\mathcal{M}$ became*

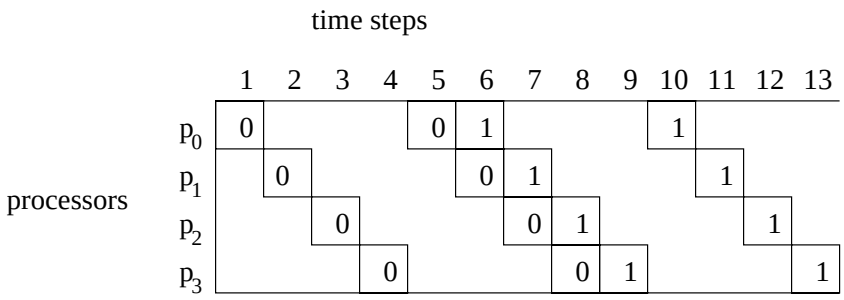


FIG. 2. *The situation when the messages are sent in ascending processor address order.*

ready for q at time step τ.

LEMMA 4.2. *There exists a partition $\Pi = \Pi_1, \ldots, \Pi_m$ of the interval $[1, T]$ and a set R of $T - m$ messages (not including those sent by S) each of which satisfies the following property: if the message became ready during Π_i its destination node is Q_i.*

Proof. Message $\mathcal{M}_i$ is received at Q_i at time step T_i. Let $T_i^{nr} < T_i$ be the maximum time step at which Q_i does not receive a message. At each time step of the interval $\Delta_i = [T_i^{nr} + 1, T_i]$, Q_i receives a message. Each of these messages became ready during the same interval Δ_i.

Observe that message $\mathcal{M}_{i-1}$ was received at time step T_{i-1} and message $\mathcal{M}_i$ became ready at time step $T_{i-1} + 1$. Therefore, $T_i^{nr} \leq T_{i-1}$. As a result there is no gap between any pair of consecutive intervals Δ_i, Δ_{i+1}. Given the intervals $\Delta_1, \ldots, \Delta_m$, we construct a partition Π as follows:

$$\begin{aligned} \Pi_m &= \Delta_m \\ \Pi_i &= \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \leq i < m. \end{aligned}$$

By construction, it follows that every message received by Q_i during Π_i became ready during Π_i and at least $T - m$ messages received by Q_i's during Π_i's were not sent by S. This establishes the lemma. ∎

To complete the proof of the theorem, we sum, over all possible partitions, choice of source S, and choice of $T - m$ messages, the probability that these $T - m$ messages chose their destinations in accordance with the partition.

The probability that a message which becomes ready during Π_i chooses Q_i as its destination equals $\frac{1}{p}$. The probability that each of $T - m$ messages makes the right choice is $p^{-(T-m)}$. The number of choices for S, the partitions, and the $T - m$ messages equals $p\binom{T+m}{m}\binom{(p-1)m}{T-m}$. The probability that $T \geq km$ is at most

$$\begin{aligned} & p\binom{T+m}{m}\binom{(p-1)m}{T-m}p^{-(T-m)} \\ & \leq 2^{T+2m}\binom{(p-1)m}{(k-1)m}p^{-((k-1)m)} \\ & \leq 2^{(k+2)m}\left(\frac{e}{k-1}\right)^{(k-1)m} \\ & \leq \left(2^{k+2}\left(\frac{e}{k-1}\right)^{k-1}\right)^{m}. \end{aligned}$$

The first two inequalities follow from the assumption that $m > \log p$ and the fact that $\binom{x}{y} < (xe/y)^y$ (the proof can be found in [16, p. 165]). For k sufficiently large, this quantity is smaller than $O(e^{-m})$. □

5. Techniques for tree searches.

5.1. Algorithmic issues. This section outlines the algorithmic and proof strategies for backtrack and branch-and-bound searches in the atomic message model. The branch-and-bound strategy is essentially that of Karp and Zhang [13]; their model allows any number of messages to be received at a node in one time step. Our technical contribution is to extend their result to the atomic transmission model. The proofs of both results extend the techniques of the previous section.

While the goal of both search procedures is to find the minimum cost leaf, there is an essential difference. Backtrack search examines *every* vertex of the search tree. In branch-and-bound search the cost associated with each vertex increases monotonically with the distance from the root, so that only the *critical subtree*, consisting of vertices with cost no greater than the minimum cost leaf, need be examined. We call such vertices *critical vertices*. For efficient branch-and-bound search, the time devoted to examining noncritical vertices must not dominate that for examining the critical subtree.

Within each synchronous reactive cycle, each processor (1) receives a tree vertex, if any, from its receive buffer, (2) examines and expands the vertex, and (3) puts the children onto the message queue, headed for an independently chosen random destination. For backtrack search we place no requirements on the message queue discipline. However, for branch-and-bound search we require that the message queue be a priority queue, so that the tree vertex selected for transmission is the one with minimum cost.

Using priority message queues for branch-and-bound search means that noncritical vertices cannot be selected for transmission when there is at least one critical vertex inside the message queue. However, a critical vertex can arrive inside the message queue while a noncritical vertex occupies the send buffer. In this case, the critical vertex will have to wait for selection, but it is easy to see that a critical vertex can be delayed by a noncritical vertex in this manner at most once.

Once a message has been selected for transmission, it is still subject to receive-delays. Receive-delays depend on the network policy and are beyond the control of the programmer, so we would like to make as few assumptions as necessary. For backtrack search we are able to carry out the analysis without making any assumptions on network arbitration. For branch-and-bound search, however, our analysis requires that the network observes a FIFO arbitration policy.

In conclusion, our analysis for branch-and-bound search makes stronger assumptions on both the message queue discipline and the network arbitration policy. The first assumption is required to guarantee that progress is made on the critical subtree and is reasonable from an algorithmic viewpoint. The second assumption, concerning network arbitration, is required for technical reasons: we bound the running time as a function of the size of the critical subtree, not the entire search tree which can be arbitrarily larger. Currently we do not know if the FIFO assumption can be weakened, and it is conceivable that it can.

5.2. Proof techniques. In this section we describe some of the ideas and terminology common to the analysis for both backtrack and branch-and-bound searches.

In both problems our goal is to analyze the time to expand a critical tree[4] of size n and depth h on a p-node system. For branch-and-bound search the quantities n and h can each be much smaller than the size and depth of the complete search tree.

In the analysis of the running time we proceed as follows. At time $t = 1$ the root is assumed to be in the send buffer of some node and is received at its destination within that time step. Suppose that the running time is T, i.e., the last time step at which a critical vertex is received. Pick one of the critical vertices received at time T—it must be a leaf in the critical subtree. Call the path $s_1, s_2, \ldots, s_h$ from the root (s_1) to this critical leaf (s_h) the *special path* and the vertices along this path the *special vertices*. Let Q_i denote the destination queue of special vertex s_i.

The first step of the proof is similar to the proof of Lemma 4.2. For a fixed run of the algorithm we construct a partition, $\Pi = \{\Pi_1, \ldots, \Pi_h\}$, of the time interval $[1, T]$. Next, we construct a *signature set* R of nonspecial, critical vertices each of which became ready for some Q_i during the corresponding time interval Π_i. Roughly speaking, the signature set, R, is constructed such that the receive delay periods of its children are disjoint and the sum of these receive-delays is large, i.e., close to T.

There are two cases to consider: the signature set R is either large or small, compared with the threshold αT, where α is some suitably chosen constant. We first show that it is unlikely that R is large when T is large. The proof closely follows the proof of Theorem 4.1.

LEMMA 5.1. *For suitable constants k, α, the probability that $T > k(\frac{n}{p} + h)$ and $|R| \geq \alpha T$ is polynomially small in n.*

Proof. We estimate $Pr(|R| \geq \alpha T)$ by summing the probability of the event $|R| \geq \alpha T$ under all possible combinations of partition Π and queue sequence Q. Given Π and Q each critical vertex appears in R with probability $1/p$, independent of other vertices. The probability that $|R| \geq \alpha T$ and all the special nodes go to destinated Q_i is therefore bounded by $\binom{n}{\alpha T} p^{-(h+\alpha T)}$.

The number of choices for Π, Q and the special path S is no more than $\binom{T+h}{h} p^h n$. Thus, the probability that $|R| \geq \alpha T$ is bounded by

$$
\begin{aligned}
&\binom{n}{\alpha T} p^{-\alpha T} \binom{T+h}{h} n \\
&< \left(\frac{ne}{\alpha T p}\right)^{\alpha T} ((1 + T/h)e)^h n \\
&< \left(\frac{e}{\alpha k}\right)^{\alpha k(\frac{n}{p}+h)} \left(\left(k\left(\frac{n}{ph} + 1\right) + 1\right) e\right)^h n, \quad \text{when } T > k(\tfrac{n}{p} + h) \\
&< \left[\left(\frac{e}{\alpha k}\right)^{\alpha k(\frac{n}{ph}+1)} \left(\left(k\left(\frac{n}{ph} + 1\right) + 1\right) e\right)\right]^h n \\
&< \left[\left(\frac{e}{\alpha k}\right)^{\alpha k} ((2k + 1)e)\right]^h n
\end{aligned}
$$

which, for appropriately chosen k, is polynomially small in n, the size of the critical subtree since the height of the tree h is at least $\log n$. ∎

The second part of the proof argues that it is unlikely that R is small when T is large. The intuition is that the expected receive-delay of any vertex is a small constant; therefore, it would seem unlikely for the children of a small number of

[4]Every vertex in backtrack search is critical.

signature vertices to suffer a large total receive-delay. Unfortunately, the delays of the children of the signature vertices are not independent random variables, so that Chernoff bounds cannot immediately be invoked.

Briefly, in analyzing backtrack search we track the destinations of the children of the signature vertices to construct a new set of queues, a new partition of time, and a new signature set. The new signature set is guaranteed to be large; consequently, the remainder of the proof follows the proof of Lemma 5.1. The analysis of branch-and-bound search is based on the observation that, under FIFO arbitration, the delays of the children of the signature set can essentially be treated as a martingale, thereby allowing us to use Chernoff bounds.

6. Analysis of backtrack search. In this section we demonstrate that when each vertex chooses its destination randomly and independently, then with high probability the completion time of backtrack search is optimal within a constant factor.

Following the outline of the previous section, we proceed in two stages. In the first stage we identify the required signature set R; the second stage establishes the unlikelihood of the event that T is large while R is small.

6.1. Signature set. We begin with some terminology and definitions. As before, let $S = \{s_1, \ldots, s_h\}$ denote the special vertices along the special root-to-leaf path and Q_i be the destination queue which receives special vertex s_i at time T_i.

DEFINITION. *A node Q is* empty at time t *if neither the send buffer nor the message queue of Q contains a vertex* right after the send phase of time step t. *By definition a node cannot be empty at time t if it receives an internal vertex at time $t-1$. However, it is possible that a node which is empty at t receives a leaf at time $t-1$. Note that any node which is nonempty throughout an interval I attempts to inject a vertex into the network at every time step of I.*

DEFINITION. *Suppose that node Q receives a vertex at each time step during time interval W. We call the interval W an* arrival window *for node Q. Recall that a receive-delay is the time interval during which a message waits in the send buffer due to destination congestion. Note that if time interval W is the receive-delay of vertex v, then W is an arrival window for the destination queue of v.*

DEFINITION. *For $1 \le i < h$, let T_i^e denote the maximum time $t < T_{i+1}$ such that Q_i is empty at t, and T_i^{nr} denote the maximum time $t \le T_i^e$ at which Q_i does not receive a message. Finally, let N_i denote the interval $[T_i^e+1, T_{i+1}]$, $A_i = [T_i^{nr}+1, T_i^e]$ and $\Delta_i = A_i \cup N_i$.*

The following lemma summarizes three properties which are a straightforward consequence of the definitions above.

LEMMA 6.1.

1. *Q_i receives an internal vertex at time T_i^e,*
2. *Q_i is nonempty throughout N_i and Q_i attempts to inject a vertex at every step of N_i, and*
3. *A_i is an arrival window for Q_i.*

Let c_i denote the set of vertices that are injected into the network from Q_i during N_i. We obtain the following lemma.

LEMMA 6.2.

1. *The parent of every vertex in c_i becomes ready for Q_i during Δ_i,*
2. *Δ_i can be partitioned into a set of arrival windows (not necessarily for all Q_i), and*
3. *the interval Δ_i contains $[T_i, T_{i+1}]$, and $\bigcup_{i=1}^{h-1} \Delta_i = [1, T]$.*

Proof. Let w be the parent of a vertex $v \in c_i$. In contradiction to (1), suppose w is ready at or before T_i^{nr}. Two cases follow: either w is received before T_i^{nr} or after T_i^{nr}.

In the first case, if w is received before T_i^{nr}, then v will stay in the message queue of Q_i until it becomes ready. However, from the definition, v cannot become ready until $T_i^e + 1$ or later. This contradicts the fact that Q_i is empty at T_i^e.

In the second case, if w is received after T_i^{nr}, then the receive-delay of w is an arrival window for Q_i. This contradicts the fact that Q_i does not receive a vertex at T_i^{nr}. As a result w must be ready after T_i^{nr}.

From part 2 of Lemma 6.1 N_i can be partitioned into arrival windows for the destinations of c_i. Therefore, Δ_i can be partitioned into arrival window A_i for Q_i and a set of arrival windows in N_i for the destinations of c_i.

Finally, for part 3 we observe that $s_{i+1} \in c_i$, so s_i must be ready after T_i^{nr} (from part 1 of this lemma) and is received at $T_i > T_i^{nr}$. Therefore Δ_i contains $[T_i, T_{i+1}]$ and part 3 follows. □

From part 3 of Lemma 6.2 the union of all Δ_i covers the time interval $[1, T]$; consequently we can define a partition $\Pi = \{\Pi_1, \ldots, \Pi_{h-1}\}$ of the interval $[1, T]$ as follows:

$$\Pi_{h-1} = \Delta_{h-1},$$
$$\Pi_i = \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \le i < h-1.$$

DEFINITION. *Let R_i be the set of critical vertices which are not special ($v \notin S$) but are ready for Q_i during the interval Π_i. Also, let $R = \cup_{i<h} R_i$. We call the set R the* signature set.

Having identified the signature set, it remains to estimate the probability that the signature set is small when T is large.[5] This estimation is completed in the following section.

6.2. A refined partition. As mentioned in section 5, our strategy will be to find a new partition of the interval $[1, T]$ and a corresponding signature set which is guaranteed to be large.

In this section we identify $O(|R| + h)$ arrival windows which cover the interval $[1, T]$. We will find arrival windows to cover each Π_i $(1 \le i < h)$ and argue that the sum of the number of windows in each Π_i is $O(|R| + h)$. The next section will identify the new partition and signature set.

DEFINITION. *Let C_i denote those k_i children of $R_i \cup s_i$ that are ready within $N_i \cap \Pi_i$. We sort C_i into a list $v_{i,1}, \ldots, v_{i,k_i}$ according to the time they become ready. Also let $Q_{i,j}$ denote the destination of vertex $v_{i,j}$ and $W_{i,j}$ denote the receive-delay of $v_{i,j}$.*

Each Π_i is the union of two disjoint intervals $N_i \cap \Pi_i$ and $A_i \cap \Pi_i$. The interval $N_i \cap \Pi_i$ is an initial segment of N_i which, from part 2 of Lemma 6.2, can be partitioned into arrival windows. Each such arrival window W is the receive-delay of a vertex v which is enqueued after T_i^e and becomes ready in Q_i at the beginning of W. From part 1 of Lemma 6.2 the parent of v must be ready for Q_i after T_i^{nr}. In other words, the parent of v must be ready during Π_i and consequently $v \in C_i$. As a result the

[5] The case when R and T are both large is covered by Lemma 5.1.

interval $N_i \cap \Pi_i$ can be partitioned into the receive-delays of the k_i vertices in C_i.[6] Each such receive-delay is an arrival window $W_{i,j}$ for $Q_{i,j}$ $(1 \le j \le k_i)$.

From part 3 of Lemma 6.1 $A_i \cap \Pi_i$ is an arrival window for Q_i. Let $W_{i,0} = A_i \cap \Pi_i$ and $Q_{i,0} = Q_i$, so that the interval Π_i can be partitioned into arrival windows $W_{i,j}$ for $Q_{i,j}$ $(0 \le j \le k_i)$. Finally we consider Π_i for all i and it follows that $\bigcup_{1\le i<h} \bigcup_{0\le j\le k_i} W_{i,j} = [1, T]$.

From the discussion above, the receive-delays, $W_{i,j}$, of the k_i vertices $v_{i,j} \in C_i$, and $W_{i,0}$ cover Π_i. Moreover, the parent of every vertex in $C = \bigcup_{1\le i<h} C_i$ is in either R or S; consequently, the number of arrival windows is at most $h + |C| \le h + 2(|S| + |R|) = 3h + 2|R|$ since we assume a binary search tree.

We need one more definition to derive the refined partition Π^* and queue sequence Q^*, where we can find T vertices that become ready for Q^* according to Π^*.

DEFINITION. *For the arrival window $W_{i,j} = [t_1, t_2]$ for $Q_{i,j}$, let $t \le t_1$ be the maximum time step at which $Q_{i,j}$ does not receive a message. Define $W^*_{i,j} = [t+1, t_2]$ so that $(W^*_{i,j}, Q_{i,j})$ is the* maximal backward extension *of the arrival window.*

Let Q^* be the sequence of all the queues $Q_{i,j}$, $1 \le i < h$, $1 \le j \le k_i$. Notice that every message received by $Q_{i,j}$ during the interval $W^*_{i,j} = [t+1, t_2]$ necessarily becomes ready for $Q_{i,j}$ during the same interval. Otherwise $Q_{i,j}$ would have received a message at time t, a contradiction.

From the extended windows $W^*_{i,j}$, $1 \le i < h$, $0 \le j \le k_i$, we next obtain the partition Π^* as follows:

$$\Pi^*_{h-1,k_{h-1}} = W^*_{h-1,k_{h-1}},$$
$$\Pi^*_{i,j} = W^*_{i,j} - \bigcup_{l>i \vee (l=i \wedge m>j)} \Pi^*_{l,m}.$$

Now we can find T vertices that become ready for Q^* according to Π^* since every vertex received by Q^* must be ready in the corresponding Π^* interval and the union of Π^* is $[1, T]$. Let $X_{i,j}$ denote the set of vertices v such that $v \notin C \cup R \cup S$ and v is received by $Q_{i,j}$ during $\Pi^*_{i,j}$. From the discussion above every vertex in $X_{i,j}$ must become ready for $Q_{i,j}$ during $\Pi^*_{i,j}$. Finally, let $X = \cup X_{i,j}$ and $V = C \cup R \cup S$. Since the arrival windows cover the interval $[1, T]$, it follows that $|X| \ge T - |V|$.

6.3. Execution templates. Our goal in this section is to estimate the probability of the event that T is large and R is small. We proceed in two stages; first we characterize the completion time in terms of an *execution template*. Then we show that execution templates corresponding to large completion times are unlikely. This follows the delay-sequence arguments used in the literature [21, 22].

DEFINITION. *An execution template $\mathcal{E}$ is an octuple $(S, R, C, \Pi, \Pi^*, X, Q, Q^*)$ whose elements are defined as follows:*

- *$S = \{s_1, \ldots, s_h\}$ denotes the set of vertices along a path from the root to a leaf;*
- *R_i, $1 \le i < h$, are disjoint sets of nonspecial critical vertices that become ready for Q_i during Π_i, and $R = \cup_{i=1}^{h-1} R_i$ is the signature set;*
- *C_i, $1 \le i < h$, are disjoint sets of tree vertices that are children of $s_i \cup R_i$ and become ready within $N_i \cap \Pi_i$; $|C_i| = k_i$ and $C = \cup C_i$;*
- *$\Pi = \{\Pi_1, \ldots, \Pi_{h-1}\}$ is a partition of $[1, T]$;*

[6] The receive-delay of the last vertex v_{i,k_i} may not fall entirely within $N_i \cap \Pi_i$, in which case we truncate it accordingly.

- $\Pi^* = \{\Pi_{1,1}, \ldots, \Pi_{1,k_1}, \ldots, \Pi_{h-1,1}, \ldots, \Pi_{h-1\ k_{h-1}}\}$ *is a partition of* $[1, T]$*;*
- $X_{i,j}$*,* $1 \le i < h$*,* $0 \le j \le k_i$*, are sets of tree vertices that are disjoint from* $V = C \cup R \cup S$ *and ready for* $Q^*_{i,j}$ *during* $\Pi^*_{i,j}$*,* $X = \cup X_{i,j}$ *and* $|X| \ge T - |V|$*;*
- $Q = \{Q_1, \ldots, Q_h\}$ *is a set of queues such that for every* $1 \le i < h$*,* Q_i *is the destination queue of* s_i *and also of every vertex in* R_i *and* $X_{i,0}$*, and* Q_h *is the destination of* s_h*;*
- $Q^* = \{Q^*_{1,1}, \ldots, Q^*_{1,k_1}, \ldots, Q^*_{h-1,1}, \ldots, Q^*_{h-1,k_{h-1}}\}$ *is a set of queues such that for every* $1 \le i < h, 1 \le j \le k_i$*,* $Q^*_{i,j}$ *is the destination for the* j*th element in* C_i *and every vertex in* $X_{i,j}$*.*

From the earlier discussion, when the backtrack search takes T time steps to complete, there exists an execution template where the destination of vertices in S, R, C, and X satisfy the following conditions (let $D(v)$ be the random destination of vertex v):

1. $D(s_i) = Q_i$, $1 \le i \le h$.
2. For all $v \in R_i$, $D(v) = Q_i$, $1 \le i < h$.
3. Let $v_{i,j}$ be the jth element in C_i, $D(v_{i,j}) = Q^*_{i,j}$, $1 \le i < h, 1 \le j \le k_i$.
4. For all $v \in X_{i,j}$, $D(v) = Q^*_{i,j}$, $1 \le i < h, 1 \le j \le k_i$.
5. For all $v \in X_{i,0}$, $D(v) = Q_i$, $1 \le i < h$.

6.4. Estimating the probability of execution templates. Let $\mathcal{L}$ be the event $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$. We bound the probability of event $\mathcal{L}$ by summing the probabilities of event $\mathcal{L}$ under all possible execution templates. For a fixed execution template the probability that all vertices in S, R, C, and X choose the right queue according to $\mathcal{E}$ is at most

$$p^{-|S\cup R\cup C|} p^{-(T-|S\cup R\cup C|)} = p^{-h} p^{-|R|} p^{-|C-(S\cup R)|} p^{-(T-|S\cup R\cup C|)}.$$

Next, we count the number of different execution templates. The number of possible S is n since there are at most n leaves in the tree. After S and R are specified, there are at most $\binom{2(|S|+|R|)}{|C|}\binom{n}{T-|V|}$ ways to choose C and X. The destinations of vertices from S and R are specified by Q, so the number of unspecified queues in Q^* is $|C - (S \cup R)|$ and the number of ways to choose Q and Q^* is $p^h p^{|C-(S\cup R)|}$. Finally, there are $\binom{T+|C|+h}{|C|+h}\binom{T+h}{h}$ ways to choose Π^* and Π, so the total number of execution templates is at most

$$n\binom{n}{|R|}\binom{2(|S|+|R|)}{|C|}\binom{n}{T-|V|} p^h p^{|C-(S\cup R)|}\binom{T+|C|+h}{|C|+h}\binom{T+h}{h}.$$

LEMMA 6.3. *For suitable constants* $k > 1, 0 < \alpha < 1/3$*, the probability that* $T > k(\frac{n}{p} + h)$ *and* $|R| < \alpha T$ *is polynomially small in* n*.*

Proof. The probability of $\mathcal{L}$ is no more than the product of the number of different execution templates and the probability that every vertex in S, R, C, and X will actually choose the destination according to $\mathcal{E}$ when $T > k(\frac{n}{p} + h)$ and $|R| < \alpha T$.

$$
\begin{aligned}
& n\binom{n}{|R|}\binom{2(|S|+|R|)}{|C|}p^{-|R|}\binom{n}{T-|V|}p^{-(T-|V|)}\binom{T+|C|+h}{|C|+h}\binom{T+h}{h} \\
\le\ & 2^{\log n}2^{2(|S|+|R|)}\binom{n}{\lceil\frac{n-p}{p+1}\rceil}p^{-\lceil\frac{n-p}{p+1}\rceil}\left(\frac{ne}{(T-|V|)p}\right)^{(T-|V|)}2^{T+|C|+h}2^{T+h} \\
\le\ & 2^{(2\alpha+2)T+5h+|C|}\left(\frac{ne}{\lceil\frac{n-p}{p+1}\rceil p}\right)^{\lceil\frac{n-p}{p+1}\rceil}\left(\frac{ne}{(T-|V|)p}\right)^{T-|V|} \\
\le\ & 2^{(2\alpha+2)T+5h+2h+2\alpha T}\left(\frac{ne}{\frac{n}{2p}p}\right)^{\frac{n}{p}}\left(\frac{ne}{(T-3\alpha T-3h)p}\right)^{T-3\alpha T-3h} \quad \text{since } |V|\le 3(\alpha T+h) \\
\le\ & 2^{7h+(4\alpha+2)k(\frac{n}{p}+h)}(2e)^{\frac{n}{p}}\left(\frac{ne}{((1-3\alpha)k-3)(\frac{n}{p}+h)p}\right)^{((1-3\alpha)k-3)(\frac{n}{p}+h)} \\
\le\ & 2^{((4\alpha+2)k+7)(\frac{n}{p}+h)}(2e)^{\frac{n}{p}}\left(\frac{e}{((1-3\alpha)k-3)}\right)^{((1-3\alpha)k-3)(\frac{n}{p}+h)} \\
\le\ & \left[2^{(4\alpha+2)k+7}(2e)\left(\frac{e}{((1-3\alpha)k-3)}\right)^{((1-3\alpha)k-3)}\right]^{(\frac{n}{p}+h)} \\
\le\ & 2^{-(\frac{n}{p}+h)} \text{ for suitably chosen constants } k,\alpha.
\end{aligned}
$$

The first inequality follows from the observation that $\binom{n}{x}p^{-x}$ is maximized when $x = \lceil\frac{n-p}{p+1}\rceil$. The second inequality follows from the fact that $\log n < h$, $|S| = h$ and the assumption that $R \le \alpha T$. The probability in the fourth inequality is roughly $2^{k_1 T}k_2^{n/p}k_3^{k_4 T} < (2^{k_1}k_2k_3^{k_4})^T$. As long as we keep $(2^{k_1}k_2k_3^{k_4}) < 1$ (by choosing sufficiently large k) the probability will diminish when n (and T) is sufficiently large. Therefore, when n is sufficiently large, we replace T with $k(n/p+h)$. Finally, since $h \ge \log n$, the bound in the last step is polynomially small in n. □

From Lemmas 5.1 and 6.3 we have the following theorem.

THEOREM 6.4. *Let $\mathcal{T}$ be any binary backtrack search tree of size n and depth h. Let T be the total time for the random destination backtrack search algorithm to expand $\mathcal{T}$ in a p-node network. The probability that T exceeds $k(\frac{n}{p}+h)$, where k is suitably chosen, is polynomially small in n.*

7. Analysis of branch-and-bound search. The proof for backtrack search does not apply in the branch-and-bound search case because an adversarial network arbiter can delay a critical node by favoring noncritical nodes. In the backtrack case, every tree node has to be expanded. Therefore, no matter which tree node the arbiter chooses to be received, some progress is made. In the branch-and-bound case, although a critical node cannot be delayed by a noncritical node in the competition for the send buffer, it can be delayed by noncritical nodes in the competition for the same destination. An adversarial arbiter can work against the critical nodes so that they suffer long receive-delays.

Our analysis of branch-and-bound search is based on the assumption that the network obeys the FIFO scheduling policy. Under FIFO scheduling incoming vertices are received in time-stamp order; a vertex that is ready cannot be delayed by a vertex that becomes ready at a later time-step; vertices that become ready at the same time can be received in arbitrary order.

We prove linear speedup in two steps. First, we prove that the aggregate delay

of m nonoverlapping receive delays is bounded by $O(m)$ with high probability under FIFO scheduling. Next, we show that for every execution there exists a signature set R and a set of $O(|R|+h)$ nonoverlapping receive-delays with aggregate delay $\Omega(T)$. As a result, it is very unlikely for T to be large and $|R|$ to be small. The other case, that of large T and large $|R|$, is already covered by Lemma 5.1.

7.1. Martingales.

LEMMA 7.1. *Let $X_1, \ldots, X_m$ be m random variables each in the range $[0, p-1]$ and let $X = \sum_{i=1}^{m} X_i$. Suppose that the conditional expectation $E(X_i \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \leq 1$, for all $1 \leq i \leq m$, and $0 \leq x_1, \ldots, x_{i-1}, \leq p-1$. Then, $Pr(X \geq \alpha m) \leq (\frac{1}{2})^{\alpha \frac{m}{p-1}}$ when $\alpha \geq 2e$.*

Proof. The analysis is similar to the generalized Chernoff bound given by Leighton et al. [15]. We first estimate the expectation of e^{tX}:

$$\begin{aligned} E(e^{tX}) &= E(e^{tX_1} e^{tX_2} \cdots e^{tX_m}) \\ &= \sum_{x=0}^{p-1} e^{tx} E(e^{tX_2} \cdots e^{tX_m} | X_1 = x) Pr(X_1 = x). \end{aligned}$$

We then choose a value x_1^* for X_1 so that $E(e^{tX_2} \cdots e^{tX_m} | X_1)$ is maximized:

$$\begin{aligned} E(e^{tX}) &\leq \sum_{x=0}^{p-1} e^{tx} E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) Pr(X_1 = x) \\ &\leq \left(\sum_{x=0}^{p-1} e^{tx} Pr(X_1 = x) \right) E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) \\ &\leq E(e^{tX_1}) E(e^{tX_2} \cdots e^{tX_m} | X_1 = x_1^*) \\ &= E(e^{tX_1}) \sum_{x=0}^{p-1} e^{x} E(e^{tX_3} \cdots e^{X_m} | X_1 = x_1^*, X_2 = x) Pr(X_2 = x | X_1 = x_1^*) \\ &= E(e^{tX_1}) E(e^{tX_2} | X_1 = x_1^*) E(e^{tX_3} \cdots e^{tX_m} | X_1 = x_1^*, X_2 = x_2^*) \\ &\vdots \\ &\leq E(e^{tX_1}) E(e^{tX_2} | X_1 = x_1^*) \cdots E(e^{tX_m} | X_1 = x_1^*, \ldots, X_{m-1} = x_{m-1}^*). \end{aligned}$$

Each of these expectations is maximized when the probability is nonzero only at 0 and $p-1$, the endpoints of the range of X_i. From Markov's inequality we can bound $E(e^{tX_i} | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ by $Pr(X_i = 0 | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*) + Pr(X_i = p-1 | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*) e^{t(p-1)} \leq (1 - \frac{1}{p-1}) + \frac{1}{p-1} e^{t(p-1)}$ since $E(X_i | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is bounded by 1 from the assumptions. As a result we choose t so that $e^{t(p-1)}$ is larger than 1 and $E(e^{tX_i} | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is maximized when $Pr(X_i = p-1 | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*)$ is maximized.

$$\begin{aligned} E(e^{tX} | X_1 = x_1^*, \ldots, X_{i-1} = x_{i-1}^*) &\leq \left(\left(1 - \frac{1}{p-1} \right) + \frac{1}{p-1} e^{t(p-1)} \right)^m \\ &= \left(1 + \frac{e^{t(p-1)} - 1}{p-1} \right)^m \\ &\leq e^{(\frac{e^{t(p-1)}-1}{p-1})m}, \qquad (1 + y < e^y). \end{aligned}$$

Then we again use Markov's inequality to bound the probability that X is greater than αm:

$$\begin{aligned} Pr(X \geq \alpha m) &= Pr(e^{tX} \geq e^{t\alpha m}) \\ &\leq \frac{e^{(\frac{e^{t(p-1)}-1}{p-1})m}}{e^{t\alpha m}} \\ &= e^{-\frac{(\alpha \ln \alpha - \alpha + 1)m}{p-1}} \qquad \text{(when } t = \tfrac{\ln \alpha}{p-1}\text{)} \\ &\leq 2^{-\frac{\alpha m}{p-1}} \qquad \text{(when } \alpha \geq 2e\text{).} \qquad \square \end{aligned}$$

LEMMA 7.2. *Let $V = \{v_1, \ldots, v_m\}$ be m vertices with nonoverlapping delays. The probability that their aggregate delay exceeds βm is smaller than $(\frac{1}{2})^{\frac{(\beta-1)m}{p-1}}$ when $\beta \geq 2e + 1$.*

Proof. Let Q_i be the destination of v_i and X_i be the number of vertices that will be received by Q_i before v_i when v_i becomes ready. The receive-delay of v_i is $X_i + 1$ and the aggregate receive-delay of V is $m + \sum_{i=1}^{m} X_i$.

Every vertex chooses its destination independently and uniformly; therefore, given $X_1, \ldots, X_{i-1}$, v_i is equally likely to pick any destination. We will argue that, given $X_1, \ldots, X_{i-1}$, the expected value of X_i is no more than 1. When v_i makes its random choice there are at most $p - 1$ other ready vertices in the system whose choices are independent of v_i's choice. Therefore, the conditional expectation of X_i is less than one.

For the aggregate delay to exceed βm, the sum of all X_i must exceed $(\beta - 1)m$. The bound on the probability of this event follows from Lemma 7.1. $\square$

7.2. The signature set. As before we consider the special vertices $S = \{s_1, \ldots, s_h\}$. Let s_i be received by Q_i at time T_i for $1 \leq i \leq h$. For every $1 \leq i < h$ we seek a set of receive-delays which together cover the interval $[T_i, T_{i+1}]$.

Let T_i^{ns} be the largest time step smaller than T_{i+1} at which the send buffer of Q_i is not occupied by a critical vertex, $(1 \leq i < h)$. Note that at each time step during the interval $\Gamma_i = [T_i^{ns} + 1, T_{i+1}]$ the send buffer of Q_i is occupied by a critical vertex. Let c_i be the critical vertices that are injected into the network from Q_i during Γ_i. As a result, Γ_i can be partitioned into receive-delays of vertices in c_i.

Among all the parents of vertices in c_i let f_i be the one that becomes ready at the earliest time step, say, T_i^f. Since $s_{i+1} \in c_i$ it follows that s_i is received no earlier than T_i^f.

It is possible for a gap to exist between the receive-delays of f_i and vertices in c_i. In this case, the send buffer of Q_i must be occupied by a noncritical vertex, call it g_i, which is received at its destination at time T_i^{ns}. Observe that g_i cannot be received at its destination any earlier, for otherwise the send buffer of Q_i would have to be occupied by a critical vertex (the message queue contains at least one critical vertex, the child of f_i, and a critical vertex gets priority over noncritical vertices to enter the send buffer). However, this contradicts the definition of T_i^{ns}.

Let T_i^g be the time step at which the parent of g_i becomes ready and let $\Delta_i = [\min(T_i^f, T_i^g), T_{i+1}]$. The following lemma summarizes our observations.

LEMMA 7.3.

1. *The parent of each vertex in c_i becomes ready for Q_i during Δ_i,*
2. *Δ_i is the union of receive-delays of vertices f_i, c_i, and g_i (if it exists), and*
3. $\bigcup_{i=1}^{h-1} \Delta_i = [1, T]$.

Proof. The parent of each vertex v in c_i must become ready before v; furthermore, it cannot become ready before f_i. For part 2, if there is a gap between the receive-delay of f_i and c_i, then this gap will be covered by the receive-delay of g_i from the discussion above. Finally s_i must be ready at or after T_i^f from the definition of f_i so it cannot be received before T_i^f; thus the interval Δ_i contains $[T_i, T_{i+1}]$, and part 3 follows. □

From part 3 of Lemma 7.3 the union of all Δ_i cover the interval $[1, T]$. We can therefore define a partition Π of $[1, T]$ as follows:

$$\Pi_{h-1} = \Delta_{h-1},$$
$$\Pi_i = \Delta_i - \bigcup_{j>i} \Pi_j, \quad 1 \le i < h-1.$$

DEFINITION. *As before, a critical vertex v is in the* signature set *R if v is not a special vertex ($v \notin S$) and v becomes ready for Q_i during Π_i.*

There are three kinds of receive-delays in Π_i: the earliest ready parent f_i, the noncritical vertex g_i, and those vertices in c_i that become ready during $\Gamma_i \cap \Pi_i$. We use F, G, and C to denote the sets of these three kinds of vertices from all Π_i. From part 1 of Lemma 7.3 and the definition of R, the parent of every vertex in C is either in R or S. As a result the number of receive-delays in $F \cup G \cup C$ is at most $2h + 2(|S| + |R|) = 4h + 2|R|$.

The receive-delays identified thus far cover the interval $[1, T]$ and there are no more than $4h + 2|R|$ in number. They are not necessarily nonoverlapping, however. Using a straightforward greedy procedure it is possible to produce a subset of no more than $2h + |R|$ intervals which are disjoint and whose union includes at least $T/2$ time steps. With this observation, we have the following theorem.

THEOREM 7.4. *Let $\mathcal{T}$ be the critical branch-and-bound subtree of size n and depth h. Let T be the total time for the random destination algorithm to expand $\mathcal{T}$ in a p-node network under FIFO scheduling strategy. The probability that T exceeds $k(\frac{n}{p} + h)$ is polynomially small, for suitably chosen k, when $n > p^2 \log p$.*

Proof. The probability that the signature set R exceeds αT in size is polynomially small by Lemma 5.1. From the above discussion we can identify a set of $2h + \alpha T$ nonoverlapping receive-delays whose aggregate delay is at least $T/2$. From the result of Lemma 7.2 this probability is bounded by $(\frac{1}{2})^{\frac{cT}{p}}$ for a suitable constant c. This quantity is polynomially small in n for $n > p^2 \log p$ and a suitably chosen k. □

8. Conclusions. In this paper we have developed a simple model which captures some aspects of message-passing systems. The model can be extended in several ways to include, for example, nonuniformity of routing times and more system buffering capacity.

We believe that the model is simple enough to carry out further algorithmic analysis which we expect will shed light on the limitations of bounded resources in parallel systems.

Acknowledgments. We thank Marina Chen, Charles Leiserson, Abhiram Ranade, Chuck Seitz, and Peter Winkler for helpful discussions. We are grateful to Lennart Johnsson and Thinking Machines Corporation for their support.

REFERENCES

[1] M. Adler, P. Gibbons, Y. Matias, and V. Ramachandran, *Modeling parallel bandwidth: Local versus global restrictions*, Algorithmica, 24 (1999), pp. 381–404.

[2] C. Berge, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[3] S. Bhatt, M. Chen, C. Lin, and P. Liu, *Abstractions for parallel N-body simulation*, in Proceedings of the Scalable High Performance Computing Conference (SHPCC-92), Williamsburg, VA, 1992, pp. 38–45.

[4] G. Blelloch, P. Gibbons, Y. Matias, and M. Zagha, *Accounting for memory bank contention and delay in high-bandwidth multiprocessors*, IEEE Trans. Parallel Distrib. Systems, 8 (1997), pp. 943–958.

[5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Eicken, *Logp: Towards a realistic model of parallel computation*, in Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, 1993.

[6] C. Dwork, M. Herlihy, and O. Waarts, *Contention in sharded memory algorithms*, J. ACM, 44 (1997), pp. 779–805.

[7] H. Gabow, *Using Euler partitions to edge color bipartite multigraphs*, Internat. J. Comput. Information Sci., 5 (1976), pp. 345–355.

[8] M. Gereb-Graus and T. Tsantilas, *Efficient optical communication in parallel computers*, in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, 1992.

[9] P. Gibbons, Y. Matias, and V. Ramachandran, *Can a shared-memory model serve as a bridging model for parallel compuations?*, in Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures, Newport, RI, 1997.

[10] P. Gibbons, Y. Matias, and V. Ramachandran, *The queue-read queue-write asynchronous PRAM model*, Theoret. Comput. Sci., 196 (1998), pp. 3–29.

[11] P. B. Gibbons, Y. Matias, and V. Ramachandran, *The queue-read queue-write PRAM model: Accounting for contention in parallel algorithms*, SIAM J. Comput., 28 (1998), pp. 733–769.

[12] R. Karp, M. Luby, and F. M. auf der Heide, *Efficient PRAM simulation on a distributed memory machine*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 1992.

[13] R. Karp and Y. Zhang, *A randomized parallel branch-and-bound procedure*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988.

[14] R. Karp and Y. Zhang, *Randomized parallel algorithms for backtrack search and branch-and-bound computations*, J. ACM, 40 (1993), pp. 765–789.

[15] F. T. Leighton, M. J. Newman, A. Ranade, and E. Schwabe, *Dynamic tree embeddings in butterfly and hypercubes*, in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architecture, Santa Fe, NM, 1989.

[16] T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, San Mateo, CA, 1992.

[17] C. Leiserson, Z. Abuhamdeh, D. Douglas, C. Feynman, M. Ganmukhi, J. Hill, W. D. Hillis, B. Kuszmaul, M. S. Pierre, D. Wells, M. Wong, S. Yang, and R. Zak, *The network architecture of the connection machine CM-5*, in Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, 1992.

[18] P. Liu, W. Aiello, and S. Bhatt, *An atomic model for message passing*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architecture, Velen, Germany, 1993.

[19] P. Liu and S. Bhatt, *Experiences with parallel n-body simulation*, in Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architecture, Cape May, NJ, 1994.

[20] K. Mehlhorn and U. Vishkin, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339–374.

[21] A. Ranade, *A Simpler Analysis of the Karp-Zhang Parallel Branch-and-Bound Method*, Tech. Report UCB/CSD 90/586, University of California, Berkeley, CA, 1990.

[22] A. Ranade, *Optimal speedup for backtrack search on a butterfly network*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architecture, Hilton Head, SC, 1991.

[23] C. Seitz, *Multicomputers*, in Developments in Concurrency and Communication, C.A.R. Hoare, ed., Addison-Wesley, 1990, pp. 131–201.

[24] L. Valiant, *A bridging model for parallel computations*, Comm. ACM, 33 (1990), pp. 103–111.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 86–112

ON-LINE RANDOMIZED CALL CONTROL REVISITED*

STEFANO LEONARDI†, ALBERTO MARCHETTI-SPACCAMELA†,
ALESSIO PRESCIUTTI†, AND ADI ROSÉN‡

Abstract. We consider the problem of on-line call admission and routing on trees and meshes. Previous work gave randomized on-line algorithms for these problems and proved that they have optimal (up to constant factors) *competitive ratios*. However, these algorithms can obtain very low profit with high probability. We investigate the question of devising for these problems on-line competitive algorithms that also guarantee a "good" solution with "good" probability.

We give a new family of randomized algorithms with asymptotically optimal competitive ratios and "good" probability to get a profit close to the expectation. We complement these results by providing bounds on the probability of any optimally competitive randomized on-line algorithm for the problems we consider to get a profit close to the expectation. To the best of our knowledge, this is the first study of the relationship between the tail distribution and the competitive ratio of randomized on-line benefit algorithms.

Key words. on-line algorithms, competitive analysis, randomized algorithms, call admission control

AMS subject classifications. 68Q25, 68W20

PII. S0097539798346706

1. Introduction.

The area of communication networks gives rise to a large number of on-line problems. One of the most extensively studied problems in this area is the problem of on-line assignment of virtual circuits in networks: a sequence of requests for calls is given on-line to an algorithm which has to select a virtual circuit between the communicating parties (or reject the request), obeying the network constraints (such as link capacities). Two kinds of decisions are involved: *admission control*, i.e., the choice of whether to accept the call or not, and *route selection*, i.e., the decision on the route of an accepted call. Each call is processed before any future call is known, while the algorithm has to make both decisions. The basic form of this problem (when link capacities are 1, and each call requests bandwidth 1) is an on-line version of the well-known problem of maximizing the number of edge-disjoint paths.

The off-line version of the maximum edge-disjoint path problem is known to be NP-hard on general networks [13]. The problem is solvable in polynomial time on tree networks [11] while it is still NP-hard on mesh networks [17]. Kleinberg and Tardos [16] give an $O(1)$ approximation algorithm for meshes and densely embedded nearly Eulerian graphs. For general graphs an $O(\sqrt{m})$ approximation is given in [14], where m is the number of edges in the graph. Recently, an $\Omega(n^{1/2-\epsilon})$ inapproximability lower bound for the maximum edge-disjoint path problem on general graphs was proved in [12] (where n is the number of nodes in the graph).

*Received by the editors November 6, 1998; accepted for publication (in revised form) February 21, 2001; published electronically June 5, 2001. A preliminary version of this paper appeared in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1998, pp. 323–332.

http://www.siam.org/journals/sicomp/31-1/34670.html

†Dipartimento di Informatica Sistemistica, Università di Roma "La Sapienza," via Salaria 113, 00198-Roma, Italia (leon@dis.uniroma1.it, alberto@dis.uniroma1.it, presciutti@dis.uniroma1.it).
The work of these authors was partly supported by EU ESPRIT Long Term Research Project ALCOM-IT under contract 20244 and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative."
‡Computer Science Department, The Technion, Technion City, Haifa 32000, Israel (adiro@cs.technion.ac.il).

Algorithms for on-line maximization problems are usually analyzed in terms of their *competitive ratio* [20], i.e., the worst case, over all input sequences, of the ratio between the benefit obtainable by an off-line optimal algorithm (that knows the whole sequence in advance), and the benefit obtained by the on-line algorithm. It is easy to see that greedy strategies for the problems we consider here are noncompetitive; hence more sophisticated strategies are necessary. Awerbuch, Azar, and Plotkin [2] gave a deterministic on-line algorithm (denoted by AAP in this paper) for the case in which the bandwidth request of each call is small when compared with the link capacities of the network. Their algorithm works for any network topology and achieves a benefit that is at least a logarithmic fraction (in the size of the network) of the optimal solution. No deterministic algorithm can have an asymptotically better competitive ratio [2]. However, if calls may request the full bandwidth of links in the network, Awerbuch, Azar, and Plotkin [2] prove a lower bound of $\Omega(n)$ for deterministic on-line algorithms for a line network, where n is the number of vertices. (Deterministic algorithms with good competitive ratios are still possible for the maximum edge-disjoint paths problem on some particular networks like expander graphs [15].)

Therefore randomized on-line algorithms have been considered for these problems. A randomized on-line algorithm, $\mathcal{A}$, is said to be c-competitive (against an oblivious adversary [7]) if for *any* sequence of requests σ, $E[\mathcal{A}(\sigma)] \geq \frac{1}{c} \cdot OPT(\sigma)$, where $\mathcal{A}(\sigma)$ and $OPT(\sigma)$ denote the algorithm's and the optimal benefit over sequence σ, and the expectation is taken over the random choices of $\mathcal{A}$. Awerbuch et al. [3] proposed an algorithm with an $O(\log n)$-competitive ratio for trees of n vertices. This result has been improved to $O(\log D)$, where D is the diameter of the tree, by Awerbuch et al. [4]. Kleinberg and Tardos [16] presented an algorithm with an $O(\log n)$-competitive ratio for meshes and a class of planar graphs. For all these topologies, matching (up to constant factors) randomized lower bounds have also been proved. On the other hand, Bartal, Fiat, and Leonardi [5] proved an $\Omega(n^{\epsilon})$ lower bound for randomized on-line algorithms on a specific network.

However, all the aforementioned algorithms have only the property that their *competitive ratio* is "good." They have not been analyzed with respect to their probability to get a "good" solution, and they only guarantee that for any sequence of requests the *expected* benefit is "close" (usually a logarithmic fraction) to the optimum. However, it may happen that, with high probability, a very poor benefit is achieved (and the main contribution to the average is given by a high benefit, obtained with low probability).[1] Indeed, algorithms of this kind have been criticized on this basis as not giving an appropriate solution for the problems they attempt to solve. For instance, the $O(\log n)$-competitive algorithm of [3] for the maximum edge-disjoint path problem on trees obtains for some sequences σ, even with $OPT(\sigma) = \Omega(n)$, a benefit of 0 with probability $1 - \frac{1}{\log n}$. The $O(\log D)$-competitive algorithm for trees [4] behaves somewhat better; still *unless* $OPT(\sigma) = \Omega(D^{\epsilon})$, the probability of getting any constant fraction of the expectation remains $o(1)$ (as a function of D). As to meshes, the algorithm of [16] decides with probability 1/2 to accept only "short" calls and with probability 1/2 to accept only "long" calls. On sequences of calls that are composed of only one kind of call, the algorithm would obtain benefit 0 with probability at least 1/2.

[1] Note that using the Markov inequality it is possible to obtain bounds on the probability of deviating from the expectation for algorithms for on-line minimization (cost) problems as a direct consequence of the analysis of the competitive ratio. This is not the case for maximization (benefit) problems.

In this paper we investigate the question of whether it is possible to obtain competitive randomized on-line algorithms (for the problems at hand) which also guarantee a "good" solution with "good" probability. Obviously, the probability of achieving a benefit close to the expectation is related to the quality of the expectation: the expectation of deterministic algorithms is obtained with probability 1; however, these algorithms have very poor competitive ratios for the problems we consider.

We formulate the following questions. Our results give answers to most of them in the context of the on-line maximum edge-disjoint paths problem.

1. Can a randomized on-line algorithm with optimal (up to a constant factor) competitive ratio guarantee a benefit of a constant fraction of the expectation with constant probability?

2. Can a randomized on-line algorithm with optimal competitive ratio (up to a constant factor) achieve a constant fraction of the expected benefit with probability that tends to 1 (say, as the size of the optimal solution grows)? What is the possible rate of convergence?

3. What is the interplay between the competitive ratio of a randomized algorithm, and the concentration of the benefit around its expectation?

1.1. Results of the paper. We address the above questions for the call admission problem in the basic case of infinite duration, requested bandwidth and link capacities 1, on the topologies of trees and meshes. Pairs of nodes are given on-line to an algorithm, which has to accept or reject every call immediately when it is received. An accepted call has to be assigned a route which is edge-disjoint with respect to all previously assigned routes. The aim is that of maximizing the number of accepted calls. This problem is equivalent to the on-line version of the maximum edge-disjoint paths problem for the above topologies. We present new randomized algorithms for trees and meshes with asymptotically optimal competitive ratios *and* good concentration of the benefit around its expectation.

Our work contains the following results for trees:

1. We present a family of randomized algorithms, parameterized by $k \geq 12$, with competitive ratios of $2k\lceil \log 2D \rceil$, and with the property that the probability to get any fixed constant fraction of the expected benefit tends to a constant, as $OPT/\log D$ tends to infinity. In addition, these algorithms have the property that their probability to obtain a $1-\delta$ fraction of the expectation, $\delta \in (0,1]$, is at least some constant value $1-\mathcal{P}(k,\delta)$ when $\frac{OPT}{\log D}$ is a large enough constant. The value of $1-\mathcal{P}(k,\delta)$ is such that it can be made as close to 1 as desired at the expense of a larger k, i.e., a larger constant in the competitive ratio.

We also present an $O(\log D)$-competitive algorithm that for *any* sequence of calls guarantees some constant fraction of the expected benefit with probability $\Omega(1)$ (equivalently, this algorithm guarantees an $O(\log D)$ fraction of the *optimal* solution with probability $\Omega(1)$).

2. We show that no optimally competitive algorithm can guarantee a constant fraction of the expected benefit with "very high" probability, unless the optimal solution is "very high." (For a formal statement see section 3.)

3. We also show that other algorithms from the family of algorithms with a slightly worse competitive ratio of $O(\log^{1+\epsilon} D)$, for arbitrary $\epsilon > 0$, have the property to achieve any fixed constant fraction of the expectation with probability that tends

to $1 - 1/\Omega(\log^{\epsilon} D)$, as $\frac{OPT}{\log^{1+3\epsilon} D}$ tends to infinity.

4. Our family of algorithms for trees is based on a new and simple scheme. We are also able to derive an algorithm with the best known competitive ratio for the problem, $6\lceil \log 4D \rceil$, as opposed to the previously known ratio of $48 \log 2D$ [4]. Finally we point out that our algorithms, and in fact *any* algorithm for link capacities 1, can be made to apply to trees with arbitrary uniform capacities.

5. We also study the problem on meshes and we present an asymptotically optimal $O(\log n)$-competitive algorithm for meshes that obtains any constant fraction of the expected benefit with probability that tends to 1, as $\frac{OPT}{\log^4 n}$ tends to infinity. This algorithm is based on some of the ideas of [16] and [6] and on new ideas presented here.

2. The algorithms for trees. The design of our family of algorithms is based on a new approach. They are composed of two conceptual steps. In the first step we apply an on-line *deterministic filter* to the input sequence. Those calls that are *not* filtered out are called *candidate calls* and may be accepted. The set of the candidate calls has two important properties: first, its cardinality is a constant fraction of the cardinality of the largest (optimal) set of calls that can be accepted (by an off-line adversary). Second, although this set cannot be fully accepted (as some of the calls intersect), the intersections between the calls of this set exhibit "nice" properties. When a call passes the deterministic filter and becomes a candidate it is presented to an on-line *randomized selection procedure* that determines if the call is actually accepted. The randomized selection procedures that we use are very simple, and the various algorithms are distinguished by the randomized selection procedure used.

Before we describe the algorithms, we give some notations: we denote by $\sigma = (s_1, t_1), (s_2, t_2), \ldots$ the sequence of requests. We denote by $OPT(\sigma)$ and $\mathcal{A}(\sigma)$ the set of calls accepted by an optimal (off-line) algorithm and by an on-line algorithm $\mathcal{A}$, respectively, out of the sequence σ. We denote by $\mathcal{C}(\sigma)$ the set of candidate calls passed on by the deterministic filtering procedure to the randomized selection procedure. By D we indicate the diameter of the tree. We abuse notation and denote by $OPT(\sigma)$, $\mathcal{A}(\sigma)$, and $\mathcal{C}(\sigma)$ also the cardinality of the corresponding sets. We start now by describing the properties of the deterministic filter that we use.

THEOREM 2.1. *There exists a deterministic on-line filtering procedure for trees of diameter D such that for any sequence σ the following properties hold:*

- $\mathcal{C}(\sigma) \geq OPT(\sigma)/6$.
- *The number of pairs of calls in $\mathcal{C}(\sigma)$ that intersect is at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$.*

We prove this theorem in section 2.2. We also show in section 2.3 an alternative filtering procedure which makes use of the AAP algorithm but achieves somewhat worse performances.

The different algorithms of the family of algorithms are distinguished by the randomized selection procedure used. We use a single procedure parameterized by a parameter p ($p \leq \frac{1}{2\lceil \log 2D \rceil}$). Algorithm $\mathcal{A}_p$ presents the candidate calls selected by the deterministic filter, one by one as they are selected to the on-line randomized selection procedure $\mathbf{RS}_p$ defined below.

$\mathbf{RS}_p$. Any candidate call $(s, t) \in \mathcal{C}$ becomes a *considered* call with probability p and is rejected with probability $1 - p$. A considered call is accepted if it does not

intersect any previous *considered* call.[2]

2.1. Analysis of the algorithms.

THEOREM 2.2. *Algorithm $\mathcal{A}_p$ is a $\frac{6}{p(1-p\lceil \log 2D\rceil\ )}$-competitive randomized on-line algorithm for call admission on trees of diameter D. For any $\delta \in (0,1]$, and any sequence σ, $Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \leq \exp(-\frac{OPT(\sigma)p}{48}(\delta(1-p\lceil \log 2D\rceil\ ))^2) + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D\rceil}-1\right)}$.*

Before proving the theorem we explain its consequences and give some corollaries. The proof of the first corollary follows from simple calculations.

COROLLARY 2.3. *For any fixed constant $k \geq 12$, algorithm $\mathcal{A}_p$ with $p = \frac{6}{k\lceil \log 2D\rceil}$ is a $2k\lceil \log 2D\rceil$-competitive randomized algorithm for on-line call admission on trees of diameter D. Let $P(k,\delta,\sigma) = 1 - Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]]$. Then for fixed $k \geq 12$ and fixed $\delta > 0$, $P(k,\delta,\sigma) \to (1-\mathcal{P}(k,\delta))$, for $= \mathcal{P}(k,\delta) = \frac{1}{1+\frac{1}{2}\delta(\frac{k}{6}-1)}$, as $\frac{OPT(\sigma)}{\log D} \to \infty$. Furthermore, when $OPT(\sigma) \geq \beta_k \log D$, for some constant β_k, then $P(k,\delta,\sigma) \geq 1-c$ for some fixed constant $c < 1$.[3] Note that $1-\mathcal{P}(k,\delta)$ is a constant bounded away from 0, if $k \geq 12$, and that it can be made as close to 1 as desired, at the expense of a larger constant k, i.e., a larger constant in the competitive ratio.*

Based on the above corollary we also define a slightly different algorithm, $\mathcal{B}_p$, that guarantees for *any* sequence a constant fraction of the expected benefit with probability $\Omega(1)$. This algorithm, with probability 1/2 accepts the first call (and stops); with probability 1/2 it runs algorithm $\mathcal{A}_p$ on σ.[4] Although $\mathcal{B}_p$ may have benefit only 1 with probability 1/2, it has the desirable property that it achieves an $O(\log D)$ fraction of the *optimal* solution with constant probability: if $OPT(\sigma) = O(\log D)$, then $\mathcal{B}_p$ will have an $O(\log D)$ fraction of the optimal benefit with probability 1/2 (if it takes the first call). If $OPT = \Omega(\log D)$, then with probability 1/2, $\mathcal{B}_p$ runs $\mathcal{A}_p$, which, for such sequences, achieves an $O(\log D)$ fraction of the optimum with constant probability by Corollary 2.3.

COROLLARY 2.4. *There exists an $O(\log D)$-competitive algorithm for call admission on trees of diameter D such that for any sequence of calls the benefit obtained is within some constant fraction of the expectation with constant probability.*

We now give another corollary, relative to a family of algorithms that obtain any fixed constant fraction of the expectation with probability that tends asymptotically to 1 (as $OPT(\sigma)$ and D grow) at the expense of a competitive ratio slightly higher than the optimum.

COROLLARY 2.5. *Algorithm $\mathcal{A}_p$, with $p = \Theta(\frac{1}{\log^{1+\epsilon} D})$, $\epsilon > 0$, is an $O(\log^{1+\epsilon} D)$-competitive randomized algorithm for on-line call admission on trees of diameter D.*

[2]The more natural algorithm that immediately rejects a candidate call if it intersects a previous *accepted* call, and otherwise accepts the candidate with probability p, performs at least as well. We leave the description as above for simplicity of the proof. To see that, note that for any given sequence of random choices, one for each candidate, any candidate accepted by $\mathbf{RS}_p$ is also accepted by the more natural algorithm. Since the call is accepted by $\mathbf{RS}_p$ its corresponding random choice is "positive," and it does not intersect any previous considered call (i.e., it does not intersect any previous candidate with a "positive" random choice). In the more natural algorithm this call cannot intersect any previously accepted call (as no previous candidate that intersects it had a "positive" random choice), and its own random choice is "positive." Hence it is accepted.

[3]To see that, note that $p = \Theta(1/(2\log 2D))$, and thus, for $OPT(\sigma) \geq \beta_k \log D$, and large enough β_k, the absolute value of the expression in the exponent of the first summand is at least a constant, and the second summand is bounded from above by a constant.

[4]A more natural algorithm would continue to run $\mathcal{A}_p$ after taking the first call in the first case. However, in the worst case its behavior is no better than the above algorithm.

For any constant $\delta \in (0,1]$, $Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] < \exp(-\delta^2 \frac{OPT(\sigma)}{\Theta(\log^{1+3\epsilon} D)}) + \frac{1}{\Omega(\delta \log^{\epsilon} D)}$.

Proof of Theorem 2.2. We prove that the expected number of calls accepted by $\mathcal{A}_p$ on a sequence σ is at least $\mathcal{C}(\sigma) \cdot p(1 - p\lceil \log 2D \rceil)$, where $\mathcal{C}(\sigma)$ is the set of candidate calls picked from σ. By Theorem 2.1, $\mathcal{C}(\sigma) \geq OPT(\sigma)/6$, and the competitive ratio follows.

We number the *candidate* calls by the order that they arrive from 1 to $\ell = C(\sigma)$. If calls i and j intersect we denote this by $i \cap j$. Let c_i, $i \geq 1$, be an indicator random variable which is 1 if and only if the ith candidate is *considered*. Let $C = \sum_i c_i$. Let $y_{i,j}$, for $i < j$ and $i \cap j$, be an indicator random variable which is 1 if and only if both calls i and j are considered, and let $Y = \sum_{i<j, i \cap j} y_{i,j}$. Let r_j be an indicator random variable which is 1 if and only if there is an index $i < j$, $i \cap j$, such that $y_{i,j} = 1$. Let $R = \sum_j r_j$. Note that $R \leq Y$.

A candidate is accepted if it was considered, but none of its (previously presented) intersecting candidates was considered. Thus, if candidate j is considered but not accepted into the on-line solution, then there is a candidate $i < j$, $i \cap j$, that was considered. That is, $c_j = 1$, and $r_j = 1$. Therefore $\mathcal{A}_p(\sigma) = C - R \geq C - Y$, and we get $E[\mathcal{A}_p(\sigma)] \geq E[C] - E[Y]$.

Clearly $E[C] = \mathcal{C}(\sigma) \cdot p$ since each candidate is considered with probability p. For any pair $i < j$, $i \cap j$, $Pr[y_{i,j} = 1] = p^2$ since the event occurs if and only if call i and call j are considered. By Theorem 2.1 there are at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$ pairs $i < j$ such that $i \cap j$. Therefore $E[Y] \leq \mathcal{C}(\sigma)\lceil \log 2D \rceil \cdot p^2$. We get that $E[\mathcal{A}_p(\sigma)] \geq E[C] - E[Y] \geq \mathcal{C}(\sigma)(p - \lceil \log 2D \rceil \cdot p^2) = \mathcal{C}(\sigma) \cdot p(1 - p\lceil \log 2D \rceil)$.

We now turn to prove the second part of the claim. For $\delta \in (0,1]$, since $E[C] \geq \frac{E[Y]}{p\lceil \log 2D \rceil} \geq \frac{E[R]}{p\lceil \log 2D \rceil}$, we have

$$
\begin{aligned}
&Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \\
&= Pr[(C - R) < (1-\delta)E[C - R]] \\
&\leq Pr\left[C < E[C] - \frac{1}{2}\delta E[C - R]\right] \\
&\quad + Pr\left[R > E(R) + \frac{1}{2}\delta E[C - R]\right] \\
&\leq Pr\left[C < E[C] - \frac{1}{2}\delta(1 - p\lceil \log 2D \rceil)E[C]\right] \\
&\quad + Pr\left[R > E[R] + \frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D \rceil} - 1\right)E[R]\right] \\
&= Pr\left[C < \left(1 - \frac{1}{2}\delta(1 - p\lceil \log 2D \rceil)\right) \cdot E[C]\right] \\
&\quad + Pr\left[R > \left(1 + \frac{1}{2}\delta\left(\frac{1}{p\lceil \log 2D \rceil} - 1\right)\right) \cdot E[R]\right] .
\end{aligned}
$$

The variable $C = \sum_{j=1}^{l} c_j$ is the sum of $\ell = \mathcal{C}(\sigma)$ random variables $c_j \in \{0,1\}$, set by independent Bernoulli trials to 1 with equal probability p and to 0 with equal probability $1-p$. We use the following version of Chernoff's bound (cf. [19]) to bound the first summand: let $\mu = E[C] = p\mathcal{C}(\sigma)$ be the expected value of variable C; then

for every $\gamma \in (0,1]$,

$$Pr[C < (1-\gamma)\mu] < e^{(-\mu\gamma^2/2)}.$$

The variable R is the sum of dependent random variables, as for all pairs $i, j, i < j$, $y_{i,j}$ depends on the random choice for candidate i. We use the Markov inequality for the second summand.

Using $E[C] \geq \frac{OPT(\sigma)p}{6}$, we have

$$\begin{aligned}
&Pr[\mathcal{A}_p(\sigma) < (1-\delta)E[\mathcal{A}_p(\sigma)]] \\
&< \exp\left(-\frac{E[C]}{2}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) \\
&\quad + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)} \\
&= \exp\left(-\frac{\mathcal{C}(\sigma)p}{2}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)} \\
&\leq \exp\left(-\frac{OPT(\sigma)p}{12}\left(\frac{1}{2}\delta(1-p\lceil\log 2D\rceil)\right)^2\right) \\
&\quad + \frac{1}{1+\frac{1}{2}\delta\left(\frac{1}{p\lceil\log 2D\rceil}-1\right)}\,. \qquad \square
\end{aligned}$$

2.2. The deterministic filter. When call (s,t) is presented to the algorithm, it is first passed through the deterministic filter, where it is discarded or becomes a candidate. We describe in the following a deterministic filter appropriately designed for trees.

First we designate an arbitrary internal vertex as the root of the tree and denote it by r. Without loss of generality we can restrict our attention to calls between two leaves of the tree. This is shown by constructing a new instance of the problem where all the calls are between leaf vertices of the tree, such that any solution to the new instance can be transformed back to a solution of the original instance, with the same size. The tree network of the new instance is obtained by adding to the original tree, for every internal vertex v and every edge e adjacent to v, a new leaf vertex v_e connected to v. For a call (s,t) of sequence σ, let e_s, e_t be the first and the last edges of the path connecting s to t. A new sequence σ' is obtained by transforming every call (s,t) into a call (s_{e_s}, t_{e_t}). It is easy to show that any subset of calls of σ can be accepted as a solution in the original tree if and only if the corresponding subset of calls of σ' can be accepted as a solution in the new tree. We note that this construction does not change the diameter of the tree.

We denote by $lca(s,t)$ the *least common ancestor* of vertices s and t in the tree rooted at r and by $p(u)$ the parent of vertex u in that tree. Path $path(s,t)$ is the *set* of edges in the path connecting s to t. Finally, $\mathcal{T}$ is the set of all edges of the tree.

Calls will be discarded on the basis of two tests. To perform the first test we block two edges when a call becomes a candidate: for any candidate (s',t') the two edges on $path(s',t')$ adjacent to $lca(s',t')$ are blocked. The edges are said to be *blocked edges.*

For the purpose of the second test we associate a weight function $w(e)$ to each edge of the tree. The value $w(e)$ is determined as follows:

1. Initially, assign $w(e) = 1$ for all edges $e \in \mathcal{T}$.

2. Double $w(e)$ whenever edge e is included in a $path(s', t')$ for a call (s', t') that becomes a candidate.

On the basis of the value of the weight function we give the following definition.

DEFINITION. *For two vertices u and v, $path(u, v)$ is a* segment *at a certain time in the run of the algorithm if at this time, for some nonempty subset of the current candidates, denoted by $\mathcal{C}'$, $path(u, v)$ is the maximal path contained in $\bigcap_{(s,t)\in\mathcal{C}'} path(s, t)$ such that no edge of the path is included in a candidate call not in $\mathcal{C}'$. We denote a segment, and its set of edges, by $seg(u, v)$.*

Note that for two vertices u and v, $path(u, v)$ may at times be a segment but later cease to be a segment. Further note that the edges of a segment may increase their weight over time, but when they form a segment they have equal weight. We thus sometimes refer to the weight of the segment being the weight of each of its edges. In what follows we will count the number of segments (of weight $w > 2$) ever created during the run of the algorithm. We identify a segment by its two endpoints. Thus, the first time that $path(u, v)$ becomes a segment, it is counted as one. If the weights of its edges increase, and it continues to be a segment, we do not count it again.

As an example, the set of segments defined by a set of candidate calls is shown in Figure 2.1.

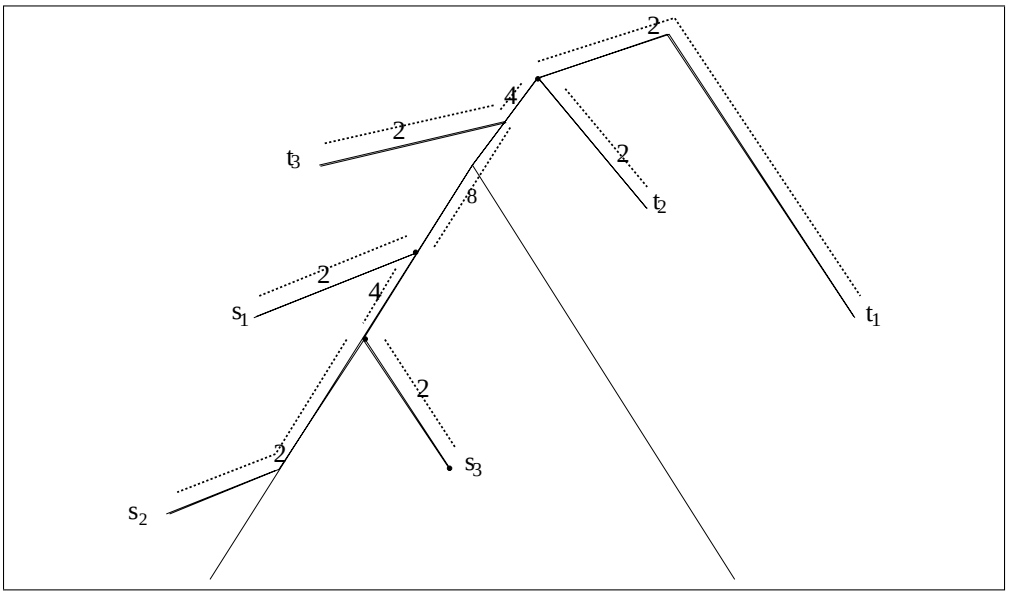


FIG. 2.1. *The segments created by the three candidate calls (s_1, t_1), (s_2, t_2), (s_3, t_3) are marked with dashed lines. The weight of each segment is indicated next to the dashed line.*

As mentioned above the filter is based on two tests:

Test 1. Discard call (s, t) if $path(s, t)$ contains a blocked edge.

If call (s, t) is not discarded by the first test, then it is submitted to the second test.

Test 2. Discard a call (s, t) if at the time the call is presented, there exists a segment $seg(u, v)$ of weight w such that $|path(lca(s, t), s) \cap seg(u, v)| \geq \frac{2D}{w}$ or $|path(lca(s, t), t) \cap seg(u, v)| \geq \frac{2D}{w}$.

A call that is not discarded by any of these test becomes a *candidate*.

DEFINITION. *For a sequence of calls σ, denote by $\mathcal{C}(\sigma)$ the set of candidates produced by the deterministic filter out of sequence σ. Denote by $\mathcal{C}_1(\sigma)$ those calls that are discarded by Test 1 and by $\mathcal{C}_2(\sigma)$ those calls that are discarded by Test 2. When the sequence σ is clear from the context, we sometimes use the notations $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2$. We also sometimes abuse notation and use $\mathcal{C}, \mathcal{C}_1, \mathcal{C}_2$ for the cardinality of the corresponding sets.*

We prove below that the number of candidates $\mathcal{C}(\sigma)$, i.e., those calls that are not discarded by any of the two tests, is at least $OPT(\sigma)/6$. In the following we will often say that edge e is included in a call (s,t) if $e \in path(s,t)$. We give some claims and lemmas as intermediate steps.

CLAIM 2.6. *A call (s,t) in $\mathcal{C} \cup \mathcal{C}_2$ intersects a previous candidate (s',t') only if (s',t') includes edge $(lca(s,t), p(lca(s,t)))$.*

Proof. If there is a previous candidate (s',t') that does not include edge $(lca(s,t), p(lca(s,t)))$ but intersects call (s,t), then $lca(s',t')$ is in the subtree rooted at $(lca(s,t))$. Then (s,t) intersects (s',t') in one of the two blocked edges of (s',t'), a contradiction to Test 1. □

CLAIM 2.7. *If the edges of a segment $seg(u,v)$ have weight $w > 2$, then either u is an ancestor of v or v is an ancestor of u.*

Proof. If $w > 2$, then the edges of the segment are included in more than one candidate call. Let (s,t) be the first such candidate call. If neither u is an ancestor of v nor v is an ancestor of u, then $lca(s,t)$ is internal to $seg(u,v)$. Two edges of $seg(u,v)$ would have been blocked when (s,t) became a candidate. The two blocked edges would also be part of any later call (s',t') that contains $seg(u,v)$, and (s',t') would not become a candidate, a contradiction. □

LEMMA 2.8. *Let (s,t) be a call in $\mathcal{C} \cup \mathcal{C}_2$ (i.e., a call that passes Test 1). Consider the intersection of $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$) with the set of calls that are candidate calls prior to the presentation of (s,t) and assume that this intersection is not empty. Then, there exists a sequence of nodes $v_0, \ldots, v_\ell$, $\ell \geq 1$, such that the following hold:*

1. *$v_0 = lca(s,t)$; v_ℓ is either equal to s (resp., t) or on $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$); and the intersection of $path(lca(s,t),s)$ (resp., $path(lca(s,t),t)$) with the set of previous candidate calls is $path(v_0, v_\ell)$.*
2. *For $0 \leq i \leq \ell - 1$, all edges between v_i and v_{i+1} have the same weight, which we denote W_i.*
3. *For $0 < i \leq \ell - 1$, $W_i < W_{i-1}$.*
4. *For $0 \leq i \leq \ell - 1$, there is a set of candidates $\mathcal{C}_i$ such that for any edge e in $path(v_i, v_{i+1})$, the set of calls that use e is exactly $\mathcal{C}_i$.*
5. *For $0 < i < \ell - 1$, $path(v_i, v_{i+1})$ is a segment.*

Proof. We prove the claim for the path from $lca(s,t)$ to s. An analogous proof holds for the path from $lca(s,t)$ to t.

Consider the time when call (s,t) is presented before it becomes a candidate. First consider an edge e on $path(lca(s,t),s)$. If there is a previous candidate call that uses this edge, then this candidate call also uses edge $(lca(s,t), p(lca(s,t))$ by Claim 2.6. It follows that for any edge e on $path(lca(s,t),s)$, if edge e is used by a previous candidate, then all edges between e and $lca(s,t)$ are used by this candidate. Let e be the edge on $path(lca(s,t),s)$ which is furthest away from $lca(s,t)$ and is included in a previous candidate call. Let the node adjacent to e and further away from $lca(s,t)$ be node v_ℓ, and let v_0 be $lca(s,t)$. This proves point 1.

Since for any edge e, any previous candidate that uses edge e also uses all edges

between edge e and $lca(s,t)$ (by Claim 2.6), it follows that going from $lca(s,t)$ to s, the weight of the edges is nonincreasing. If there is a decrease in the weight of the edges, we assign the point of decrease as one of the nodes v_i. We thus get a sequence of nodes v_i that satisfy points 2 and 3.

For $0 \leq i \leq \ell - 1$, now consider $path(v_i, v_{i+1})$. We claim that there is a set of candidates $\mathcal{C}_i$ such that all edges on $path(v_i, v_{i+1})$ are used by exactly the calls in $\mathcal{C}_i$. Consider two edges e and e' on this path, where e' is further away from $lca(s,t)$ than e. Any candidate that uses e' also uses e, following Claim 2.6. On the other hand, if there were a candidate that uses e, but not e', then the weights of e and e' would have been different. We thus establish point 4.

We further claim that for $0 < i < \ell - 1$, $path(v_i, v_{i+1})$ is the maximal path of edges that are used by exactly the calls in $\mathcal{C}_i$. For edges on $path(lca(s,t), s)$, any edge which is not on $path(v_i, v_{i+1})$ has weight not equal to W_i and thus cannot be used by exactly the calls in $\mathcal{C}_i$. Furthermore, observe that there is at least one call in $\mathcal{C}_i$ that uses also the edge leading from v_{i+1} towards s and must also reach $lca(s,t)$ (by Claim 2.6). Thus no edge outside of $path(lca(s,t), s)$ can possibly be used by the calls in $\mathcal{C}_i$. It follows from the definition of a segment that $path(v_i, v_{i+1})$ is a segment satisfying point 5. □

The number of calls that are discarded by Test 2, while accepted in the optimal solution, will be proved to be related to the number of segments of weight bigger than 2 ever created. We prove the following upper bound on the number of such segments.

LEMMA 2.9. *Consider a run of the algorithm on a sequence of calls σ. The number of segments ever created, and which at some point in time have weight bigger than 2, is at most $4\mathcal{C}(\sigma)$.*

Proof. Note that when a call is presented but does not become a candidate, there is no change in the segments in the tree. Therefore, we consider events when a new call becomes a candidate. To prove the lemma we give an upper bound on the number of segments that at such event either are created with weight bigger than 2 or reach a weight bigger than 2 (without changing their endpoints). We show that the number of such segments is at most 4 per such event.

Let (s,t) be a call that becomes a candidate. If it does not intersect any previous candidate, then no new segments of weight bigger than 2 are created. We therefore assume that call (s,t) intersects at least one previous candidate. We distinguish between two cases. The first one is when all the intersections of (s,t) with previous candidates are either on $path(lca(s,t), s)$ or on $path(lca(s,t), t)$. The second case is when there are such intersections on both $path(lca(s,t), s)$ and $path(lca(s,t), t)$.

We now consider the first case (without loss of generality assume that all intersections are on $path(lca(s,t), s)$). We use the sequence of nodes v_i, $0 \leq i \leq \ell$, guaranteed by Lemma 2.8. A new segment of weight bigger than 2 can be created either by the augmentation of the weight of edges of weight 2, or by the augmentation of the weight of edges of weight already bigger than 2, but while creating new endpoints for the segments (this may create more than one new segment).

If prior to (s,t) becoming a candidate there is a segment of weight bigger than 2 that includes $(lca(s,t), p(lca(s,t))$ and the edge e that leads from $lca(s,t)$ towards s, then two new segments of weight bigger than 2 are created with a new endpoint at $lca(s,t)$. One of these two segments is $path(v_0, v_1)$. See Figure 2.2.

Further distinguish the case where v_ℓ is inside a segment from the case where v_ℓ is not inside a segment.

If v_ℓ is inside a segment (see Figure 2.3), two new segments are created, both

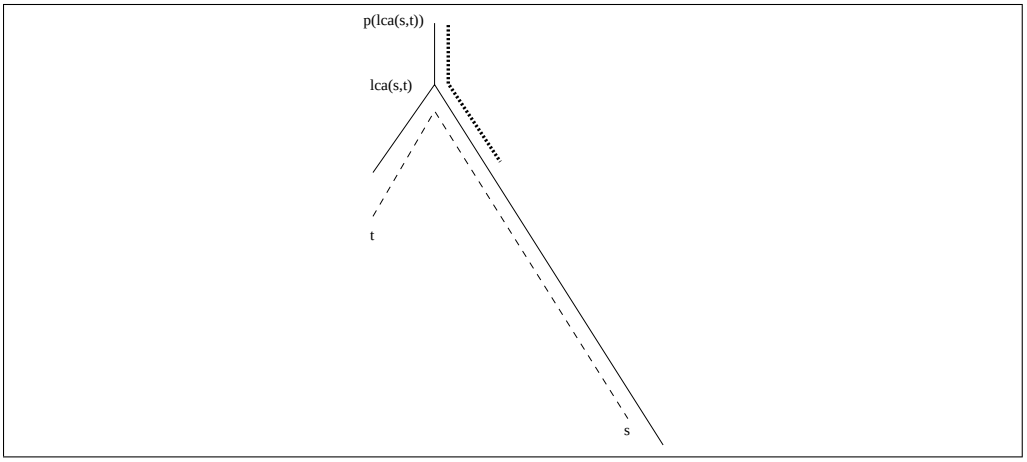


FIG. 2.2. *Two new segments are created with endpoint* $lca(s,t)$.

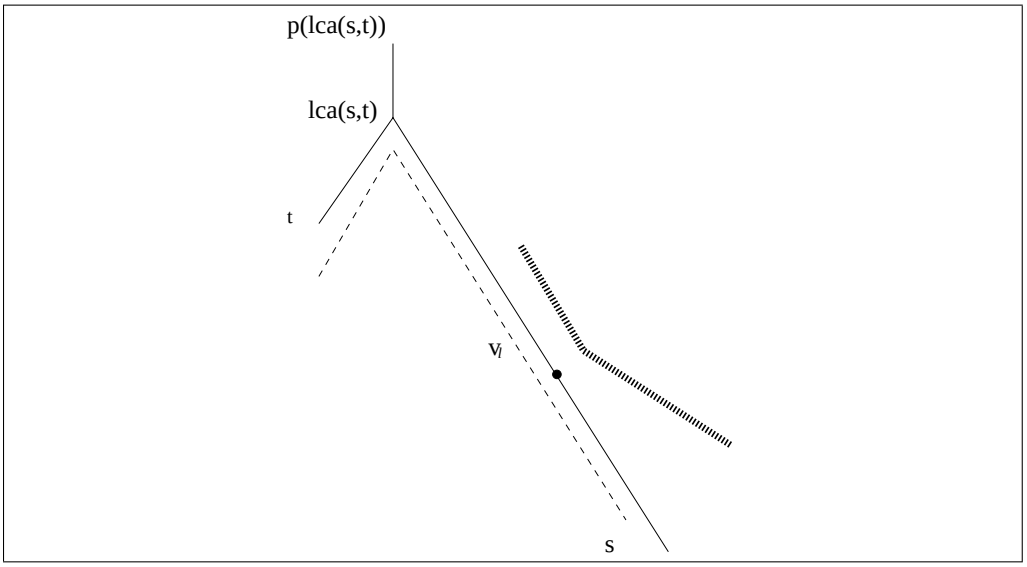


FIG. 2.3. *Vertex* v_ℓ *is inside a segment.*

with an endpoint at v_ℓ. If the weight of the segment was already bigger than 2, then two new segments of weight bigger than 2 are created. In this case all the segments $seg(v_{j-1}, v_j)$, $1 < j < \ell$, are already of weight bigger than 2; their weight will increase, but since their endpoints do not change there are no new segments here. If the weight of the segment was 2, then only one segment of weight bigger than 2 is created (while the other one is a segment of weight 2 with new endpoints). Whatever the weight of this segment was, all the segments $seg(v_{j-1}, v_j)$, $1 < j < \ell$, have weight bigger than 2 (by Lemma 2.8); their weight will increase, but since their endpoints do not change no new segments of weight bigger than 2 are created here. Altogether we get at most four new segments of weight bigger than 2.

For the case where v_ℓ is not inside a segment (see Figure 2.4), observe that the weights of the edges on $path(v_{\ell-1}, v_\ell)$ are doubled, thus possibly creating one new

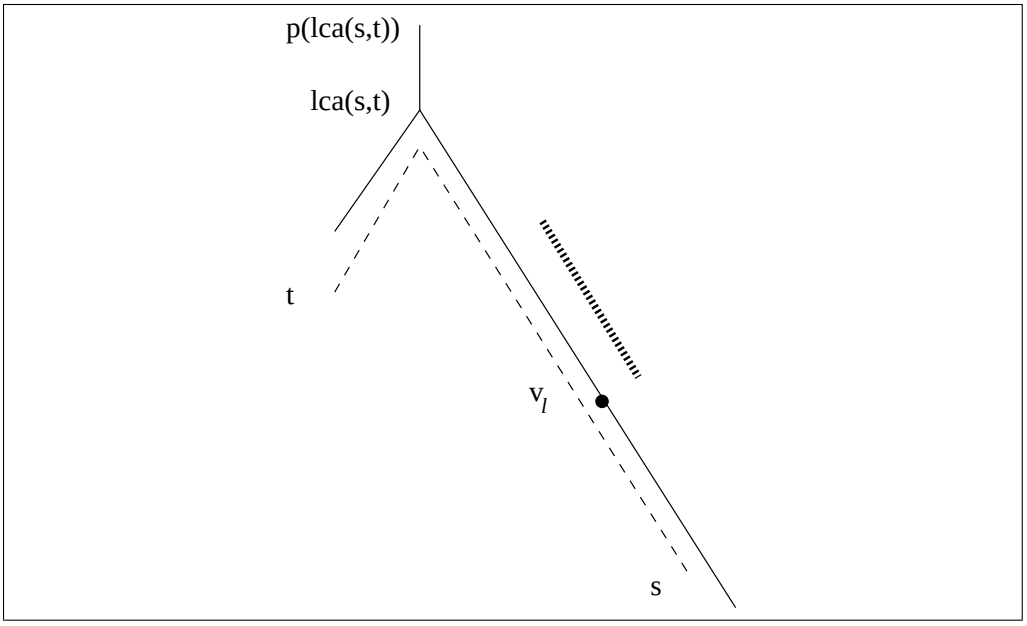


FIG. 2.4. *Vertex v_ℓ is not inside a segment.*

segment of weight bigger than 2 (which is $seg(v_{\ell-1}, v_\ell)$), if the weight of this segment was not bigger than 2 beforehand. Altogether we get for this case at most three new segments of weight bigger than 2.

We note that a new segment of weight 2 may be created as $seg(v_\ell, s)$ if $v_\ell \neq s$.

Now consider the second case where (s,t) intersects previous candidates on both $path(lca(s,t), s)$ and $path(lca(s,t), t)$. Consider the two edges e_1, and e_2, adjacent to $lca(s,t)$ and leading to s and t, respectively. Since by Claim 2.6 any previous candidate call that intersects (s,t) also uses the edge between $lca(s,t)$ and $p(lca(s,t))$, it follows that there are two candidate calls p_1 and p_2 such that p_1 uses e_1 and $(lca(s,t), p(lca(s,t))$, and p_2 uses e_2 and $(lca(s,t), p(lca(s,t))$. It follows that prior to (s,t) becoming a candidate, $(lca(s,t), p(lca(s,t))$ and e_1 (resp., e_2) cannot be in the same segment. We now use the sequence of nodes v_i guaranteed by Lemma 2.8 for each of $path(lca(s,t), s)$ and $path(lca(s,t), t)$. From the above argument it follows that the two paths $path(v_0, v_1)$ (on the two paths $path(lca(s,t), s)$ and $path(lca(s,t), t)$) are segments. No new segment will be created with a low endpoint at $lca(s,t)$ when (s,t) becomes a candidate. We therefore now consider separately $path(lca(s,t), s)$ and $path(lca(s,t), t)$. We consider in what follows $path(lca(s,t), s)$. Analogous arguments hold for $path(lca(s,t), t)$.

As argued above, prior to (s,t) becoming a candidate, $path(v_0, v_1)$ is a segment, $seg(v_0, v_1)$. When (s,t) becomes a candidate, the weight of this segment doubles.

If prior to (s,t) becoming a candidate the weight of $seg(v_0, v_1)$ was already bigger than 2, then there is no new segment at the top of $path(lca(s,t), s)$. Using the same arguments as in the case where all intersections are on $path(lca(s,t), s)$ (and not also on $path(lca(s,t), t)$) it follows that there are at most 1 or at most 2 new segments of weight bigger than 2 on $path(lca(s,t), s)$, depending on whether or not v_ℓ equals s.

If prior to (s,t) becoming a candidate the weight of $seg(v_0, v_1)$ was 2, then when the (s,t) becomes a candidate this segment becomes a new segment with weight bigger

than 2. However, it also follows that v_1 must equal v_ℓ, as there is only one previous candidate on $path(lca(s,t),s)$. No additional new segment of weight bigger than 2 will be created on $path(lca(s,t),s)$. (Note that a new segment of weight 2 may be created if $v_\ell \neq s$.)

We can conclude that on any of $path(lca(s,t),s)$ and $path(lca(s,t),t)$ at most two new segments of weight bigger than 2 can be created. Altogether at most four new segments of weight bigger than 2 are created when (s,t) becomes a candidate. □

We now prove that the number of calls not discarded (i.e., the number of calls that become candidates) is a good fraction of size of the optimal solution.

LEMMA 2.10. *For any sequence of calls σ, the number of candidate calls $\mathcal{C}(\sigma)$ has the property that $\mathcal{C}(\sigma) \geq \frac{OPT(\sigma)}{6}$.*

Proof. Since $\sigma = \mathcal{C} \cup \mathcal{C}_1 \cup \mathcal{C}_2$, we prove that the optimal solution accepts at most $2\mathcal{C}$ calls from $\mathcal{C} \cup \mathcal{C}_1$ and at most $4\mathcal{C}$ calls from $\mathcal{C}_2$.

First consider the calls from $\mathcal{C} \cup \mathcal{C}_1$ accepted in the optimal solution. Any call of $\mathcal{C} \cup \mathcal{C}_1$ includes at least one blocked edge. Since there are at most $2\mathcal{C}$ blocked edges in the tree, any solution can accept at most $2\mathcal{C}$ calls from $\mathcal{C} \cup \mathcal{C}_1$.

We now consider the calls from $\mathcal{C}_2$ accepted in the optimal solution. Consider a call $(s,t) \in \mathcal{C}_2$. It includes, when presented, at least $\frac{2D}{w}$ edges of some segment $seg(u',v')$ of weight $w > 2$ either on $path(lca(s,t),s)$ or on $path(lca(s,t),t)$. (Observe that no call is discarded because of a segment of weight 2 since $path(lca(s,t),s)$ and $path(lca(s,t),t))$ are of size at most $D-1$.) For the sake of analysis, we assign call (s,t) to a single such segment, $seg(u',v')$.

We now show that the length of $seg(u',v')$ is less than $\frac{4D}{w}$. By definition, the edges of segment $seg(u',v')$ are included in the intersection of a set $\mathcal{C}'$ of candidate calls. Let (s',t') be the call of $\mathcal{C}'$ presented last, and let $\mathcal{C}'' = \mathcal{C}' \setminus \{(s',t')\}$. Set $\mathcal{C}''$ defines a segment $seg(u'',v'')$ at the time call (s',t') is presented, and this segment includes segment $seg(u',v')$. To see that, observe that $path(u',v')$ is included in the intersection of the calls in $\mathcal{C}''$ and that at the above time no other candidate includes any edge of $path(u',v')$. We know that the weight of this segment at the time call (s',t') is presented is $\frac{w}{2}$ (since when (s',t') becomes a candidate it doubles the weights to w). Since call (s',t') becomes a candidate it passes Test 2, and we can conclude that its intersection with $seg(u'',v'')$, on either $path(lca(s',t'),s')$ or $path(lca(s',t'),t')$, is of length less than $\frac{4D}{w}$. However, by Claim 2.7, either u' is an ancestor of v' or vice versa, as $w > 2$, and $path(u',v')$ is included in call (s',t'), as $seg(u',v')$ is defined by $\mathcal{C}'$. Therefore all of $path(u',v')$ is included in this intersection. If follows that the length of segment $seg(u',v')$ is less than $\frac{4D}{w}$.

It now follows that if we consider the calls of the optimal solution, at most one call can be assigned to each segment of weight bigger than 2 in the way we described. This is because any assigned call uses more than half the edges of the segment it is assigned to, and thus any two would intersect. By Lemma 2.9, at most $4\mathcal{C}$ segments of weight bigger than 2 are ever created while the algorithm processes the sequence. We conclude that at most $4\mathcal{C}$ calls of $\mathcal{C}_2$ can be in the optimal solution. □

LEMMA 2.11. *Every candidate call intersects at most $\lceil \log 2D \rceil$ previous candidates.*

Proof. By Claim 2.6, for any candidate (s,t), all the previous intersecting candidates include edge $(lca(s,t), p(lca(s,t)))$. Their number is at most $\lceil \log 2D \rceil$, since once an edge is included in $\lceil \log 2D \rceil$ candidates, it has weight of at least $2D$. No call can become a candidate if it includes an edge of weight $2D$, as it will fail Test 2. □

Proof of Theorem 2.1. The first part of the theorem is proved in Lemma 2.10. For the second part, by Lemma 2.11 every candidate call of $\mathcal{C}(\sigma)$ intersects at most $\lceil \log 2D \rceil$ previous candidates. The number of intersecting pairs is thus at most $\mathcal{C}(\sigma) \cdot \lceil \log 2D \rceil$. □

2.3. An alternative deterministic filter. In this section we describe an alternative design of the deterministic filter at the cost of having a somewhat worse performance. This filter uses the AAP (deterministic) call control algorithm [2], originally designed for general networks of high capacity. The test runs the AAP algorithm on the tree, assuming link capacities of $\log 4D$. A call that is accepted by the AAP algorithm becomes a candidate, while a call that is rejected by AAP is also rejected by the filter. This filter relies on the following property: if the capacity is increased in that way, then the set of calls accepted by AAP is a constant fraction of the optimal set accepted by an adversary that has only capacity 1. (A similar result is also derived in [18] if the capacity is increased by a factor of $\Omega(\log n)$.) On the other hand, since the capacity of each link is bounded, the number of intersections between the candidate calls is small.

In Appendix A we give for completeness the AAP algorithm for this specific setting and prove the following claim.

THEOREM 2.12. *If for every $e \in E$ the AAP algorithm has capacity $u(e) = \log 4D$ while the adversary has capacity $u(e) = 1$, then the AAP algorithm has competitive ratio* 5.

Based on the above theorem we can now give the properties of the filter.

LEMMA 2.13. *The above filter based on the AAP algorithm, applied to trees of diameter D, achieves the following for any sequence σ:* (1) $\mathcal{C}(\sigma) \geq OPT(\sigma)/5$; (2) *The number of pairs of calls in $\mathcal{C}(\sigma)$ that intersect is at most $\mathcal{C}(\sigma) \cdot 2\lceil \log 2D \rceil$.*

Proof. The first part of the claim follows from Theorem 2.12. For the second part of the claim we consider an intersection graph of paths on a tree.[5] Our claim follows from the fact that if each edge in the tree is included in at most g of these paths, then the intersection graph is a $(2(g-1))$-inductive graph (see [6]).[6] Since the capacity used by the AAP algorithm is $\log 4D$, any edge is used by at most $\log 4D$ calls. Therefore the intersection graph of the candidate calls is a $2(\log 4D - 1)$-inductive graph, which implies that the total number of edges in this graph is at most $\mathcal{C}(\sigma) \cdot 2(\log 4D - 1) = \mathcal{C}(\sigma) \cdot 2 \log 2D$. This is a bound on the number of pairs of candidates that intersect. □

The randomized selection procedure used in conjunction with this filter is the same as for our first filter. The results stated in Theorem 2.2 are proved in a similar way with necessary modifications in the constants.

3. A bound on the probability to achieve a "good" solution. We have presented $O(\log D)$-competitive algorithms for call admission on trees that for any sequence σ, with $OPT(\sigma) = \Omega(\log D)$, achieve any constant fraction of the expected benefit with at least constant probability. The probability tends to some constant as $OPT(\sigma)$ tends to infinity. The obvious question is then if the rate of convergence and the limit probability can be improved for $O(\log D)$-competitive algorithms and in particular if the limit can be 1.

[5] The intersection graph has a node for each one of the paths. There is an edge between two nodes in the intersection graph if and only if the corresponding two paths intersect.

[6] A d-inductive graph is a graph $G = (V, E)$ for which there is an ordering of the vertices such that for any i, $|\{j : i < j, (i, j) \in E\}| \leq d$.

In this section we give a partial answer to this question. We show that for any algorithm with an $O(\log D)$-competitive ratio, there is a sequence σ with $OPT(\sigma) = \Theta(D^{1-c})$, for any $0 < c < 1/2$, for which with probability $\Omega(c)$, the on-line benefit is $o(OPT(\sigma)/\log D)$ (in fact the on-line benefit is only a $n^{-\epsilon}$ fraction of the expectation for some constant $\epsilon > 0$).

We give our lower bound on a line network of $n+1$ vertices, with $n = 2^s$, for some s. Observe that on such a line network we have $D = n$. Given a randomized $k \log n$-competitive algorithm for the line, we present a sequence of requests formed by $l = \lfloor \alpha \cdot \log n \rfloor + 1$ ($\alpha < 1/2$) phases, plus a final extra phase. At phase i, $0 \le i \le l$, 2^i pairwise disjoint calls of size $\frac{n}{2^i}$ are presented. These are calls $((j-1)\frac{n}{2^i}+1, j\frac{n}{2^i}+1)$, $1 \le j \le 2^i$. Thus, during phase l, 2^l calls of size $n/2^l$ are presented. We will prove that for any $k \log n$-competitive algorithm, at least one of these calls has the property that with probability at least $\frac{\alpha}{4k}$ all its edges are occupied by the time phase l ends. Then, we present $n/2^l$ disjoint calls of length 1 on these edges. The optimal solution is of size $n/2^l \ge n/2^{\alpha \log n} = n^{1-\alpha}$, while with probability at least $\frac{\alpha}{4k}$ the on-line algorithm can achieve a benefit of at most $2^l \le 2^{\alpha \log n} = n^\alpha$. We get that the algorithm cannot be close (by any constant fraction) to its expectation with probability $1 - o(1)$ (as a function of D), even when the size of the optimal solution is $\Omega(n^{1-c})$.

We now turn to prove our claim. Let p_m^i be the absolute probability that the mth call of phase i is accepted. We denote by P_m^l the sum of the absolute probabilities of acceptance of all the calls that include call m of level l, plus the probability that this call itself is accepted. We will show that for any $k \log n$-competitive randomized algorithm, $\sum_{m=1}^{2^l} P_m^l \ge 2^l \cdot \frac{\alpha}{4k}$. Therefore, there is at least one call m' such that $P_{m'}^l \ge \frac{\alpha}{4k}$, from which our claim follows.

We have

$$\begin{aligned}
\sum_{m=1}^{2^l} P_m^l &= \sum_{i=0}^{l} 2^{l-i} \left(\sum_{m=1}^{2^i} p_m^i \right) \\
&\ge \sum_{i=0}^{l} 2^l \frac{1}{2} \sum_{j=i}^{l} 2^{-j} \left(\sum_{m=1}^{2^i} p_m^i \right) \\
&= 2^{l-1} \sum_{j=0}^{l} 2^{-j} \sum_{i=0}^{j} \left(\sum_{m=1}^{2^i} p_m^i \right).
\end{aligned}$$

The first equation follows since the absolute probability p_m^i of accepting call m of phase i is summed up for the 2^{l-i} included calls of level l. Now, observe that $\sum_{i=0}^{j}(\sum_{m=1}^{2^i} p_m^i)$ is the algorithm's expected benefit after the calls of phase j have been presented.

We continue with $2^{-j}\sum_{i=0}^{j}(\sum_{m=1}^{2^i} p_m^i) \ge \frac{1}{k \log n}$, since the algorithm is $k \log n$-competitive, and the optimal solution can accept 2^j calls after phase j. We get

$$\sum_{m=1}^{2^l} P_m^l \ge 2^{l-1} \frac{l+1}{k \log n} > 2^l \cdot \frac{\alpha}{4k}$$

(for large enough n), which proves our claim. We can conclude the following theorem.

THEOREM 3.1. *For any $k \log n$-competitive algorithm for call admission on the line of n nodes, and any constant probability p ($p < \frac{1}{8k}$), there is a sequence σ with*

$OPT(\sigma) = n^{1-O(p)}$ *such that with probability at least* p*, the algorithm* does not *achieve any constant fraction of the expected benefit.*

4. Further extensions for trees. In this section we show that a better competitive ratio can be obtained at the expense of a larger deviation. We obtain a competitive ratio of $6\lceil \log 4D \rceil$ which improves upon the previously known bound of $48 \log 2D$ [4]. We also consider extensions of our algorithms to trees with edges of any uniform capacity and point out that *any* algorithm for call admission on trees can be converted to apply to uniform-capacity trees with almost the same competitive ratio.

First, we give an algorithm with a competitive ratio of $6\lceil \log 4D \rceil$. This is done at the expense of a larger deviation from the expectation. To obtain the improved competitive ratio we use our general framework, with the first version of the deterministic filter (section 2.2), but a different randomized selection procedure. We now use a variation of the "classify and randomly select" technique [3]: we use a set of $\lceil \log 4D \rceil$ colors and define an arbitrary order on them. We first choose uniformly at random one color among these colors, and we denote it A. When a new candidate call is presented to the randomized selection procedure, we assign to it the first color (according to the defined order) that was not assigned to any of the previous candidates that intersect the present one (i.e., we color the candidate calls with a proper coloring in their intersection graph). By Lemma 2.11, $\lceil \log 4D \rceil$ colors are sufficient, since each candidate intersects at most $\lceil \log 2D \rceil$ previous candidates. The algorithm will now accept those candidates that are assigned the color A. Since the above coloring procedure classifies the candidate calls into $\lceil \log 4D \rceil$ disjoint classes, each class fully acceptable, we obtain the following theorem.

THEOREM 4.1. *There exists a* $6\lceil \log 4D \rceil$*-competitive randomized algorithm for call admission on trees of diameter* D*.*

Proof. We partition $\mathcal{C}$ into a set of $\lceil \log 4D \rceil$ disjoint classes. Class $\mathcal{C}_i$, $i = 1, \ldots, \lceil \log 4D \rceil$, contains all the candidates which are assigned color i. All the candidates of a class can be accepted together in a solution since they are nonintersecting. The algorithm accepts a set of calls $\mathcal{A}$ that corresponds to the randomly selected class. The expected size of the on-line solution is $E(\mathcal{A}) \geq \frac{1}{\lceil \log 4D \rceil} \sum_{i=1} \mathcal{C}_i = \frac{1}{\lceil \log 4D \rceil} \mathcal{C} \geq \frac{OPT}{6\lceil \log 4D \rceil}$, since, by Lemma 2.10, $\mathcal{C} \geq \frac{OPT}{6}$. □

We also point out that our algorithms also apply to trees with any uniform capacity. In fact, *any* c-competitive algorithm for call admission on trees of capacity 1 can be converted into a $2\frac{e^{1/c}}{e^{1/c}-1} \leq (2(c+1))$-competitive algorithm for call admission on trees with arbitrary uniform capacity. Given any c-competitive algorithm for the capacity-1 case, we run a "first fit"-based technique [1] that uses k copies of the original algorithm on k copies of the tree but each with capacity 1. If the call is accepted by any of the algorithms we accept it into the actual k-capacity tree. Using [1, 9] we get for this case that we have a $\frac{e^{1/c}}{e^{1/c}-1}$ -competitive algorithm with respect to an adversary that uses k distinct, capacity-1 trees. Since this adversary is only a $1/2$ fraction away from the original adversary that has a single tree of uniform capacity k (cf. [4]), we obtain a $2\frac{e^{1/c}}{e^{1/c}-1} \leq (2(c+1))$-competitive algorithm for the problem.

5. Routing on meshes with high probability. In this section we propose a randomized algorithm for the on-line maximum edge-disjoint paths problem on meshes that has a logarithmic competitive ratio. The algorithm achieves the best possible competitive ratio up to a constant multiplicative factor *and* a benefit close to the expectation with high probability on any sequence of a large class of input instances.

Kleinberg and Tardos [16] presented the first $O(\log n)$-competitive algorithm (denoted KT in the following) for the on-line maximum edge-disjoint paths problem on meshes. They partition the input sequence into two classes: *short calls* and *long calls*. Define the distance between two vertices of the mesh as the number of edges on a shortest path connecting the two vertices. According to the partitioning of KT, every call with endpoints at distance bigger than a given value $d = \Theta(\log n)$ is a long call. KT decides with equal probability to accept only long calls or only short calls. KT has thus the drawback of obtaining benefit 0 with probability at least 1/2 on any input sequence containing only calls with endpoints at distance bigger than d (or only calls with endpoints at distance at most d).

We propose an algorithm with a logarithmic competitive ratio that achieves a constant fraction of its expected benefit with probability tending to 1 as the size of the optimal solution grows. Our algorithm for meshes is composed of a randomized stage followed by a deterministic stage. The first stage of the algorithm is a *randomized filter* that selects a subset $\mathcal{C}$ of *candidate calls* out of the input sequence σ. Calls of $\sigma \setminus \mathcal{C}$ are discarded to make space for the routing of candidate calls. The calls in $\mathcal{C}$ form a new input sequence for the second stage of the algorithm, which is a completely deterministic procedure. The subset of $\mathcal{C}$ accepted by the second stage is the set of calls finally accepted by the algorithm. Our algorithm uses a routing strategy different from that of KT. Some of its ideas are borrowed from the algorithm of Bartal and Leonardi [6] for on-line path coloring on meshes [6] .

Let us denote by $ON(\mathcal{T})$ and $OPT(\mathcal{T})$ the subsets, of some sequence $\mathcal{T}$, that are accepted (out of $\mathcal{T}$) by our on-line algorithm and by the optimal solution, respectively, and their cardinalities.

We prove (see Lemma 5.2) that $E(OPT(\mathcal{C}))$ is at least a constant fraction of $OPT(\sigma)$ and that $OPT(\mathcal{C})$ is within a constant fraction of $E(OPT(\mathcal{C}))$ with probability tending to 1 as the optimal solution grows. We present the deterministic procedure in section 5.2. We denote by $ON_D(\mathcal{T})$ the set of calls accepted by the deterministic procedure, out of some sequence $\mathcal{T}$, and its cardinality. We prove (see Lemma 5.8) that for any sequence of candidates $\mathcal{C}$, $OPT(\mathcal{C}) = O(\log n)ON_D(\mathcal{C})$. Combining the two statements (for the randomized filter and for the deterministic procedure), we conclude with the following theorem.

THEOREM 5.1. *There exists an $O(\log n)$-competitive algorithm for call control on the $n \times n$ mesh such that for any $\delta \in (0, 1]$ and any sequence σ, $Pr[ON(\sigma) \geq (1-\delta)E(ON(\sigma))] \geq 1 - 2\exp(-\frac{\delta^2 OPT(\sigma)}{O(\log^4 n)})$.*

Our algorithm thus has asymptotically optimal $O(\log n)$-competitive ratio and it is close to the expectation with probability tending to 1 as $\frac{OPT(\sigma)}{\log^4 n}$ tends to infinity.

5.1. The randomized filter. The $n \times n$ two dimensional mesh G is partitioned into a set of disjoint squares. Let $B = \lfloor \log n \rfloor$. We assume n large enough such that $B \geq 16$. Let $L = \lfloor \gamma \log n \rfloor$ for $\gamma = 13$. Let $s = \lfloor \frac{n}{L} \rfloor$, and let $s_1 = n \bmod L = n - s \cdot L$. We partition the mesh into $s \times s$ submeshes of logarithmic size. The partition of the mesh is obtained by segmenting each of its sides into $s - s_1$ contiguous segments of size L and then s_1 segments of size $L+1$. Note that with the above assumption on B this segmentation is always attainable, and that s_1 can be 0, in which case each side is segmented into s segments of size L. Every submesh is called a *square*, even if the size of its two sides differs by 1.

The first *ring* in a square S consists of all nodes of S that either are incident to a node outside of S or are on the border of the mesh. Recursively, the *ith ring* of S

for $i > 1$ consists of all nodes of S that are incident to a node of ring $i-1$ of S. Each ring, except the innermost, forms a rectangle of nodes. The first ring of a square is also called the *border* of the square. Denote by S_v the square containing vertex v.

In any square S we define three *regions* S^1, S^2, and S^3. Region S^1 consists of rings 1 to $2B$, region S^2 consists of rings $2B+1$ to $4B$, and region S^3 is formed by rings $4B+1$ to $6B$. The remaining part of S is called the *central region* of S (see Figure 5.1). The central region is a rectangle with sides of size at least B.

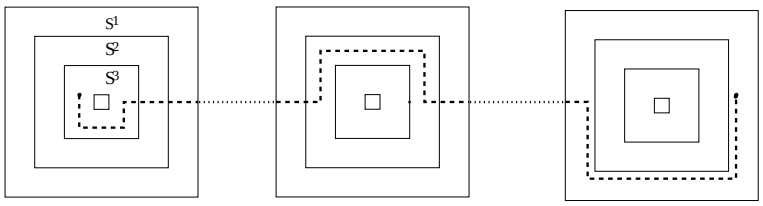


FIG. 5.1. *The routing of a long call.*

The randomized filter makes the following random choice before starting the processing of the sequence.

For every square S choose one of the regions S^1, S^2, S^3 with equal probability.

The region that has been chosen for square S is called the *selected region* of S.

Every call $(s,t) \in \sigma$, when presented, is submitted to the following test:

Add call (s,t) to $\mathcal{C}$ if s is not in the selected region of S_s *and* t is not in the selected region of S_t, otherwise discard (s,t).

As a result of this filtering procedure, no call (s,t) of $\mathcal{C}$ has an endpoint in the selected region of a square. The selected region will be used later to route calls through the square without blocking calls with endpoints inside the square.

LEMMA 5.2. *For any input sequence σ, $E(OPT(\mathcal{C})) \geq \frac{1}{3}OPT(\sigma)$. For any constant $\delta \in (0,1]$, for any sequence σ, $Pr[OPT(\mathcal{C}) \geq (1-\delta)E(OPT(\mathcal{C}))] \geq 1 - 2\exp(-\frac{OPT(\sigma)\delta^2}{82(\lfloor \gamma \log n \rfloor)^4})$.*

Proof. For the first part of the lemma we prove that every call (s,t) of $OPT(\sigma)$ is part of set $\mathcal{C}$ with probability at least $1/3$. A call (s,t) becomes a candidate if both s and t are not in the selected region of S_s and S_t. We distinguish between (i) calls with both endpoints in the same square and (ii) calls with endpoints in different squares.

(i) $S_s = S_t$. If s and t are in the same region of S_s, then this region is not selected with probability $2/3$. If s and t are in different regions, the probability that the region not containing s and t is selected is $1/3$. The claim then follows.

(ii) $S_s \neq S_t$. Vertex s (resp., t) is outside the selected region of S_s (resp., S_t) with probability $2/3$. The probability that both s and t are outside a selected region is then at least $4/9$. The first part of the claim is thus proved.

For the second part of the claim we use the "independent bounded differences inequality" in the formulation proposed by Maurey.

LEMMA 5.3 (see [10]). *Let $X_1, \ldots, X_m$ be independent random variables with X_k taking values in a set A_k for each k. Suppose that the (measurable) function $f : \prod_{k=1}^m A_k \to \Re$ satisfies $|f(\underline{x}) - f(\underline{x}')| \leq c_k$, whenever the vectors $\underline{x}$ and $\underline{x}'$ differ only in the kth coordinate, for some constant c_k. Let Y be the random variable $f(X_1, \ldots, X_m)$. Then, for any $t > 0$,*

$$Pr[|Y - E(Y)| \geq t] \leq 2\exp\left(\frac{-2t^2}{\sum_k c_k^2}\right).$$

In our problem we consider $m = s^2 = (\lfloor \frac{n}{L} \rfloor)^2$ independent random variables $X_1, \ldots, X_m$ to indicate the selected region for every square. Every random variable assumes one of three values with equal probability. For the function $f(\underline{x})$, we will use the function $|OPT(\sigma) \cap \mathcal{C}(\underline{x})|$, where $\underline{x}$ ranges over all the possible assignments of $\underline{x} = (X_1, \ldots, X_m)$. Note that $|OPT(\mathcal{C}(\underline{x}))| \geq |OPT(\sigma) \cap \mathcal{C}(\underline{x})|$. A different assignment of variable X_k results in a maximum absolute variation c_k for $f(\underline{x})$, where c_k is equal to the number of calls of $OPT(\sigma)$ with an endpoint in the kth square. For a square that does not contain any endpoint of a call of the optimal solution, we have $c_k = 0$.

Let K be the set of squares that contain the endpoint of at least one call of the optimal solution. We also denote by K the cardinality of the set K. Every square contains at most $2(L+1)^2$ edges. This is clearly a bound on the number of calls with both endpoints in the same square that can be accepted in a solution. The number of calls with only one endpoint in a square that can be accepted in a solution is bounded by the the number of edges that have one endpoint in the square and one endpoint outside of it, i.e., $4(L+1)$. Altogether, $c_k \leq 3(L+1)^2$ (using that L is large enough) for any square $k \in K$.

We have $\mu = E_{\underline{x}}(OPT(\sigma) \cap \mathcal{C}(\underline{x})) \geq \frac{1}{3} OPT(\sigma)$, and clearly $OPT(\sigma) \geq K/2$. From the bounded differences inequality of Lemma 5.3 it follows that

$$
\begin{aligned}
&Pr[|OPT(\sigma) \cap \mathcal{C}(\underline{x})| < \delta\mu] < 2\exp\left(\frac{-2(\delta\mu)^2}{\sum_k c_k^2}\right) \\
&\leq 2\exp\left(\frac{-2(\delta\frac{OPT(\sigma)}{3})^2}{K \cdot 9(L+1)^4}\right) \leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{81(L+1)^4}\right) \\
&\leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{82L^4}\right) \leq 2\exp\left(\frac{-\delta^2 OPT(\sigma)}{82(\lfloor \gamma \log n \rfloor)^4}\right),
\end{aligned}
$$

from which the claim follows. □

5.2. The deterministic procedure for meshes. The deterministic procedure receives as input the set of candidate calls $\mathcal{C}$ accepted by the randomized filter in the order they appear in the input sequence σ.

We follow KT and partition the set of calls into the set of *long* and the set of *short* calls. A call (s,t) is a short call if both s and t are in the same square and a long call if s and t are in different squares. Set $\mathcal{C}$ is partitioned into the set of long calls $\mathcal{L} = \{(s,t) \in \mathcal{C} : S_s \neq S_t\}$ and the set of short calls $\mathcal{S} = \{(s,t) \in \mathcal{C} : S_s = S_t\}$. Long and short calls are dealt with differently by the algorithm. However, as opposed to KT, we will accept both long calls and short calls at the same time.

The selected region of every square is dedicated to the routing of long calls through the square. For every square S, we number the columns from left to right and the rows from top to bottom. The routes assigned to long calls will traverse the border between two adjacent squares only on columns or rows in the interval $6B+1, \ldots, 7B$. A *crossbar* row (column) of square S is defined as a path on a row (column) in the interval $6B+1, \ldots, 7B$ connecting the central region of S to the closest vertex of the border of S. For every square S and two rings a and b, with ring a of index higher than ring b, define a *straight-line extension* of ring a as a path from a corner of ring a to ring b that does not include edges of a. Observe that the straight-line extensions starting from two distinct rings of a square are edge-disjoint.

We say that a ring, a crossbar row or column, or a straight-line extension is assigned to an accepted call if they contain at least one edge of the path assigned to

the accepted call.

We first describe the algorithm for long calls and then the algorithm for short calls.

Long calls.

Every call (s,t) of $\mathcal{L}$ is first submitted to the following test:

1. If a short or a long call with endpoint in S_s or S_t has been previously accepted, then discard (s,t). Otherwise add (s,t) to set $\mathcal{L}'$.

Calls of $\mathcal{L}'$ are dealt with using the idea of KT to build a simulated network G', with links of higher capacity, and run AAP on this network. A vertex of G' is associated with every square of the original mesh. Two vertices of G' are connected by an edge if and only if there is at least one edge in the original mesh that connects two nodes, each on the border of one of the corresponding squares (i.e., the two squares are adjacent in the mesh).

Every call (s,t) of $\mathcal{L}'$ is transformed into a call between the two vertices of G' associated with S_s and S_t and then submitted to the AAP algorithm. If AAP accepts (s,t), then (s,t) is added to $ON_D(\mathcal{C})$, the set of calls accepted by the algorithm. Calls not accepted by AAP are discarded.

We use the AAP algorithm with the parameter $\epsilon = 6/7$. Observe that one can use AAP with this value of ϵ, since the capacity B of the edges in the simulated network is high enough. To see that, recall that we assumed $B = \lfloor \log n \rfloor \geq 16$. Thus (see also Appendix A)

$$\epsilon \log D + 1 + \epsilon \leq \frac{6}{7}\log n + 1 + \frac{6}{7} \leq \frac{6}{7}\log n + 2 \leq \frac{6}{7}(B+1) + \frac{1}{8}B \leq B \ .$$

If AAP accepts call (s,t) it also returns a route in G' that has a straightforward interpretation as a sequence of squares $S_1, \ldots, S_p$ in the original mesh with $S_1 = S_s$ and $S_p = S_t$. Our algorithm will assign call (s,t) to a path (i) from s to the border between S_1 and S_2, (ii) from the border between S_{i-1} and S_i to the border between S_i and S_{i+1}, $i = 2, \ldots, p-1$, and (iii) from the border between S_{p-1} and S_p to t. In the following description we give the details of this route assignment.

(i) From s to the border between S_1 and S_2.
Without loss of generality assume that the border between S_1 and S_2 is on a column. We consider two cases: (a) s is outside the central area; (b) s is inside the central area.

(a) The call is routed from s through the ring containing s until it meets an unassigned crossbar row leading to the border between S_1 and S_2; this crossbar row is followed until the border between S_1 and S_2.

(b) The call is routed from s through the ring containing s until a corner of the ring is reached, where a straight-line extension is followed until ring $6B$. From there the path continues as in point (a).

(ii) From the border between S_{i-1} and S_i to the border between S_i and S_{i+1}.
Without loss of generality, assume that the border between S_{i-1} and S_i and the border between S_i and S_{i+1} are on columns of G. The path enters the border between S_{i-1} and S_i on a crossbar row that is followed until an unassigned ring of the selected region. The path is routed through the ring of the selected region until an unassigned crossbar row leading to the border between S_i and S_{i+1} is met. This crossbar row is followed until the border between S_i and S_{i+1}.

(iii) From the border between S_{p-1} and S_p to t. This case is equal to case (i).

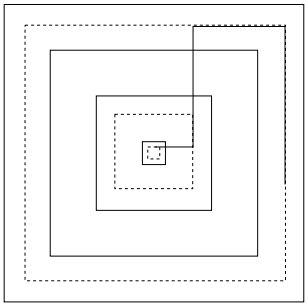


FIG. 5.2. *The routing of short calls.*

Figure 5.1 shows the routing of a long call with an endpoint in the first region and an endpoint in the third region.

Short calls.

A short call (s, t) is accepted on the basis of the following test:

1. If the ring containing s or the ring containing t has been previously assigned, then discard (s, t).
2. If either s or t, but not both, is in the central area, the other vertex is in region S^1 or S^2, but all rings in the range from $4B + 1$ to $6B$ are already assigned, then discard (s, t).
3. Otherwise, add (s, t) to $ON_D(\mathcal{C})$.

In the following we describe the route assigned to an accepted call (s, t).

If s and t are on the same ring, we route (s, t) on a path contained in the ring of s and t. Otherwise, we assume without loss of generality that the ring of s is internal to the ring of t and distinguish between two cases:

1. Both s and t are inside or are both outside the central area. We route (s, t) through the ring of s until a corner of the ring. There, we continue on a straight-line extension until the ring of t, which is then followed until t itself.
2. Either s or t, but not both, is in the central area. One of the rings from $4B + 1$ to $6B$ must be unassigned and will be assigned to (s, t) (possibly the ring containing t). A straight-line extension from a corner of the ring of s to an unassigned ring of S^3 not intersecting the crossbar row or column assigned to the single long call with endpoint in S_s is assigned to (s, t). The path follows the ring of s until the corner of the selected straight-line extension that is followed until the assigned ring of S^3. There, if t is in S^3, we follow the ring until t, otherwise we continue as at point 1.

Figure 5.2 shows the routing of short calls.

Proof of correctness. We prove that the algorithm described above routes accepted calls on edge-disjoint paths. To that end, we prove the following lemmas.

LEMMA 5.4. *The maximum number of calls routed between two adjacent squares is B.*

Proof. Each edge in the simulated network has capacity B and AAP does not violate the capacity constraints. □

LEMMA 5.5. *Let S be a square. There are at most $2B$ calls routed through square S.*

Proof. Every call routed through a square consumes two units of bandwidth on the edges incident to the vertex of G' associated with the square. The overall

bandwidth on the edges incident to a vertex of G' is $4B$; thus at most $2B$ calls are routed through a square. □

LEMMA 5.6. *The following claims hold at any time during the execution of the algorithm:*

1. *Every crossbar row or column is assigned to at most a single call.*
2. *Every ring is assigned to at most a single call.*
3. *Every straight-line extension is assigned to at most a single call.*

Proof. Clearly the claims hold before any call is accepted. We now assume that the claims hold before call (s,t) is accepted and prove that they still hold after call (s,t) is accepted.

1. Assume (s,t) is accepted on a path crossing the border between S and S'. Without loss of generality assume that the border is on a column. By Lemma 5.4 at most $B-1$ (long) calls have been previously routed between S and S'. We conclude that there is an unassigned crossbar row of S and an unassigned crossbar row of S' leading to the border between S and S', both contained in the same row, that can be assigned to (s,t).
2. Two cases: (i) (s,t) is a long call; (ii) (s,t) is a short call. (i) We prove that the rings assigned to call (s,t) are not assigned to any other call. Call (s,t) is accepted by AAP on a path crossing a sequence of squares $S_1,\ldots,S_p$, with $S_1 = S_s$ and $S_p = S_t$. A ring for every square S_i is assigned to (s,t). We first consider squares S_s and S_t. By step 1 of the algorithm for long calls, no long or short call is previously accepted in S_s and in S_t. Since vertices s and t are outside the selected regions of S_s and of S_t, the rings containing s and t are unassigned when (s,t) is accepted. Call (s,t) is also assigned to a ring of the selected region in every square S_i, $i = 2,\ldots,p-1$. By Lemma 5.5, at most $2B-1$ rings of the selected region of a square S_i are previously assigned when (s,t) is accepted. A ring of the selected region of S_i is then still available to be assigned to (s,t). (ii) By steps 1 and 2 of the routing algorithm for short calls, if (s,t) is accepted, it is assigned to at most three rings which were not previously assigned to any other call.
3. All the straight-line extensions in a square are edge-disjoint. A straight-line extension is assigned to a call if it starts from a ring that is also assigned to that call. By point 2 of the claim, every ring is assigned to at most a single call. Therefore, every straight-line extension is assigned to at most a single call. □

We can now prove the main lemma.

LEMMA 5.7. *Every pair of accepted calls is routed on two edge-disjoint paths.*

Proof. By Lemma 5.6, every crossbar row or column, ring, and straight-line extension is assigned to at most a single call. Rings and straight-line extensions are edge-disjoint, as are rings and crossbar rows and columns. Only straight-line extensions from the central region of a square may intersect crossbar rows or columns. We prove that the edges common to a straight-line extension from a ring of the central region and to a crossbar row or column are not assigned to more than one call.

Consider square S. We first exclude intersections between calls with an endpoint in S and calls that are routed through S. By point (i)b of the routing algorithm for long calls and by step 2 of the routing algorithm for short calls, a straight-line extension from a ring of the central region is followed by the route of a call at most until a ring of S^3. In this case the selected region in S is either S^1 or S^2. Therefore, by point (ii) of the routing algorithm for long calls, a call routed through S includes

only edges of S^1 and S^2 that belong to a crossbar row or column. Therefore, there is no intersection with calls routed through S.

We now consider a long call (s,t) with an endpoint, say, s, in S and a call (s',t') assigned to a straight-line extension from a ring of the central region. By step 1 of the algorithm for long calls, (s',t') must be a short call accepted after (s,t), (s,t) being the single long call with endpoint in S that is accepted. There could be a potential intersection only if (s',t') is in the central region, and (s,t) has an endpoint in S^3. There are four possible straight-line extensions that connect the ring of s' with a ring of S^3. By step 2 of the algorithm for short calls, (s',t') is assigned to a straight-line extension that does not overlap with the crossbar row or column assigned to (s,t). No edge of S^3 on the chosen straight-line extension is thus assigned to more than one call. □

The analysis. We conclude the proof that the algorithm is $O(\log n)$-competitive by showing the following lemma.

LEMMA 5.8. *For any set of candidate calls $\mathcal{C}$, $OPT(\mathcal{C}) = O(\log n) ON_D(\mathcal{C})$.*

Proof. We will prove the following lemmas:

1. $OPT(\mathcal{L} \setminus \mathcal{L}') = O(\log n)\ ON_D(\mathcal{C})$ (Lemma 5.9).
2. $OPT(\mathcal{L}') = O(\log n)\ ON_D(\mathcal{L}')$ (Lemma 5.10).
3. $OPT(\mathcal{S}) = O(\log n)\ ON_D(\mathcal{C})$ (Lemma 5.11).

The proof then easily follows:

$$\begin{aligned} OPT(\mathcal{C}) = OPT(\mathcal{L} \cup \mathcal{S}) &\leq OPT(\mathcal{L} \setminus \mathcal{L}') + OPT(\mathcal{L}') + OPT(\mathcal{S}) \\ &= O(\log n)\ ON_D(\mathcal{C})\ . \end{aligned}$$ □

LEMMA 5.9. $OPT(\mathcal{L} \setminus \mathcal{L}') = O(\log n)\ ON_D(\mathcal{C})$.

Proof. A call (s,t) of $\mathcal{L}$ is discarded if a long or a short call with an endpoint in S_s or in S_t has been previously added to $ON_D(\mathcal{C})$. At most $4(L+1)$ long calls with an endpoint in a given square can be accepted by the optimal solution, since the border of S is formed by at most $4(L+1)$ vertices (and edges). Then, for every call accepted in $ON_D(\mathcal{C})$, at most $8(L+1)$ calls of $\mathcal{L} \setminus \mathcal{L}'$ are accepted by the optimal solution. As $L = \lfloor \gamma \log n \rfloor$ the claim follows. □

LEMMA 5.10. $OPT(\mathcal{L}') = O(\log n)\ ON_D(\mathcal{L}')$.

Proof. Every call of $\mathcal{L}'$ is submitted to the AAP algorithm on G' with edges of bandwidth $B = \lfloor \log n \rfloor$. Note that this capacity is large enough to allow AAP to work correctly if we use the AAP algorithm with $\epsilon = 6/7$ (and using the assumption that $B \geq 16$). The benefit $OPT(\mathcal{L}')$, obtained by an optimal solution on the set $\mathcal{L}'$, is bounded by the benefit obtained by an optimal algorithm on G' with edges of bandwidth $L+1$, which is the maximum number of calls that can be routed between two adjacent squares. The AAP algorithm is still $O(\log n)$-competitive even if the bandwidth used by the on-line algorithm (AAP) on the edges is smaller by a constant multiplicative factor than the bandwidth used by the off-line algorithm (see [2, 16]; for completeness this is also proved in the appendix). In our case this constant factor is at most 14. The claim then follows. □

LEMMA 5.11. $OPT(\mathcal{S}) = O(\log n)\ ON_D(\mathcal{C})$.

Proof. Let us restrict our attention to a single square. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the set of calls discarded at steps 1 and 2 of the algorithm for short calls. We have $OPT(\mathcal{S}) \leq OPT(\mathcal{S}_1) + OPT(\mathcal{S}_2) + ON_D(S)$. The proof of the lemma derives from the two following statements: (i) $OPT(\mathcal{S}_1) = O(\log n)\ ON_D(\mathcal{C})$; (ii) $OPT(\mathcal{S}_2) \leq 8\ ON_D(\mathcal{C})$.

(i) For every accepted call, at most three rings of a square are assigned by the algorithm to that call. The number of edges incident to a ring is at most $4(L+1)$. This

is a bound on the number of calls with an endpoint on a ring that can be accepted in an optimal solution. Therefore, the number of calls of $OPT(\mathcal{S})$ that are discarded at step 1 of the algorithm for short calls, per each call in $ON_D(\mathcal{C})$, is at most $12(L+1)$. As $L = \lfloor \gamma \log n \rfloor$, this proves the first statement.

(ii) Short calls with an endpoint in the central region and an endpoint outside the central region are discarded at step 2 of the algorithm if all the rings of S^3 outside the central region are assigned. At most two rings of S^3 are assigned to an accepted call. If a short call is discarded at step 2, we thus have the evidence that B calls with an endpoint in S have already been accepted. For the sake of the proof we charge, for every accepted long call, the value of $1/2$ to S and the value of $1/2$ to the square containing its other endpoint. A square is therefore already charged with at least the value of $B/2$ if a short call of the square is discarded at step 2. The number of short calls of $OPT(\mathcal{S}_2)$ with endpoints in S is bounded by $4B$, as this is the number of vertices on the border of the central region. The ratio between the number of calls of $\mathcal{S}_2$ with endpoints in S that belong to $OPT(\mathcal{S}_2)$, and the number of calls of $ON_D(\mathcal{C})$ charged to S, is thus at most 8. This proves the second statement. ∎

Appendix A. The AAP algorithm. For completeness we describe a restricted version of the AAP algorithm. We consider the case where all calls are of infinite duration, request bandwidth of 1, and are of uniform benefit that without loss of generality we assume equals to D, the length of the longest simple path in the network. Let $u(e)$, for $e \in E$, be the capacity of edge e and assume that for all $e \in E$,

$$u(e) \geq \epsilon \log D + 1 + \epsilon \quad \text{for some } 0 < \epsilon \leq 1. \tag{A.1}$$

Let $b_j(e)$ be the number of calls routed trough edge e by AAP among the first j presented calls. Define the relative load of edge e, just after call j has been processed, to be $\lambda_j(e) = \frac{b_j(e)}{u(e)}$. Define the cost of edge e just after call j has been processed to be $c_j(e) = u(e)[\mu^{\lambda_j(e)} - 1]$, where $\mu = 2^{1+1/\epsilon}D$. A request (s_j, t_j) is accepted to path P_j if $\sum_{e \in P_j)} \frac{1}{u(e)} \cdot c_{j-1}(e) \leq D$.

The above algorithm has competitive ratio $O(2^{1/\epsilon}/\epsilon + 2^{1/\epsilon} \log D)$. We give below a sketch of the proof. The proof follows the proof that appears in [8].

THEOREM A.1. *The above AAP algorithm has competitive ratio* $2^{1+1/\epsilon} \log \mu + 1$.

Proof. Let $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$, be the sequence of requests. Let A be the set of indices of requests that are accepted by AAP and let A' be the set of indices of requests that are rejected by AAP but are accepted by the adversary. Let $C = \sum_{e \in E} c_n(e)$. Let $\mathcal{Q}(\sigma)$ denote the benefit obtained by algorithm AAP on sequence σ. We prove the following two claims that yield the desired result.

1. $C \leq (2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D$.
2. $|A'| \cdot D \leq C$.

The result follows from the above two inequalities since $\mathcal{Q}(\sigma) = |A| \cdot D$, and $OPT(\sigma) \leq \mathcal{Q}(\sigma) + |A'| \cdot D$.

The proof of inequality 1 can be found in [8]. We repeat here the proof of inequality 2 since we slightly modify it below to give the corollaries we need.

For a call σ_j that is accepted by the off-line algorithm let P_j^* be the path that is used by the off-line algorithm to accept this call. For any $j \in A'$

$$D < \min_{P_j} \sum_{e \in P_j} \frac{1}{u(e)} c_{j-1}(e) \leq \sum_{e \in P_j^*} \frac{1}{u(e)} c_{j-1}(e) \ .$$

We get

$$\begin{aligned}
|A'| \cdot D &< \sum_{j \in A'} \sum_{e \in P_j^*} \frac{1}{u(e)} c_{j-1}(e) \\
&\leq \sum_{j \in A'} \sum_{e \in P_j^*} \frac{1}{u(e)} c_n(e) \\
&= \sum_e c_n(e) \sum_{j \in A' : e \in P_j^*} \frac{1}{u(e)} \\
&\leq \sum_e c_n(e) \\
&= C.
\end{aligned}$$

The last inequality follows since the capacity of the adversary on any edge e is $u(e)$, that is, it can route at most $u(e)$ calls though edge e. □

We now give two corollaries of the above theorem.

COROLLARY A.2. *If for all $e \in E$, $u(e)$ satisfied condition* A.1, *and the off-line algorithm has capacity $c \cdot u(e)$ for each edge e, for some constant c, then AAP is $(1 + c(2^{1+1/\epsilon} \log \mu))$-competitive.*

Proof. We slightly modify the above proof for inequality 2. Since for each $e \in E$ the capacity available to the off-line algorithm is $c \cdot u(e)$, then the last inequality in the proof would yield

$$|A'| \cdot D < \sum_e c_n(e) \cdot c = C \cdot c \ .$$

Thus we get $(|A'| \cdot D)/c \leq (2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D$. And we get

$$\begin{aligned}
OPT(\sigma) &\leq \mathcal{Q}(\sigma) + |A'| \cdot D \leq \mathcal{Q}(\sigma) + c(2^{1+1/\epsilon} \log \mu) \cdot |A| \cdot D \\
&= (1 + c(2^{1+1/\epsilon} \log \mu)) \cdot \mathcal{Q}(\sigma). \qquad \square
\end{aligned}$$

COROLLARY A.3. *If AAP, with $\epsilon = 1$, has for each $e \in E$ capacity $u(e) = \log 4D$, while the off-line algorithm has for all e capacity* 1, *then the competitive ratio is* 5.

Proof. We use $\epsilon = 1$ and slightly modify the above proof of inequality 2. We again modify the last inequality of the the proof. Since the off-line algorithm has for each e only capacity $u(e)/\log 4D$, we get

$$|A'| \cdot D < \sum_e c_n(e)/\log 4D \ ,$$

i.e.,

$$|A'| \cdot D < C/\log 4D \ .$$

We get

$$\begin{aligned} OPT(\sigma) &\le \mathcal{Q}(\sigma) + |A'| \cdot D \\ &\le \mathcal{Q}(\sigma) + C/\log 4D \\ &\le \mathcal{Q}(\sigma) + ((2^{1+1/\epsilon}\log\mu)\cdot|A|\cdot D)/\log 4D \\ &= \mathcal{Q}(\sigma) + ((2^{1+1/\epsilon}\log\mu)\cdot|A|\cdot D)/\log\mu \\ &= \mathcal{Q}(\sigma) + 4\cdot|A|\cdot D = 5\cdot\mathcal{Q}(\sigma)\ . \qquad \square \end{aligned}$$

Acknowledgments. Special thanks are due to Amos Fiat for suggesting to us to look into the relationship between the competitive ratio and the deviation of the result from its expectation in randomized benefit on-line algorithms. We thank Dimitri Achlioptas and Hannah Bast for useful discussions and suggestions. We also thank an anonymous referee for very useful comments.

REFERENCES

[1] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, *On-line competitive algorithms for call admission in optical networks*, in Proceedings of the Fourth European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer, Berlin, 1996, pp. 431–444.

[2] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput-competitive online routing*, in Proceeding of the 34th Symposium Foundations of Computer Science, 1993, pp. 32–40.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, *Competitive non-preemptive call control*, in Proceedings of the Fifth Annual Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 312–320.

[4] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani, *On-line admission control and circuit routing for high performance computing and communication*, in Proceedings of the 35th Symposium Foundations of Computer Science, 1994, pp. 412–423.

[5] Y. Bartal, A. Fiat, and S. Leonardi, *Lower bounds for on-line graph problems with application to on-line circuit and optical routing*, in Proceedings of the 28th Symposium Theory of Computing, 1996, pp. 531–540.

[6] Y. Bartal and S. Leonardi, *On-line routing in all-optical networks*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1256, Springer, Berlin, 1997, pp. 516–526.

[7] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[8] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998; also available online from http://www.cup.org/Titles/56/0521563925.html.

[9] M. F. G. Cornuejols and G. Nemhauser, *Location of bank accounts to optimize float*, Management Sci., 62 (1977), pp. 789–810.

[10] C. M. Diarmid, *On the method of bounded difference*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 141, J. Siemon, ed., Cambridge University Press, Cambridge, UK, 1989.

[11] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 700, Springer, 1993, pp. 64–75.

[12] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC99), 1999, pp. 19–28.

[13] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[14] J. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, MIT, Cambridge, MA, 1996.

[15] J. Kleinberg and R. Rubenfield, *Short paths in expander graphs*, in Proceedings of the 37th Symposium Foundations of Computer Science, 1996, pp. 86–95.

[16] J. Kleinberg and E. Tardos, *Disjoint paths in densely embedded graphs*, in Proceedings of the 36th Symposium Foundations of Computer Science, 1995, pp. 52–61.

[17] M. Kramer and J. van Leeuwen, *The complexity of wire routing and finding the minimum area layouts for arbitrary vlsi circuits*, in Advances in Computing Research 2: VLSI Theory, F. Preparata, ed., JAI Press, London, 1984, pp. 129–146.

[18] S. Leonardi and A. Marchetti-Spaccamela, *On-line resource management with applications to routing and scheduling*, in Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 944, Springer, Berlin, 1995, pp. 303–314.

[19] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer program*, J. Comput. System Sci., 2 (1988), pp. 130–143.

[20] D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 113–145

FIXED-PARAMETER TRACTABILITY, DEFINABILITY, AND MODEL-CHECKING*

JÖRG FLUM† AND MARTIN GROHE‡

Abstract. In this article, we study parameterized complexity theory from the perspective of logic, or more specifically, descriptive complexity theory.

We propose to consider parameterized *model-checking* problems for various fragments of first-order logic as generic parameterized problems and show how this approach can be useful in studying both fixed-parameter tractability and intractability. For example, we establish the equivalence between the model-checking for existential first-order logic, the homomorphism problem for relational structures, and the substructure isomorphism problem. Our main tractability result shows that model-checking for first-order formulas is fixed-parameter tractable when restricted to a class of input structures with an excluded minor. On the intractability side, for every $t \geq 0$ we prove an equivalence between model-checking for first-order formulas with t quantifier alternations and the parameterized halting problem for alternating Turing machines with t alternations. We discuss the close connection between this *alternation hierarchy* and Downey and Fellows' W-hierarchy.

On a more abstract level, we consider two forms of definability, called *Fagin definability* and *slicewise definability*, that are appropriate for describing parameterized problems. We give a characterization of the class FPT of all fixed-parameter tractable problems in terms of slicewise definability in finite variable least fixed-point logic, which is reminiscent of the Immerman–Vardi theorem characterizing the class PTIME in terms of definability in least fixed-point logic.

Key words. parameterized complexity, model-checking, descriptive complexity

AMS subject classifications. 68Q15, 03C13, 03C40

PII. S0097539799360768

1. Introduction.

Parameterized complexity is a branch of complexity theory which has matured in the last 10 years, as witnessed in the culminating monograph [10]. It gives a framework for a refined complexity analysis of hard algorithmic problems. The basic idea can best be explained by an example: Consider the problem of evaluating a query in a relational database. This problem usually has a high complexity (depending on the query language, of course, but the problem is NP-complete even for the very basic conjunctive queries [5]). The main factor contributing to this complexity is the length of the query. In practice, however, queries are usually short, certainly much shorter than the size of the database. Thus when analyzing the complexity of the problem we should put much more emphasis on the size of the database than on the length of the query. An algorithm evaluating a query of length k in a database of size m in time $O(2^k \cdot m)$ is therefore much better than one performing the same task in time $O(m^{k/2})$, although both are exponential.

Parameterized complexity theory studies problems whose instances are *parameterized* by some function of the input, such as the length of the query in our example. The idea is to choose the parameterization in such a way that it can be assumed to take small values for the instances one is interested in. Then the complexity of an algorithm is measured not only in the size of the input but also in terms of the param-

*Received by the editors August 31, 1999; accepted for publication (in revised form) February 12, 2001; published electronically June 5, 2001.
http://www.siam.org/journals/sicomp/31-1/36076.html
†Institut für Mathematische Logik, Eckerstr. 1, 79104 Freiburg, Germany (flum@sun2.ruf.uni-freiburg.de).
‡Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 851 S. Morgan St. (M/C 249), Chicago, IL 60607-7045 (grohe@math.uic.edu).

eter. A parameterized problem is *fixed-parameter tractable* if there is an algorithm solving it in time $f(k) \cdot n^c$, where n denotes the size of the input, k the parameter, and $f : \mathbb{N} \to \mathbb{N}$ is a computable function and $c > 0$ a constant.

Parameterized complexity theory provides methods for proving problems to be fixed-parameter tractable but also gives a framework for dealing with apparently intractable problems in a similar way that the theory of NP-completeness does in classical complexity theory.

The purpose of this article is to establish a very fruitful connection between parameterized complexity theory and logic. Our approach is that of descriptive complexity theory. We study the definability of parameterized problems and try to obtain information about the parameterized complexity of the problems through the syntactical structure of the defining sentences. On the one hand, we use this approach to prove that certain problems are tractable because they can be defined by syntactically simple formulas. On the other hand, we characterize classes of intractable problems by syntactical means.

Central to our approach are *parameterized model-checking* problems of the following form. For a class Φ of formulas, we let $MC(\Phi)$ be the problem

$MC(\Phi)$		
	Input:	A finite structure $\mathcal{A}$.
	Parameter:	A sentence $\varphi \in \Phi$.
	Question:	Does $\mathcal{A}$ satisfy φ?

In most cases, Φ will be a fragment of first-order logic.

After a preliminary section, we discuss some basic facts about parameterized model-checking problems in section 3. In section 4 we introduce two notions of definability of parameterized problems, which we call slicewise definability and Fagin definability, and relate them to model-checking. We then show how Fagin definability can be used to establish the fixed-parameter tractability of various problems.

In section 5 we study the parameterized complexity of the model-checking problem for Σ_1-formulas (that is, existential first-order formulas in prenex normal form). We associate a graph with each such formula and use it to establish a surprisingly close connection between this model-checking problem, the homomorphism problem, and the subgraph isomorphism problem. As an application of our result we show that for Σ_1-sentences whose graph has bounded tree-width, the model-checking problem is fixed-parameter tractable, even if inequalities are disregarded in the graph of the formula. Model-checking for formulas with a tree-like graph or hypergraph has recently received much attention (see [6, 24, 18, 16]).

So far we have looked only for tractable cases of the model-checking problem $MC(\Phi)$ that are obtained by restricting the class of formulas Φ. A different approach is to restrict the class of structures where the input structure $\mathcal{A}$ is taken from (see, for example, [7, 29, 17]). We prove a far reaching result: For any class C of graphs with an excluded minor, the model-checking problem for first-order logic is fixed-parameter tractable if the inputs are taken from C. This implies, for example, that parameterized versions of the dominating set problem or the (induced) subgraph isomorphism problem are fixed-parameter tractable when restricted to such classes of graphs.

Our last result on fixed-parameter tractability is a descriptive characterization of the complexity class FPT of all fixed-parameter tractable problems in terms of slicewise definability in finite variable fragments of least-fixed point logic. This simple result can be seen as a parameterized analogue of the well-known Immerman–Vardi

theorem [21, 30] characterizing the class PTIME in terms of definability in least-fixed-point logic.

The final section is devoted to fixed-parameter intractability. We define a hierarchy A[t] of parameterized complexity classes in terms of alternating Turing machine acceptance (t is the number of alternations). This hierarchy can be seen as a parameterized analogue of the polynomial hierarchy. We prove that for all $t \geq 1$, the model-checking problem for Σ_t-formulas is complete for the tth level of this hierarchy. Then we study the relation between our A-hierarchy and Downey and Fellows' W-hierarchy. It is known that the first levels of the respective hierarchies, A[1] and W[1], coincide [4]. We slightly improve a result of Downey, Fellows, and Regan [11] relating W[t], the tth level of the W-hierarchy, to the model-checking problems for a certain fragment of Σ_t. However, the question whether A[t] and W[t] coincide for $t \geq 2$ remains open.

2. Preliminaries.

2.1. Logic. We assume that the reader is familiar with first-order logic; we recall only a few basic notions to fix our notation (compare with [13] for a more detailed introduction of these notions).

In this article, a vocabulary *is a finite set of relation symbols.* Associated with every relation symbol is a natural number, its *arity*. The *arity* of a vocabulary is the maximal arity of the relation symbols it contains. Usually, vocabularies are also permitted to contain function and constant symbols. All results of this article, with the single exception of Theorem 8.4, would remain true if function and constant symbols were allowed, but adding them would not give us any new insights. Therefore, for convenience, we restrict our attention to relational vocabularies. In the following, τ always denotes a vocabulary.

A *τ-structure* $\mathcal{A}$ consists of a set A, called the *universe* of $\mathcal{A}$, and a relation $R^{\mathcal{A}} \subseteq A^r$ for each r-ary relation symbol $R \in \tau$. We synonymously write $\bar{a} \in R^{\mathcal{A}}$ or $R^{\mathcal{A}}\bar{a}$ to denote that the tuple $\bar{a} \in A^r$ belongs to the relation $R^{\mathcal{A}}$. For $\tau \subseteq \tau'$, a τ-structure $\mathcal{A}$ is the *τ-reduct* of a τ'-structure $\mathcal{A}'$ if $A = A'$ and $R^{\mathcal{A}} = R^{\mathcal{A}'}$ for all $R \in \tau$. A τ'-structure $\mathcal{A}'$ is a *τ'-expansion* of a τ-structure $\mathcal{A}$ if $\mathcal{A}$ is the *τ-reduct* of $\mathcal{A}'$.

We consider only finite structures. When we consider classes of structures, they are always assumed to be closed under isomorphism. *STR* denotes the class of all (finite) structures. If C is a class of structures, $C[\tau]$ denotes the subclass of all τ-structures in C. Furthermore, $C[s]$ denotes the class of all structures in C whose vocabulary is at most s-ary. We consider graphs as $\{E\}$-structures $\mathcal{G} = (G, E^{\mathcal{G}})$, where $E^{\mathcal{G}}$ is an irreflexive and symmetric binary relation (i.e., graphs are loop-free and undirected). *GRAPH* denotes the class of all graphs.

The class of all first-order formulas is denoted by FO. Recall that *atomic formulas* are formulas of the form $x = y$ or $Rx_1 \dots x_r$, where $x, y, x_1, \dots, x_r$ are variables and R is an r-ary relation symbol. *Literals* are atomic or negated atomic formulas. A first-order formula φ is in *negation normal form* if negation symbols occur only directly in front of atomic subformulas. φ is *existential* (*universal*) if it is in negation normal form and contains no universal quantifiers (no existential quantifiers, respectively). φ is in *prenex normal form* if it is of the form $Q_1 x_1 \dots Q_k x_k \theta$, where $Q_1, \dots, Q_k \in \{\exists, \forall\}$ and θ is quantifier-free.

EFO (AFO) denotes the class of all existential (respectively, universal) first-order

formulas. For $t \geq 1$, Σ_t denotes the class of all FO-formulas of the form

$$\exists x_{11} \ldots \exists x_{1k_1} \forall x_{21} \ldots \forall x_{2k_2} \ \ldots \ Q x_{t1} \ldots Q x_{tk_t} \ \theta,$$

where $Q = \forall$ if t is even and $Q = \exists$ otherwise and θ is quantifier-free. Π_t-formulas are defined analogously starting with a block of universal quantifiers.

If Φ is a class of formulas of some logic, then $\Phi[\tau]$ denotes the class of all formulas of vocabulary τ in L, and $\Phi[s]$ denotes the class of all formulas in Φ whose vocabulary is at most s-ary. We write $\mathcal{A} \models \varphi$ if, for some τ, $\mathcal{A}$ is a τ-structure, φ is in $\mathrm{L}[\tau]$, and $\mathcal{A}$ is a model of φ.

2.2. Coding issues. We use random access machines (RAMs) with the uniform cost measure as our underlying model of computation (cf. [1]).

Very often, the objects of our computations are structures. Therefore, we have to fix a way of representing structures on a RAM. The two most common ways of doing this are the *array representation* and the *list representation*. For both representations we assume that the universes of our structures are initial segments of the natural numbers; of course this is no real restriction because every structure is isomorphic to one with such a universe.

Both representations start with an encoding of the vocabulary and a natural number representing the size of the universe of the structure. The difference between the two representations is in how relations are stored. In the array representation, a k-ary relation is stored as a k-dimensional array with $0,1$-entries. For graphs, this is just the adjacency matrix. The advantage of this representation is that for each tuple it can be checked in constant time whether it belongs to the relation or not. However, for sparse relations this representation wastes a lot of space.

In the more concise list representation, a relation is represented as a list of all tuples it contains. Clearly, the list representation of a structure can be computed from the array representation in linear time but not vice versa. For graphs $\mathcal{G}$, it is easy to construct the common adjacency list representation from the list representation (in time linear in $|G| +$ (size of the representation), where $|G|$ denotes the number of elements in G). In this article, we always assume that structures are given in the list representation, but all results also hold for the array representation. The *size* of a structure $\mathcal{A}$, denoted by $||\mathcal{A}||$, is defined to be $|A| +$ (size of the list representation of $\mathcal{A}$). The complexity of algorithms on structures is measured in this size. Remark 3.3 shows that this can be relevant.

2.3. Parameterized problems. We recall only those notions of the theory needed in this article. For a comprehensive treatment we refer the reader to Downey and Fellows' recent monograph [10]. A *parameterized problem* is a set $P \subseteq \Sigma^* \times \Pi^*$, where Σ and Π are finite alphabets. Following [10], we usually represent a parameterized problem P in the following form:

P		
	Input:	$x \in \Sigma^*$.
	Parameter:	$y \in \Pi^*$.
	Question:	Is $(x, y) \in P$?

In most cases, we have $\Pi = \{0,1\}$ and consider the parameters $y \in \Pi^*$ as natural numbers (in binary). A natural example is the parameterized version of the well-known *VERTEX COVER* problem:

VC		
	Input:	Graph $\mathcal{G}$.
	Parameter:	$k \in \mathbb{N}$.
	Question:	Does $\mathcal{G}$ have a vertex cover of size k?

Recall that a *vertex cover* of a graph is a set X of vertices such that every edge is incident to one of the vertices in X. Similarly, we can define parameterized versions of *DOMINATING SET (DS)* and *CLIQUE*. (A *dominating set* of a graph is a set X of vertices such that every vertex not contained in X is adjacent to a vertex in X. A *clique* is a set of pairwise adjacent vertices.)

An example where the set of parameters is not $\mathbb{N}$, but the class *GRAPH* of all finite graphs, is the following parameterized *SUBGRAPH ISOMORPHISM (SI)* problem:

SI		
	Input:	Graph $\mathcal{G}$.
	Parameter:	Graph $\mathcal{H}$.
	Question:	Is $\mathcal{H}$ isomorphic to a subgraph of $\mathcal{G}$?

Similarly, we can define parameterized versions of the *INDUCED SUBGRAPH ISOMORPHISM* problem and the *GRAPH HOMOMORPHISM* problem.

DEFINITION 2.1. *A parameterized problem $P \subseteq \Sigma^* \times \Pi^*$ is* fixed-parameter tractable[1] *if there is a computable function $f : \Pi^* \to \mathbb{N}$, a constant $c \in \mathbb{N}$, and an algorithm that, given a pair $(x, y) \in \Sigma^* \times \Pi^*$, decides if $(x, y) \in P$ in time $f(y) \cdot |x|^c$.*

We denote the class of all fixed-parameter tractable problems by FPT.

Of course we can always consider parameterized problems as classical problems and determine their complexity in the classical sense. Clearly, every parameterized problem in PTIME is also in FPT.

The best currently known algorithm for vertex cover *VC* has running time $O(k \cdot n + \max\{1.255^k \cdot k^2, 1.291^k \cdot k\})$ [15], where n denotes the size of the input graph. Thus $VC \in$ FPT.

2.4. Reductions between parameterized problems. It is conjectured that none of the problems *DS*, *CLIQUE*, *SI* is in FPT. As it is often the case in complexity theory, we cannot actually prove this but prove only that the problems are hard for certain complexity classes that are conjectured to contain FPT strictly. To do this we need a suitable concept of reduction. We actually introduce three different types of reduction.

DEFINITION 2.2. *Let $P \subseteq \Sigma^* \times \Pi^*$ and $P' \subseteq (\Sigma')^* \times (\Pi')^*$ be parameterized problems.*

(1) *A* parameterized T-reduction *from P to P' is an algorithm with an oracle for P' that solves any instance (x, y) of P in time $f(|y|) \cdot |x|^c$ in such a way that for all questions $(x', y') \in P'$? to the oracle we have $|y'| \leq g(|y|)$ (for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$).*

P is fixed-parameter T-reducible *to P' (we write $P \leq_{\mathrm{T}}^{\mathrm{fp}} P'$) if there is a parameterized T-reduction from P to P'.*

[1] This is what Downey and Fellows call *strongly uniformly fixed-parameter tractable*. For variants of this definition, and also of Definition 2.2 and the definition of the W-hierarchy, the reader should consult [10].

(2) *A* parameterized m-reduction *from P to P' is an algorithm that computes for every instance (x, y) of P an instance (x', y') of P' in time $f(|y|) \cdot |x|^c$ such that $|y'| \leq g(|y|)$ and*

$$(x, y) \in P \iff (x', y') \in P'$$

(for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$).
P is fixed-parameter m-reducible *to P' (we write $P \leq_{\mathrm{m}}^{\mathrm{fp}} P'$) if there is a parameterized m-reduction from P to P'.*

Whereas every parameterized problem that is in PTIME (when considered as a classical problem) is in FPT, it is not the case that every PTIME many-one reduction between two parameterized problems is also a parameterized m-reduction. To capture both concepts we occasionally use the following third kind of reduction.

DEFINITION 2.3. *Let $P \subseteq \Sigma^* \times \Pi^*$ and $P' \subseteq (\Sigma')^* \times (\Pi')^*$ be parameterized problems.*

A pp m-reduction *from P to P' is a parameterized m-reduction from P to P' that is also a polynomial time many-one reduction from P to P' in the classical sense, i.e., the function f in Definition* 2.2(2) *is a polynomial.*

P is pp m-reducible *to P' (we write $P \leq_{\mathrm{m}}^{\mathrm{fpp}} P'$) if there is a pp m-reduction from P to P'.*

For example, *CLIQUE* $\leq_{\mathrm{m}}^{\mathrm{fp}}$ *SI* by the simple parameterized m-reduction that reduces the instance $(\mathcal{G}, k)$ of *CLIQUE* to the instance $(\mathcal{G}, \mathcal{K}_k)$ of *SI*. Here $\mathcal{K}_k$ denotes the complete graph with k vertices. Note that if we represent integers in binary, this reduction is not a pp m-reduction.

Observe that $\leq_{\mathrm{T}}^{\mathrm{fp}}$, $\leq_{\mathrm{m}}^{\mathrm{fp}}$, and $\leq_{\mathrm{m}}^{\mathrm{fpp}}$ are transitive and that for all P, P' we have

$$P \leq_{\mathrm{m}}^{\mathrm{fpp}} P' \implies P \leq_{\mathrm{m}}^{\mathrm{fp}} P' \quad \text{and} \quad P \leq_{\mathrm{m}}^{\mathrm{fp}} P' \implies P \leq_{\mathrm{T}}^{\mathrm{fp}} P'.$$

Furthermore, if $P \leq_{\mathrm{T}}^{\mathrm{fp}} P'$ and $P' \in \mathrm{FPT}$, then $P \in \mathrm{FPT}$. For any of the reductions $\leq_{\mathrm{T}}^{\mathrm{fp}}, \leq_{\mathrm{m}}^{\mathrm{fp}}, \leq_{\mathrm{m}}^{\mathrm{fpp}}$ we let $\equiv_{\dots}^{\dots}$ denote the corresponding equivalence relation.

We define *hardness* and *completeness* of parameterized problems for a parameterized complexity class (under parameterized m- or T-reductions) in the usual way. For a parameterized problem P, we let $[P]_{\mathrm{m}}^{\mathrm{fp}} := \{P' \mid P' \leq_{\mathrm{m}}^{\mathrm{fp}} P\}$, and for a class P of parameterized problems $[\mathrm{P}]_{\mathrm{m}}^{\mathrm{fp}} := \bigcup_{P \in \mathrm{P}} [P]_{\mathrm{m}}^{\mathrm{fp}}$.

Remark 2.4. Very often, it is natural to think of a parameterized problem P as derived from a (classical) problem $L \subseteq \Sigma^*$ by a *parameterization* $p : \Sigma^* \to \mathbb{N}$ in such a way that $P = \{(x, k) \mid x \in L, k = p(x)\}$.

Slightly abusing notation, we represent such a P in the following form:

P		
	Input:	$x \in \Sigma^*$.
	Parameter:	$p(x)$.
	Question:	Decide if $x \in L$?

As an example, let us reconsider the subgraph isomorphism problem. Instead of taking graph $\mathcal{H}$ as the parameter, we may also consider pairs of graphs $(\mathcal{G}, \mathcal{H})$ as inputs and parameterize the problem by the size of $\mathcal{H}$. In our new notation, this would be the problem:

SI′		
	Input:	Graphs $\mathcal{G}$, $\mathcal{H}$.
	Parameter:	$\|\|\mathcal{H}\|\|$.
	Question:	Is $\mathcal{H}$ isomorphic to a subgraph of $\mathcal{G}$?

It is easy to see, however, that $SI \equiv_{\mathrm{m}}^{\mathrm{fp}} SI'$.

2.5. Parameterized intractability. Some combinatorial problems are provably not fixed-parameter tractable, and others, such as *GRAPH COLORABILITY*, are not fixed-parameter tractable unless PTIME = NP. However, many interesting problems, such as the parameterized *CLIQUE* problem, do not seem to be fixed-parameter tractable, although there is no known way to prove this or reduce it to classical complexity theoretic questions such as PTIME $\stackrel{?}{=}$ NP. To classify such problems, Downey and Fellows (cf. [10]) introduced a hierarchy

$$\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots$$

of classes above FPT. These classes can best be defined in terms of the satisfiability problem for formulas of propositional logic. Formulas of propositional logic are built up from *propositional variables* $X_1, X_2, \ldots$ by taking conjunctions, disjunctions, and negations. The negation of a formula φ is denoted by $\neg\varphi$. We distinguish between *small conjunctions*, denoted by $\wedge$, which are conjunctions of two formulas, and *big conjunctions*, denoted by $\bigwedge$, which are conjunctions over arbitrary finite sets of formulas. Analogously, we distinguish between *small disjunctions*, denoted by $\vee$, and *big disjunctions*, denoted by $\bigvee$.

Every formula φ corresponds to a labeled tree $\mathcal{T}_\varphi$ in a natural way. The *size* of φ is defined to be the number of vertices of $\mathcal{T}_\varphi$. The *depth* of φ is defined to be the maximum number of nodes labeled $\wedge, \bigwedge, \vee, \bigvee$ on a path from the root to a leaf of $\mathcal{T}_\varphi$. Thus when computing the depth, we do not count negations.

A formula is *small* if it contains only small conjunctions and small disjunction. We define $C_0 = D_0$ to be the class of all small formulas. For an $i \geq 1$, we define C_i to be the class of all big conjunctions of formulas in D_{i-1}, and we define D_i to be the class of all big disjunctions of formulas in C_{i-1}. Note that these definitions are purely syntactical; every formula in a C_i or D_i is equivalent to a formula in C_0. Of course the translation from a formula in C_i to an equivalent formula in C_0 usually increases the depth of a formula. For all $i, d \geq 0$ we let $C_{i,d}$ denote the class of all formulas in C_i whose small subformulas have depth at most d (equivalently, we may say that the whole formula has depth at most $d + i$). We define $D_{i,d}$ analogously.

The *weight* of an assignment α for the variables of a propositional formula is the number of variables set to TRUE by α. For any class P of propositional formulas, let *weighted satisfiability for* P be the following parameterized problem:

WSAT(P)		
	Input:	$\varphi \in P$.
	Parameter:	$k \in \mathbb{N}$.
	Question:	Does φ have a satisfying assignment of weight k?

Now we are ready to define the *W-hierarchy*: For every $t \geq 1$, we let

$$\mathrm{W}[t] := \bigcup_{d \geq 0} [\, \mathit{WSAT}(C_{t,d})]_{\mathrm{m}}^{\mathrm{fp}}.$$

In other words, a parameterized problem is in W[t] if there is a $d \geq 0$ such that the problem is fixed-parameter m-reducible to the weighted satisfiability problem for $C_{t,d}$. It is an immediate consequence of the definition of parameterized m-reductions that FPT $\subseteq$ W[1]. Actually, it is conjectured that this inclusion is strict and that W[t] is strictly contained in W[$t+1$] for every $t \geq 1$.

Example 2.5. The parameterized *CLIQUE*-problem is in W[1]. To see this, for every graph $\mathcal{G}$ we describe a propositional formula $\varphi := \varphi(\mathcal{G}) \in C_{1,1}$ such that $\mathcal{G}$ has a clique of size k if and only if φ has a satisfying assignment of weight k. It will be obvious from the construction that φ can be computed from $\mathcal{G}$ in polynomial time.

So let $\mathcal{G}$ be a graph. For all $a \in G$ let X_a be a propositional variable. Let

$$\varphi := \bigwedge_{\substack{a,b \in G, a \neq b \\ ab \notin E^G}} (\neg X_a \vee \neg X_b).$$

Then every satisfying assignment of φ corresponds to a clique of $\mathcal{G}$.

Actually, Downey and Fellows proved the following nontrivial result.

THEOREM 2.6 (Downey and Fellows [8, 9]).

(1) *CLIQUE is* W[1]*-complete under parameterized m-reductions.*

(2) *DS is* W[2]*-complete under parameterized m-reductions.*

Remark 2.7. Downey and Fellows phrase their definition of the W-hierarchy in terms of Boolean circuits rather than propositional formulas. However, since the classes of the hierarchy involve only circuits/formulas of bounded depth, this does not really make a difference (cf. [10]). In their definition of W[t], Downey and Fellows admit more complicated formulas than those in C_t. However, they prove that our definition is equivalent. A surprising by-product of their results is that for every $t \geq 1$ and every $d \geq 0$, the problem $WSAT[D_{t+1,d}]$ is contained in W[t]. It is not hard to prove this result directly, and even easier to prove that $WSAT[D_{1,d}]$ is in FPT. (This explains why we defined only a hierarchy using the C_ts.)

There is another, more serious source of confusion in the various definitions of the W-hierarchy: Downey and Fellows are never really clear about what kind of reductions they are using to define the classes. We decided, more or less in accordance with [10], that parameterized m-reductions are most natural.

We will further discuss the W-hierarchy and other seemingly intractable classes in section 8.

3. Model-checking. In this article we are mainly concerned with the complexity of various *parameterized model-checking problems*. For a set Φ of formulas, we let

$$MC(\Phi) := \big\{(\mathcal{A}, \varphi) \;\big|\; \mathcal{A} \in STR, \varphi \text{ sentence in } \Phi, \mathcal{A} \models \varphi\big\},$$

or more intuitively,

$MC(\Phi)$		
	Input:	$\mathcal{A} \in STR$.
	Parameter:	$\varphi \in \Phi$.
	Question:	Does $\mathcal{A} \models \varphi$?

In this section we collect a few basic facts about parameterized model-checking problems. For every Φ we consider, we assume that we have fixed an encoding $\gamma : \Phi \to \{0,1\}^*$, and we let $||\varphi||$ be the length of $\gamma(\varphi)$.

Taking sentences as parameters seems a little unusual. The following parameterization of the model-checking problem looks more natural:

$MC'(\Phi)$		
	Input:	$\mathcal{A} \in STR$, $\varphi \in \Phi$.
	Parameter:	$\|\|\varphi\|\|$.
	Question:	Does $\mathcal{A} \models \varphi$?

However, it is easy to see that $MC(\Phi) \equiv^{\text{fp}}_{\text{m}} MC'(\Phi)$.

It is well known that various problems of model theory or complexity theory can be reduced from structures to graphs. The following two lemmas contain such reductions. Although their proofs use only standard techniques, they are subtle and require some care. Therefore we decided to give the proofs in some detail. We will apply these lemmas several times later.

LEMMA 3.1. *There are polynomial time transformations that associate with every structure $\mathcal{A} \in STR$ a graph $\mathcal{H}(\mathcal{A})$ and with every sentence $\varphi \in$ FO a sentence $\varphi_{GRAPH} \in$ FO, respectively, such that*

$$\mathcal{A} \models \varphi \iff \mathcal{H}(\mathcal{A}) \models \varphi_{GRAPH}.$$

Furthermore for every $t \geq 1$, if $\varphi \in \Sigma_t$ then $\varphi_{GRAPH} \in \Sigma_{t+1}$ and if $\varphi \in \Pi_t$ then $\varphi_{GRAPH} \in \Pi_{t+1}$.

Proof. Let $\mathcal{A} \in STR[\tau]$ and $\varphi \in \text{FO}[\tau]$. Without loss of generality we can assume that φ is in prenex and in negation normal form.

Step 1. In the first step we translate $\mathcal{A}$ to a structure $\mathcal{B}(\mathcal{A})$ of a vocabulary $\beta(\tau)$ that consists only of unary and binary relation symbols. φ is translated to a corresponding sentence φ_B of vocabulary $\beta(\tau)$.

$\beta(\tau)$ contains unary relation symbols U and U_R for each symbol $R \in \tau$ and binary relation symbols $E_1, \ldots, E_s$, where s is the arity of τ.

The universe of $\mathcal{B}(\mathcal{A})$ is

$$B(\mathcal{A}) := A \cup \{b(R, \bar{a}) \mid R \in \tau, \bar{a} \in R^{\mathcal{A}}\}.$$

We assume that the elements $b(R, \bar{a})$ are all pairwise distinct and distinct from those in A. Note that the cardinality of $B(\mathcal{A})$ is essentially $||\mathcal{A}||$, up to an additive term depending on τ. The unary relations are defined in the obvious way: We let $U^{\mathcal{B}(\mathcal{A})} := A$ and $U_R^{\mathcal{B}(\mathcal{A})} := \{b(R, \bar{a}) \mid \bar{a} \in R^{\mathcal{A}}\}$ for every $R \in \tau$. The binary relations $E_1, \ldots, E_s$ are defined by

$$E_i^{\mathcal{B}(\mathcal{A})} := \{(a_i, b(R, \bar{a})), (b(R, \bar{a}), a_i) \mid R \in \tau, \bar{a} = (a_1, \ldots, a_r) \in R^{\mathcal{A}}, 1 \leq i \leq r\}.$$

Note that $E_i^{\mathcal{B}(\mathcal{A})}$ is symmetric; this will be useful later.

To define φ_B, we first relativize all quantifiers to U, i.e., we inductively replace all subformulas $\exists x\psi$ by $\exists x(Ux \wedge \psi)$ and all subformulas $\forall x\psi$ by $\forall x(Ux \to \psi)$. We obtain a formula φ'.

φ_B is obtained from φ' by replacing every atomic subformula $R\bar{x}$, for r-ary $R \in \tau$, by

$$\exists z \left(U_R z \wedge \bigwedge_{i=1}^{r} E_i x_i z \right), \tag{3.1}$$

where z is a new variable. Then we have

$$\mathcal{A} \models \varphi \iff \mathcal{B}(\mathcal{A}) \models \varphi_B. \tag{3.2}$$

Furthermore, $\mathcal{B}(\mathcal{A})$ can be computed from $\mathcal{A}$ in time $O(||\tau|| \cdot ||\mathcal{A}||)$, where $||\tau||$ denotes the length of the encoding of τ, and φ_B can be computed from φ in linear time.

Step 2. In this step we replace the binary relations $E_1, \dots, E_s$ by a single new binary relation E. We let

$$\gamma(\tau) := (\beta(\tau) \setminus \{E_1, \dots, E_s\}) \cup \{E, P_1, \dots, P_s\},$$

where $P_1, \dots, P_s$ are new unary relation symbols.

We transform $\mathcal{B}(\mathcal{A})$ to a $\gamma(\tau)$-structure $\mathcal{C}(\mathcal{A})$ as follows: For $1 \leq i \leq s$ and for all $a, b \in B(\mathcal{A})$ such that $E_i^{\mathcal{B}(\mathcal{A})}ab$ we introduce a new element $c(i,a,b)$, add the pairs $(a, c(i,a,b))$, $(c(i,a,b), a)$, $(c(i,a,b), b)$, $(b, c(i,a,b))$ to $E^{\mathcal{C}(\mathcal{A})}$, and add $c(i,a,b)$ to $P_i^{\mathcal{C}(\mathcal{A})}$.

We define a sentence φ_C by replacing each subformula of φ_B of the form $E_i xy$ by $\exists z(Exz \wedge Ezy \wedge P_i z)$. Then we have the analogue of (3.2) and the subsequent remarks for $\mathcal{C}(\mathcal{A}), \varphi_C$ instead of $\mathcal{B}(\mathcal{A}), \varphi_B$.

Step 3. The restriction of $\mathcal{C}(\mathcal{A})$ to E is already a graph, i.e., $E^{\mathcal{C}(\mathcal{A})}$ is symmetric and irreflexive. Therefore all we have to do is to eliminate the unary relations $U, (U_R)_{R\in\tau}, (P_i)_{1\leq i\leq s}$. Say, $Q_1, \dots, Q_l$ is an enumeration of all these relations. Note that every $a \in C(\mathcal{A})$ is either isolated or of valence two or adjacent to a vertex of valence two. We use this to define certain trees $\mathcal{T}_1, \dots, \mathcal{T}_l$ and corresponding existential first-order formulas $\xi_1(x), \dots, \xi_l(x)$ and attach a copy of $\mathcal{T}_i$ to each vertex in $Q_i^{\mathcal{C}(\mathcal{A})}$ in such a way that in the resulting graph $\mathcal{H}(\mathcal{A})$ we have for all vertices a

$$\mathcal{H}(\mathcal{A}) \models \xi_i(a) \iff a \in Q_i^{\mathcal{C}(\mathcal{A})}.$$

Furthermore, the $\mathcal{T}_i$ and thus the ξ_i can be chosen of size polynomial in l. We omit the details.

Then we let φ_{GRAPH} be the formula obtained from φ_C by replacing every subformula of the form $Q_i x$ by $\xi_i(x)$, for $1 \leq i \leq l$, and transforming the resulting formula into prenex normal form in the usual manner.

Clearly the transformations $\mathcal{A} \mapsto \mathcal{H}(\mathcal{A})$ and $\varphi \mapsto \varphi_{GRAPH}$ are polynomial, and we have

$$\mathcal{A} \models \varphi \iff \mathcal{H}(\mathcal{A}) \models \varphi_{GRAPH}.$$

It remains to prove that if $\varphi \in \Sigma_t$ ($\varphi \in \Pi_t$), then $\varphi_{GRAPH} \in \Sigma_{t+1}$ ($\varphi_{GRAPH} \in \Pi_{t+1}$, respectively). This follows easily from the way we defined the sentences φ_B, φ_C, and φ_{GRAPH} in Steps 1–3, noting that each positive (negative) occurrence of a relation symbol $R \in \tau$ gives rise only to positive (respectively, negative) occurrences of U_R, the E_i, and the P_i. Thus for the positive occurrences of the $R \in \tau$ we get a new block of existential quantifiers and for the negative occurrences a new block of universal quantifiers. This increases the alternation depth by at most one. □

Recall that for a class C of structures and an $s \geq 0$, by $C[s]$ we denote the class of all structures in C whose vocabulary is at most s-ary. Similarly, for a class Φ of formulas, by $\Phi[s]$ we denote the class of all formulas in Φ whose vocabulary is at most s-ary.

LEMMA 3.2. *Let $s \geq 1$. There is a polynomial time transformation that associates with every structure $\mathcal{A} \in STR[s]$ a graph $\mathcal{H}'(\mathcal{A})$ and a linear time function that associates with every sentence $\varphi \in \mathrm{FO}$ a sentence $\varphi'_{GRAPH} \in \mathrm{FO}$ such that*

$$\mathcal{A} \models \varphi \iff \mathcal{H}'(\mathcal{A}) \models \varphi'_{GRAPH}.$$

Furthermore, for all $t \geq 1$, if $\varphi \in \Sigma_t$, then $\varphi'_{GRAPH} \in \Sigma_t$ and if $\varphi \in \Pi_t$, then $\varphi'_{GRAPH} \in \Pi_t$.

Proof. Let us first look back at the proof of the last lemma and note that if all relation symbols occur only positively in φ, then the transformation $\varphi \mapsto \varphi_{GRAPH}$ generates only a new block of existential quantifiers. If all relation symbols occur only negatively, then we get only a new block of universal quantifiers. Thus if $\varphi \in \Sigma_{2t-1}$ ($\varphi \in \Pi_{2t}$) and all relation symbols occur only positively in φ, then also $\varphi_{GRAPH} \in \Sigma_{2t-1}$ (respectively, $\varphi_{GRAPH} \in \Pi_{2t}$). Similarly, if $\varphi \in \Sigma_{2t}$ ($\varphi \in \Pi_{2t-1}$) and all relation symbols occur only negatively in φ, then also $\varphi_{GRAPH} \in \Sigma_{2t}$ (respectively, $\varphi_{GRAPH} \in \Pi_{2t-1}$).

We first transform $\mathcal{A}$ to an $\mathcal{A}'$ and φ to a φ' in the same prefix class such that

$$\mathcal{A} \models \varphi \iff \mathcal{A}' \models \varphi',$$

and either all relation symbols in φ' occur positively or negatively, whichever we need to apply the previous remark. For this purpose, let τ be an at most s-ary vocabulary. We let $\tau' := \tau \cup \{\overline{R} \mid R \in \tau\}$, where $\overline{R}$ is a new relation symbol that has the same arity as R. For every $\mathcal{A} \in STR[\tau]$, we let $\mathcal{A}'$ be the τ'-expansion of $\mathcal{A}$ with $\overline{R}^{\mathcal{A}'} = A^r \setminus R^{\mathcal{A}}$ for r-ary $R \in \tau$. Note that $\mathcal{A}'$ can be computed from $\mathcal{A}$ in time $O(||\mathcal{A}||^s)$. To define φ' we either replace each negative literal $\neg R\bar{x}$ by $\overline{R}\bar{x}$ or each positive literal $R\bar{x}$ by $\neg\overline{R}\bar{x}$. □

Remark 3.3. Note that if we represent structures by the array representation, then the transformation of Lemma 3.2 is actually polynomial even if we do not fix the arity of the vocabulary in advance. This follows from the fact that the array representation of the structure $\mathcal{A}'$ (in the proof of the lemma) can be computed from the array representation of $\mathcal{A}$ in linear time, uniformly over all vocabularies.

For a parameterized problem $P \subseteq STR \times \Pi^*$ and a class C of structures we let $P|_C$ denote the *restriction* of P to C. In particular,

$$MC(\Phi)|_{GRAPH} = \{(\mathcal{G}, \varphi) \mid \mathcal{G} \in GRAPH, \varphi \in \Phi, \mathcal{G} \models \varphi\}.$$

Note that for every class Φ of formulas and vocabulary τ the two problems $MC(\Phi[\tau])$ and $MC(\Phi)|_{STR[\tau]}$, though formally different, are essentially the same. Therefore we do not distinguish between them.

COROLLARY 3.4.

(1) $MC(\mathrm{FO}) \equiv^{\mathrm{fpp}}_{\mathrm{m}} MC(\mathrm{FO})|_{GRAPH}$.

(2) *For all $t \geq 1$ we have*

$$MC(\Sigma_t) \leq^{\mathrm{fpp}}_{\mathrm{m}} MC(\Sigma_{t+1})|_{GRAPH} \quad \text{and} \quad MC(\Pi_t) \leq^{\mathrm{fpp}}_{\mathrm{m}} MC(\Pi_{t+1})|_{GRAPH}.$$

(3) *For all $s \geq 2$, $t \geq 1$ we have*

$$MC(\Sigma_t[s]) \equiv^{\mathrm{fpp}}_{\mathrm{m}} MC(\Sigma_t)|_{GRAPH} \quad \text{and} \quad MC(\Pi_t[s]) \equiv^{\mathrm{fpp}}_{\mathrm{m}} MC(\Pi_t)|_{GRAPH}.$$

We do not know if $MC(\Sigma_t) \leq^{\mathrm{fp}}_{\mathrm{m}} MC(\Sigma_t)|_{GRAPH}$ or $MC(\Pi_t) \leq^{\mathrm{fp}}_{\mathrm{m}} MC(\Pi_t)|_{GRAPH}$ for any $t \geq 1$.

4. Defining parameterized problems. Definability is the connection between arbitrary parameterized problems and our logical analysis that focuses on model-checking problems. In [11], Downey, Fellows, and Regan consider two forms of definability: Their exposition motivates two general notions of definability, which we call *slicewise definability* and *Fagin definability*.

For a parameterized problem $P \subseteq \Sigma^* \times \Pi^*$ and $y \in \Pi^*$, we call $P \cap (\Sigma^* \times \{y\})$ the yth *slice* of P.

DEFINITION 4.1. *Let $P \subseteq STR \times \Pi^*$ be a parameterized problem and Φ a class of formulas. P is* slicewise Φ-definable *if there is a computable function $\delta : \Pi^* \to \Phi$ such that for all $\mathcal{A} \in STR$ and $y \in \Pi^*$ we have $(\mathcal{A}, y) \in P \iff \mathcal{A} \models \delta(y)$.*

For example, the parameterized subgraph isomorphism problem SI is slicewise Σ_1-definable via the function $\delta : GRAPH \to \Sigma_1$ defined as follows: For a graph $\mathcal{H}$ with vertex set $H = \{h_1, \dots, h_n\}$ of cardinality n, $\delta(\mathcal{H})$ is the sentence

$$\exists x_1 \dots \exists x_n \left(\bigwedge_{1 \le i < j \le n} x_i \neq x_j \wedge \bigwedge_{\substack{1 \le i,j \le n \\ E^{\mathcal{H}} h_i h_j}} E x_i x_j \right).$$

Slicewise Φ-definability is closely related to the model-checking problem for Φ: If $P \subseteq STR \times \Pi^*$ is slicewise Φ-definable, then $P \le^{\mathrm{fp}}_{\mathrm{m}} MC(\Phi)$. On the other hand, if Φ is a decidable set of formulas, then the problem $MC(\Phi)$ is slicewise Φ-definable for trivial reasons.

DEFINITION 4.2. *Let τ be a vocabulary, $P \subseteq STR[\tau] \times \mathbb{N}$ a parameterized problem, and Φ a class of formulas. P is* Φ-Fagin-definable *if there is a relation symbol $X \notin \tau$ (say, r-ary) and a sentence $\varphi \in \Phi[\tau \cup \{X\}]$ such that for all $\mathcal{A} \in STR[\tau]$ and $k \in \mathbb{N}$ we have $(\mathcal{A}, k) \in P$ if and only if there is a $B \subseteq A^r$ such that $|B| = k$ and $(\mathcal{A}, B) \models \varphi$. (Here $(\mathcal{A}, B)$ denotes the $\tau \cup \{X\}$-expansion of $\mathcal{A}$ that interprets X by B.) Then φ* Fagin-defines *P.*

We often consider X as a relation *variable* and thus write $\varphi(X) \in \Phi[\tau]$ instead of $\varphi \in \Phi[\tau \cup \{X\}]$ and $\mathcal{A} \models \varphi(B)$ instead of $(\mathcal{A}, B) \models \varphi$.

For example, parameterized vertex cover VC is Fagin-defined by the formula

$$\varphi_{VC} := \forall y \forall z \big(Eyz \to (Xy \vee Xz) \big).$$

It is easy to see that every problem that is FO-Fagin-definable is also FO-slicewise definable. Indeed, if $P \subseteq STR[\tau] \times \mathbb{N}$ is Fagin-defined by a formula $\varphi(X) \in \mathrm{FO}[\tau]$, where X is r-ary, then it is slicewise FO-defined via the function $\delta : \mathbb{N} \to \mathrm{FO}[\tau]$ with $\delta(k) = \exists \bar{x}_1 \dots \exists \bar{x}_k (\bigwedge_{1 \le i < j \le k} \bar{x}_i \neq \bar{x}_j \wedge \varphi_k)$, where $\bar{x}_1, \dots, \bar{x}_k$ are r-tuples of distinct variables not occurring in φ, and φ_k is the sentence obtained from φ by replacing each subformula of the form $X\bar{y}$ by $\bigvee_{i=1}^k \bar{x}_i = \bar{y}$. Fagin definability implies slicewise definability also for other reasonable classes Φ of formulas, e.g., for the class Σ^1_1 of formulas of second-order logic of the form $\exists X_1 \cdots \exists X_l \psi$, where $X_1, \dots X_l$ are relation variables and ψ is first-order.

The converse is certainly not true, not even for problems of the specific form $P \subseteq STR[\tau] \times \mathbb{N}$: It is obvious that there are slicewise FO-definable problems of arbitrarily high classical complexity (choose δ in Definition 4.1 arbitrarily complex). On the other hand, we have the following characterization of Σ^1_1-Fagin-definable problems.

PROPOSITION 4.3. *Let $P \subseteq STR[\tau] \times \mathbb{N}$. Then,* (1) *and* (2) *are equivalent, where*

(1) *P is Σ^1_1-Fagin-definable;*

(2) *P is in* NP *(when considered as a classical problem) and for some $r \geq 1$, $(\mathcal{A}, k) \in P$ implies $k \leq |A|^r$.*

Proof. The implication of (1) $\Rightarrow$ (2) being clear, we turn to a proof of (2) $\Rightarrow$ (1). Choose r according to (2). Then,

$$\{(\mathcal{A}, B) \mid B \subseteq A^r \text{ and } (\mathcal{A}, |B|) \in P\}$$

is a class of $\tau \cup \{X\}$-structures in NP, where X is r-ary. By Fagin's theorem [14], there is a Σ^1_1-formula $\varphi(X)$ of vocabulary $\tau \cup \{X\}$ axiomatizing this class. Then, $\varphi(X)$ Σ^1_1-Fagin-defines P. □

Thus, slicewise definability is the more general notion. Nevertheless, Fagin definability can be very useful. We illustrate this by the following generalization of a result due to Cai and Chen, namely, Theorem 3.5 of [3], which is based on a result due to Kolaitis and Thakur [23] that syntactically characterizes certain minimization problems. It is motivated by comparing the formula φ_{VC} defining the fixed-parameter tractable problem *VC* with the following formulas φ_{DS} and φ_{CLIQUE} defining the W[1]-hard problems *DS* and *CLIQUE*, respectively:

$$\varphi_{DS} := \forall y \exists x (Xx \wedge (x = y \vee Exy)\big),$$
$$\varphi_{CLIQUE} := \forall y \forall z \big((Xy \wedge Xz) \rightarrow (y = z \vee Eyz)\big).$$

Observe that in φ_{DS} the relation variable X is in the scope of an existential quantifier and in φ_{CLIQUE} it occurs negatively.

THEOREM 4.4. *Let τ be a vocabulary and $P \subseteq STR[\tau] \times \mathbb{N}$ a parameterized problem that is Fagin-defined by a* FO$[\tau]$*-formula $\varphi(X)$ in which X does not occur in the scope of an existential quantifier or negation symbol. Then P is in* FPT.

Proof. For simplicity, let us assume that X is unary. Without loss of generality we can assume that

$$\varphi = \forall y_1 \ldots \forall y_l \bigvee_{i=1}^{m} \bigwedge_{j=1}^{p} \psi_{ij},$$

where each ψ_{ij} is either Xy_q for some $q \in \{1, \ldots, l\}$ or a first-order formula with free variables in $\{y_1, \ldots, y_l\}$ in which X does not occur. In a preprocessing phase we replace the latter ones by atomic formulas: For each such ψ_{ij} we introduce a new relation symbol R_{ij} whose arity matches the number of free variables of ψ_{ij} and let τ^* be the set of all these relation symbols. We let φ^* be the formula obtained from φ by replacing each subformula $\psi_{ij}(\bar{z})$ by $R_{ij}\bar{z}$. Then $\varphi^* = \forall y_1 \ldots \forall y_l \bigvee_{i=1}^{m} \chi_i$, where each χ_i is a conjunction of atomic formulas.

For a structure $\mathcal{A} \in STR[\tau]$ we let $\mathcal{A}^*$ be the τ^*-structure with universe A and with

$$R_{ij}^{\mathcal{A}^*} := \{\bar{a} \mid \mathcal{A} \models \psi_{ij}(\bar{a})\}.$$

Then we have for $B \subseteq A$,

$$\mathcal{A} \models \varphi(B) \iff \mathcal{A}^* \models \varphi^*(B).$$

Given $\mathcal{A}$, each $R_{ij}^{\mathcal{A}^*}$ can be computed in time $O(|A|^{||\psi_{ij}||})$; thus $\mathcal{A}^*$ can certainly be computed in time $O(|A|^{||\varphi||})$.

```
CHECK-φ(A ∈ STR[τ], k ∈ ℕ)
 1  compute A*
 2  initialize set S ⊆ Pow(A) by S := {∅}
 3  for all ā ∈ A^l do
 4      for all B ∈ S do
 5          if A* ⊭ ⋁_{i=1}^{m} χ_i(B, ā) then
 6              S := S \ {B}
 7              for i = 1 to m do
 8                  compute B' := β(B, ā, i)
 9                  if |B'| ≤ k and A* ⊨ χ_i(B', ā)
10                      then S := S ∪ {B'}
11  if S ≠ ∅
12      then accept
13      else reject.
```

Algorithm 1.

For $1 \leq i \leq m$, $\bar{a} = a_1 \dots a_l \in A^l$, and $B \subseteq A$ we let

$$\beta(B, \bar{a}, i) := B \cup \{a_j \mid Xy_j \text{ is a conjunct of } \chi_i\}.$$

Since $\chi_i(X, \bar{y})$ is positive in X, the following two statements are equivalent for every B' with $B \subseteq B' \subseteq A$:

- $\mathcal{A}^* \models \chi_i(B', \bar{a})$.
- $\beta(B, \bar{a}, i) \subseteq B'$ and $\mathcal{A}^* \models \chi_i(\beta(B, \bar{a}, i), \bar{a})$.

This equivalence is used by Algorithm 1 to decide P.

Recall that, given a τ-structure $\mathcal{A}$ and a parameter $k \in \mathbb{N}$, the algorithm is supposed to decide whether there is a $B \subseteq A$ with $|B| = k$ such that for all $\bar{a} \in A^l$ there is an i such that $\mathcal{A}^* \models \chi_i(B, \bar{a})$.

The crucial observation to see that the algorithm is correct is that whenever the main loop in lines 3–10 is entered, $\mathcal{S}$ is a set of subsets $B \subseteq A$ such that $|B| \leq k$ and for all $\bar{a}$ considered so far (in earlier runs through the loop) we have $\mathcal{A}^* \models \bigvee_{i=1}^{m} \chi_i(B, \bar{a})$.

To get a bound on the running time, we note that whenever a new set is added to $\mathcal{S}$ (in line 10), then it is an extension of a strictly smaller set that has been removed from $\mathcal{S}$ (in line 6). Furthermore, for each set removed (in line 6) at most m such extensions can be added. Thus an upper bound for the number of sets that can be in $\mathcal{S}$ at any time is m^k. The main loop (in lines 3–10) is called n^l times, where $n := |A|$. This gives an overall bound on the running time of $O(m^k \cdot n^l)$ plus the time needed to compute $\mathcal{A}^*$.

Since l does not depend on the instance, but only on the formula φ, this yields the fixed-parameter tractability of P. □

Besides *VC*, many other parameterized problems can be shown to be fixed-parameter tractable by a simple application of this theorem. Let us consider one example in detail: The *valence* of a graph is the maximal number of neighbors a vertex in the graph has. For an $l \geq 1$, we consider the restriction of *DOMINATING SET* to graphs of valence at most l, i.e., the following problem:

VC_l		
	Input:	Graph $\mathcal{G}$.
	Parameter:	$k \in \mathbb{N}$.
	Question:	Is the valence of $\mathcal{G}$ at most l and does $\mathcal{G}$ have a dominating set of size at most k?

This problem is Fagin-defined by the following first-order formula:

$$\forall x \exists^{\leq l} z\, Exz \wedge \forall y_0 \forall y_1 \ldots \forall y_l \left(\forall z \left(Ey_0 z \rightarrow \bigvee_{i=1}^{l} z = y_i \right) \rightarrow \bigvee_{i=0}^{l} Xy_i \right).$$

($\exists^{\leq m} x \psi(x)$ abbreviates $\exists y_1 \ldots \exists y_m \forall x (\psi(x) \rightarrow \bigvee_{i=1}^{m} x = y_i)$.)

Other examples of problems that can be shown to be in FPT by Theorem 4.4 are *HITTING SET FOR SIZE THREE SETS*, *MATRIX DOMINATION*, or *SHORT 3DIMENSIONAL MATCHING*. These problems are also considered in [10].

5. Homomorphisms, embeddings, and model-checking. In this section we analyze the close relationship between the homomorphism problem, the embedding problem, and model-checking problems for Σ_1-formulas from the point of view of parameterized complexity (compare with [24] for a further analysis of this relationship).

A *homomorphism* from a τ-structure $\mathcal{B}$ into a τ-structure $\mathcal{A}$ is a mapping $h : B \rightarrow A$ such that for all $R \in \tau$ and tuples $\bar{b} \in R^{\mathcal{B}}$ we have $h(\bar{b}) \in R^{\mathcal{A}}$. The parameterized *HOMOMORPHISM PROBLEM (HOM)* is defined as follows:

HOM		
	Input:	$\mathcal{A} \in STR$.
	Parameter:	$\mathcal{B} \in STR$.
	Question:	Is there a homomorphism from $\mathcal{B}$ to $\mathcal{A}$?

A *(weak) embedding* of $\mathcal{B}$ into $\mathcal{A}$ is an injective homomorphism from $\mathcal{B}$ to $\mathcal{A}$. Note that a graph $\mathcal{H}$ is isomorphic to a subgraph of a graph $\mathcal{G}$ in the usual graph theoretic sense if there is an embedding of $\mathcal{H}$ into $\mathcal{G}$. Thus the following parameterized *EMBEDDING PROBLEM (EMB)* is a generalization of the *SI* problem:

EMB		
	Input:	$\mathcal{A} \in STR$.
	Parameter:	$\mathcal{B} \in STR$.
	Question:	Is there an embedding of $\mathcal{B}$ into $\mathcal{A}$?

The *Gaifman graph* of a τ-structure $\mathcal{A}$ is the graph $\mathcal{G}(\mathcal{A})$ with universe A in which two elements $a \neq b$ are adjacent if there is an $R \in \tau$ and a tuple $\bar{a} \in R^{\mathcal{A}}$ such that both a and b occur in the tuple $\bar{a}$. For a class C of graphs we let $STR[C]$ denote the class of all structures whose Gaifman graph is in C. Note that $STR[GRAPH] = STR$. Furthermore, we let $STR[\tau, C] := STR[\tau] \cap STR[C]$ for every vocabulary τ and $STR[s, C] := STR[s] \cap STR[C]$ for every $s \geq 1$. We define restrictions $HOM[\ldots]$ and $EMB[\ldots]$ of the respective problems, where for every possible restriction "..." in the square brackets we require the *parameter* $\mathcal{B}$ to belong to the class $STR[\ldots]$. For example, we let

$$EMB[s, C] := EMB \cap (STR \times STR[s, C]).$$

LEMMA 5.1. *For all classes C of graphs and $s \geq 1$ we have $HOM[C] \leq_{\mathrm{m}}^{\mathrm{fpp}} EMB[C]$ and $HOM[s,C] \leq_{\mathrm{m}}^{\mathrm{fpp}} EMB[s,C]$.*

Proof. Suppose we are given an instance $(\mathcal{A},\mathcal{B})$ of $HOM[C]$. Let τ be the vocabulary of $\mathcal{A}$. Let $\mathcal{A}_B$ be the τ-structure which for every element of $\mathcal{A}$ contains $|B|$ duplicates, i.e., $A_B := A \times B$ and, for every r-ary $R \in \tau$,

$$R^{\mathcal{A}_B} := \big\{((a_1,b_1),\dots,(a_r,b_r)) \;\big|\; R^{\mathcal{A}} a_1 \dots a_r\big\}.$$

Then, every homomorphism $h : \mathcal{B} \to \mathcal{A}$ gives rise to an embedding $f : \mathcal{B} \to \mathcal{A}_B$ defined by $f(b) = (h(b), b)$ and every embedding $f : \mathcal{B} \to \mathcal{A}_B$ induces a homomorphism $h : \mathcal{B} \to \mathcal{A}$ defined by letting $h(b)$ be the projection on the first component of $f(b)$. □

Note that, unless PTIME = NP, there is no polynomial time reduction from $EMB[C]$ to $HOM[C]$ for the class C of all paths because $HOM[C]$ can easily be seen to be in PTIME by a dynamic programming algorithm, whereas $EMB[2,C]$ is NP-complete by a reduction from *HAMILTONIAN PATH*. Thus $EMB[s,C] \leq_{\mathrm{m}}^{\mathrm{fpp}} HOM[s,C]$ does not hold for any $s \geq 2$. However, we will see that for all $s \geq 2$ and C we actually have $HOM[s,C] \equiv_{\mathrm{T}}^{\mathrm{fp}} EMB[s,C]$.

Before we show this, we introduce two related model-checking problems. With each first-order formula φ we associate a graph $\mathcal{G}(\varphi)$. Its universe is $\mathrm{var}(\varphi)$, the set of all variables in φ, and there is an edge between distinct $x, y \in \mathrm{var}(\varphi)$ in $\mathcal{G}(\varphi)$ if φ has an atomic subformula in which both x, y occur.

Let $\varphi^{\neq}$ be the formula obtained from φ by deleting all inequalities, i.e., all atomic subformulas of the form $x = y$ that occur in the scope of an odd number of negation symbols. We are also interested in $\mathcal{G}(\varphi^{\neq})$. Let us see an example:

$$\varphi := \exists x_1 \dots \exists x_k \left(\bigwedge_{1 \leq i < j \leq k} \neg x_i = x_j \wedge \bigwedge_{i=1}^{k-1} E x_i x_{i+1} \right).$$

$\mathcal{G}(\varphi)$ is the complete graph with vertex set $\{x_1,\dots,x_k\}$, whereas $\mathcal{G}(\varphi^{\neq})$ is the path $x_1 \dots x_k$. Note that φ says that a graph has a subgraph isomorphic to a path of length k, whereas $\varphi^{\neq}$ says that a graph contains a homomorphic image of a path of length k. This generalizes to the following simple lemma, whose proof we omit.

LEMMA 5.2. *For every structure $\mathcal{B} \in STR$ there is a Σ_1-sentence $\varphi_{\mathcal{B}}$ (whose quantifier-free part is a conjunction of literals) such that $\mathcal{G}(\varphi_{\mathcal{B}}^{\neq}) = \mathcal{G}(\mathcal{B})$ and for every structure $\mathcal{A}$ we have*

$$\mathcal{A} \models \varphi_{\mathcal{B}} \iff \text{there is an embedding of } \mathcal{B} \text{ into } \mathcal{A},$$
$$\mathcal{A} \models \varphi_{\mathcal{B}}^{\neq} \iff \text{there is a homomorphism from } \mathcal{B} \text{ to } \mathcal{A}.$$

Furthermore, the mapping $\mathcal{B} \mapsto \varphi_{\mathcal{B}}$ is computable in linear time.

For $s \in \mathbb{N}$ and a class C of graphs we let

$$\Sigma_1[s,C] := \big\{\varphi \in \Sigma_1[\tau] \;\big|\; \tau \text{ } s\text{-ary vocabulary}, \mathcal{G}(\varphi) \in C\big\},$$
$$\Sigma_1^{\neq}[s,C] := \big\{\varphi \in \Sigma_1[\tau] \;\big|\; \tau \text{ } s\text{-ary vocabulary}, \mathcal{G}(\varphi^{\neq}) \in C\big\}.$$

Furthermore, we let $\Sigma_1[C] := \bigcup_{s \geq 1} \Sigma_1[s,C]$ and $\Sigma_1^{\neq}[C] := \bigcup_{s \geq 1} \Sigma_1^{\neq}[s,C]$.

Lemma 5.2 implies that for every $s \geq 1$ and for every class C of graphs we have $HOM[s,C] \leq_{\mathrm{m}}^{\mathrm{fpp}} MC(\Sigma_1[s,C])$ and $EMB[s,C] \leq_{\mathrm{m}}^{\mathrm{fpp}} MC(\Sigma_1^{\neq}[s,C])$. Unless PTIME =

NP, the converse of these statements is wrong. To see this, let C be the class of all graphs that consist only of isolated vertices, i.e., all graphs $\mathcal{G}$ with $E^{\mathcal{G}} = \emptyset$. Then clearly $HOM[C]$ and $EMB[C]$ are in PTIME, but we can reduce the satisfiability problem for propositional formulas to $MC(\Sigma_1[1, C])$ and $MC(\Sigma_1^{\neq}[1, C])$.

THEOREM 5.3. *Let C be a class of graphs and $s \geq 2$. Then*

$$HOM[s, C] \equiv_{\mathrm{T}}^{\mathrm{fp}} EMB[s, C] \equiv_{\mathrm{T}}^{\mathrm{fp}} MC(\Sigma_1^{\neq}[s, C]) \equiv_{\mathrm{T}}^{\mathrm{fp}} MC(\Sigma_1[s, C]).$$

Proof. We have already seen that $HOM[s, C] \leq_{\mathrm{T}}^{\mathrm{fp}} EMB[s, C]$ (Lemma 5.1) and that $EMB[s, C] \leq_{\mathrm{T}}^{\mathrm{fp}} MC(\Sigma_1^{\neq}[s, C])$ (Lemma 5.2). To complete the cycle we shall prove that $MC(\Sigma_1^{\neq}[s, C]) \leq_{\mathrm{T}}^{\mathrm{fp}} MC(\Sigma_1[s, C])$ and that $MC(\Sigma_1[s, C]) \leq_{\mathrm{T}}^{\mathrm{fp}} HOM[s, C]$.

We first prove that $MC(\Sigma_1[s, C]) \leq_{\mathrm{T}}^{\mathrm{fp}} HOM[s, C]$. Let $\varphi \in \Sigma_1[s, C]$, say, of vocabulary τ, and let $\mathcal{A}$ be a τ-structure. We shall describe an algorithm that decides whether $\mathcal{A} \models \varphi$ using $HOM[s, C]$ as an oracle.

Let S be a binary relation symbol not contained in τ and $\tau' := \tau \cup \{S\}$. Furthermore, let $\mathcal{A}'$ be the τ'-expansion of $\mathcal{A}$ with $S^{\mathcal{A}'} = \emptyset$. Our algorithm first computes a sentence $\varphi' := \bigvee_{i=1}^{m} \exists \bar{x}_i \psi_i$ of vocabulary τ' such that

(1) $\mathcal{A} \models \varphi$ if and only if $\mathcal{A}' \models \varphi'$;
(2) for $1 \leq i \leq m$, the formula ψ_i is a conjunction of literals, and we have $\mathcal{G}(\psi_i) = \mathcal{G}(\varphi)$.

This can be achieved by first translating φ to a sentence whose quantifier-free part is in disjunctive normal form, then swapping existential quantifiers and the disjunction, and then adding dummy literals of the form $\neg Sxy$ until $\mathcal{G}(\psi_i) = \mathcal{G}(\varphi)$.

Let $\tau'' := \tau' \cup \{\overline{R} \mid R \in \tau'\} \cup \{E, \overline{E}\}$, where for all $R \in \tau'$ the symbol $\overline{R}$ is a new relation symbol of the same arity as R, and $E, \overline{E}$ are new binary relation symbols. Let $\mathcal{A}''$ be the τ'' expansion of $\mathcal{A}'$ in which $\overline{R}$ is interpreted as the complement of $R^{\mathcal{A}''}$ and $E, \overline{E}$ are interpreted as equality and inequality, respectively. For $1 \leq i \leq m$, we define a τ''-structure $\mathcal{B}_i$ with $\mathcal{G}(\mathcal{B}_i) = \mathcal{G}(\varphi)$ such that $\mathcal{A}' \models \exists \bar{x}_i \psi_i$ if and only if there is a homomorphism from $\mathcal{B}_i$ into $\mathcal{A}''$. We let $\mathcal{B}_i$ be the τ''-structure with universe $\mathrm{var}(\psi_i)$ and

$$\begin{aligned}
R^{\mathcal{B}_i} &:= \{\overline{y} \mid R\overline{y} \text{ is a literal of } \psi_i\} \qquad (\text{for } R \in \tau'),\\
\overline{R}^{\mathcal{B}_i} &:= \{\overline{y} \mid \neg R\overline{y} \text{ is a literal of } \psi_i\} \qquad (\text{for } R \in \tau'),\\
E^{\mathcal{B}_i} &:= \{yz \mid y = z \text{ is a literal of } \psi_i\},\\
\overline{E}^{\mathcal{B}_i} &:= \{yz \mid \neg y = z \text{ is a literal of } \psi_i\}.
\end{aligned}$$

It is obvious that $\mathcal{B}_i$ does indeed have the desired property. Altogether, our construction yields a parameterized T-reduction.

It remains to prove that $MC(\Sigma_1^{\neq}[s, C]) \leq_{\mathrm{T}}^{\mathrm{fp}} MC(\Sigma_1[s, C])$. We use the so-called *color coding* technique of Alon, Yuster, and Zwick [2].

Let $l \geq 1$ and let X be a set. An *l-perfect family of hash functions on X* is a family F of functions $f : X \to \{1, \ldots, l\}$ such that for all subsets $Y \subseteq X$ of size l there is an $f \in F$ such that $f(Y) = \{1, \ldots, l\}$ (i.e., on Y, f is one-to-one). Alon, Yuster, and Zwick [2] show that given $n, l \geq 1$, an l-perfect family of hash functions on $\{1, \ldots, n\}$ of size $2^{O(l)} \cdot \log n$ can be computed in time $2^{O(l)} \cdot n \cdot \log n$.

For a similar reason as outlined above, without loss of generality we can restrict our attention to sentences $\varphi \in MC(\Sigma_1^{\neq}[s, C])$ whose quantifier-free part is a conjunction of literals. Given such a sentence $\varphi = \exists x_1 \ldots \exists x_k \psi$, say, of vocabulary τ and

a τ-structure $\mathcal{A}$, we define a family of sentences $\varphi_\gamma \in MC(\Sigma_1[s,C])$ and a family of structures $\mathcal{A}_f$ such that $\mathcal{A} \models \varphi$ if and only if there is a γ and f such that $\mathcal{A}_f \models \varphi_\gamma$.

A *coloring of* φ is a function $\gamma : \{1,\dots,k\} \to \{1,\dots,k\}$ such that $\gamma(i) \neq \gamma(j)$ if $\neg x_i = x_j$ occurs in φ. For a coloring γ we let ψ_γ be the formula obtained from ψ by replacing all literals $\neg x_i = x_j$ by $C_{\gamma(i)}x_i \wedge C_{\gamma(j)}x_j$ (here, $C_1,\dots,C_k$ are new unary "color" relation symbols) and let $\varphi_\gamma := \exists x_1 \dots \exists x_k \psi_\gamma$. Note that $\mathcal{G}(\varphi_\gamma) = \mathcal{G}(\varphi^{\neq})$. Thus $\varphi_\gamma \in MC(\Sigma_1[s,C])$.

With every $f : A \to \{1,\dots,k\}$, which we call a *coloring of* $\mathcal{A}$, we let $\mathcal{A}_f$ be the $\tau \cup \{C_1,\dots,C_k\}$-expansion of $\mathcal{A}$ with $C_i^{\mathcal{A}_f} := f^{-1}(i)$ for $1 \leq i \leq k$.

Observe that

$$\mathcal{A} \models \varphi \iff \text{there is a coloring } \gamma \text{ of } \varphi \text{ and a coloring } f \text{ of } \mathcal{A} \text{ such that } \mathcal{A}_f \models \varphi_\gamma. \tag{5.1}$$

The problem is that there are $k^{|A|}$ colorings of $\mathcal{A}$, so (5.1) does not yet give rise to a parameterized reduction. The crucial trick is that to achieve this equivalence we do not have to consider all possible colorings f of $\mathcal{A}$. For $1 \leq l \leq k$, let F_l be an l-perfect family of hash-function on A and $F := \bigcup_{l=1}^{k} F_l$. We claim that

$$\mathcal{A} \models \varphi \iff \text{there is a coloring } \gamma \text{ of } \varphi \text{ and an } f \in F \text{ such that } \mathcal{A}_f \models \varphi_\gamma. \tag{5.2}$$

The backward direction follows immediately from (5.1). For the forward direction, suppose that $\mathcal{A} \models \varphi$. Let $\bar{a} \in A^k$ such that $\mathcal{A} \models \psi(\bar{a})$. There is a function $f \in F$ whose restriction to $\{a_1,\dots,a_k\}$ is one-to-one. Define γ by $\gamma(i) := f(a_i)$. Then, γ is a coloring of φ, $\mathcal{A}_f \models \psi_\gamma(\bar{a})$, and hence $\mathcal{A}_f \models \varphi_\gamma$.

Since the family F can be chosen sufficiently small and computed sufficiently fast, the equivalence (5.2) gives rise to a parameterized reduction. □

Remark 5.4. We do not know if the parameterized T-reductions in Theorem 5.3 can be replaced by parameterized m-reductions. However, for many interesting classes C they can be replaced. One such example is the class of all graphs. Similar techniques work for all classes C of graphs for which there exists an algorithm that, given a graph $\mathcal{H} \in C$, computes a connected $\mathcal{H}' \in C$ such that $\mathcal{H}$ is a subgraph of $\mathcal{H}'$. For all such classes C we can show that

$$HOM[s,C] \equiv_{\mathrm{m}}^{\mathrm{fp}} EMB[s,C] \equiv_{\mathrm{m}}^{\mathrm{fp}} MC(\Sigma_1^{\neq}[s,C]) \equiv_{\mathrm{m}}^{\mathrm{fp}} MC(\Sigma_1[s,C])$$

(for all $s \geq 1$).

5.1. Sentences of bounded tree-width. Our main application of Theorem 5.3 is to sentences whose underlying graphs have bounded tree-width. In the time since we submitted this article, considerable progress has been made in this area. For an extensive discussion of model-checking algorithms based on tree-decompositions of the sentences, we refer the reader to [16].

We think of a *tree* $\mathcal{T}$ as directed from its root, which we denote by $r^{\mathcal{T}}$, to the leaves and thus can speak of a *child* and of the *parent* of a vertex.

A *tree-decomposition* of a τ-structure $\mathcal{A}$ is a pair $(\mathcal{T}, (A_t)_{t\in T})$, where $\mathcal{T}$ is a tree and $(A_t)_{t\in T}$ a family of subsets of A such that

(1) for every $a \in A$, the set $\{t \in T \mid a \in A_t\}$ is nonempty and induces a subtree of $\mathcal{T}$ (that is, is connected);
(2) for every k-ary relation symbol $R \in \tau$ and all $a_1,\dots,a_k \in A$ such that $R^{\mathcal{A}}a_1,\dots,a_k$ there exists a $t \in T$ such that $a_1,\dots,a_k \in A_t$.

The *width* of a tree-decomposition $(\mathcal{T}, (A_t)_{t\in T})$ is $\max\{|A_t| \mid t \in T\} - 1$. The *tree-width* $\mathrm{tw}(\mathcal{A})$ of $\mathcal{A}$ is the minimal width of a tree-decomposition of $\mathcal{A}$.

For $s \geq 1$, let W_s denote the class of all structures of tree-width at most s and $GW_s := GRAPH \cap W_s$. Note that $W_s \subseteq STR[s+1]$ because a graph of tree-width s has clique number at most $s+1$.[2] (The clique number of a graph $\mathcal{G}$ is the maximal cardinality of a set of pairwise adjacent vertices of $\mathcal{G}$.)

Plehn and Voigt [27] were the first to realize that tree-width is a relevant parameter for the problems considered in the previous section. They proved that the parameterized embedding problem restricted to (parameter) graphs of bounded tree-width is fixed-parameter tractable. Chekuri and Rajaraman [6] proved that for every $s \geq 1$ the problem $HOM[GW_s]$ is in PTIME (when considered as an unparameterized problem) and therefore fixed-parameter tractable. They phrased their result in terms of the equivalent *conjunctive query containment* problem (see also [24]).

Thus as a corollary of Theorem 5.3 we obtain the following corollary.

COROLLARY 5.5. *Let $s \geq 1$. Then the problems $EMB[GW_s]$, $MC(\Sigma_1[GW_s])$, and $MC(\Sigma_1^{\neq}[GW_s])$ are in* FPT.

Papadimitriou and Yannakakis [26] proved the model-checking results of this corollary for the related case of acyclic conjunctive queries.

Unfortunately, it turns out that Corollary 5.5 is the only real application of Theorem 5.3. Very recently, Grohe, Schwentick, and Segoufin [20] have proved that for every class C of graphs of unbounded tree-width and every $s \geq 2$, the problem $HOM[C, s]$ is W[1]-complete under parameterized T-reductions.

6. FO-model-checking on graphs with excluded minors. The fixed-parameter tractability results of the previous section were obtained by putting syntactical restrictions on the sentences, i.e., the *parameter* of the model-checking problem. In this section we put restrictions on the structures, i.e., the *input* of the model-checking problem.

Recall the definition of the parameterized problem $MC(\Phi)|_D$ for a class Φ of formulas and a class D of structures:

$MC(\Phi)\|_D$		
	Input:	$\mathcal{A} \in STR$.
	Parameter:	$\varphi \in \Phi$.
	Question:	Is $\mathcal{A} \in D$ and $\mathcal{A} \models \varphi$?

Our starting point is the following theorem due to Courcelle [7]. Remember that *monadic second-order logic* is the extension of first-order logic where one is allowed to quantify not only over individual elements of a structure but also over sets of elements. MSO denotes the class of all formulas of monadic second-order logic. Remember that W_s denotes the class of all structures of tree-width at most s (for $s \geq 0$).

THEOREM 6.1 (see [7]). *Let $s \geq 0$. Then $MC(\mathrm{MSO})|_{W_s}$ is in* FPT.

A graph $\mathcal{H}$ is a *minor* of a graph $\mathcal{G}$ (we write $\mathcal{H} \preceq \mathcal{G}$) if $\mathcal{H}$ can be obtained from a subgraph of $\mathcal{G}$ by contracting edges. $\mathcal{H}$ is an *excluded minor* for a class D if $\mathcal{H}$ is not a minor of any graph in D. Note that a class D of graphs has an excluded minor if and only if there is an $n \in \mathbb{N}$ such that $\mathcal{K}_n$ is an excluded minor for D.

[2] This is slightly imprecise because a structure $\mathcal{A} \in W_s$ might have a vocabulary of arbitrarily high arity, as long as no tuple contained in a relation of $\mathcal{A}$ consists of more than $s+1$ distinct elements. However, since any structure with this property can easily be transformed to an $(s+1)$-ary structure that is essentially the same, we decided to accept this imprecision in exchange for a simpler notation.

Examples of classes of graphs with an excluded minor are classes of graphs of bounded tree-width or classes of graphs embeddable in a fixed surface.

Recall that for a class D of graphs, $STR[D]$ denotes the class of all structures whose Gaifman graph is in C.

THEOREM 6.2. *Let D be a* PTIME*-decidable class of graphs with an excluded minor. Then $MC(\mathrm{FO})|_{STR[D]}$ is in* FPT.

The rest of this section is devoted to the proof of this theorem, which needs some preparation.

A class D of graphs is called *minor closed* if for all $\mathcal{G} \in D$ and $\mathcal{H} \preceq \mathcal{G}$ we have $\mathcal{H} \in D$. Robertson and Seymour proved the following theorem.

THEOREM 6.3 (see [28]). *Every minor-closed class of graphs is* PTIME*-decidable.*

This, together with Theorem 6.2, immediately yields the following corollary.

COROLLARY 6.4. *Let $D \subsetneq GRAPH$ be minor closed. Then $MC(\mathrm{FO})|_{STR[D]}$ is in* FPT.

Recall the definition of the Gaifman graph $\mathcal{G}(\mathcal{A})$ of a structure $\mathcal{A}$ (cf. p. 127). The distance $d^{\mathcal{A}}(a,b)$ between $a, b \in A$ is the length of the shortest path from a to b in $\mathcal{G}(\mathcal{A})$. For $r \in \mathbb{N}$ and $a \in A$, the *r-ball* around a is the set $B_r^{\mathcal{A}}(a) := \{b \in A \mid d^{\mathcal{A}}(a,b) \leq r\}$. For an $X \subseteq A$, $\langle X \rangle^{\mathcal{A}}$ denotes the substructure induced by $\mathcal{A}$ on X, i.e., the structure with universe X and $R^{\langle X \rangle^{\mathcal{A}}} = R^{\mathcal{A}} \cap X^r$ for all r-ary relation symbols R in the vocabulary of $\mathcal{A}$. Furthermore, we let $\mathcal{A} \setminus X := \langle A \setminus X \rangle^{\mathcal{A}}$.

The *local tree-width* of $\mathcal{A}$ is the function $\mathrm{ltw}(\mathcal{A}) : \mathbb{N} \to \mathbb{N}$ defined by

$$\mathrm{ltw}(\mathcal{A})(r) := \max\big\{\mathrm{tw}(\langle B_r^{\mathcal{A}}(a)\rangle^{\mathcal{A}}) \;\big|\; a \in A\big\}.$$

For functions $f, g : \mathbb{N} \to \mathbb{N}$ we write $f \leq g$ if $f(n) \leq g(n)$ for all $n \in \mathbb{N}$. A class D of structures has *bounded local tree-width* if there is a function $\lambda : \mathbb{N} \to \mathbb{N}$ such that for all $\mathcal{A} \in D$ we have $\mathrm{ltw}(\mathcal{A}) \leq \lambda$.

The "local" character of first-order formulas allows us to generalize Theorem 6.1 for first-order logic from classes of structures of bounded tree-width to classes of structures of bounded local tree-width.

THEOREM 6.5 (see [17]). *Let D be a* PTIME*-decidable class of structures of bounded local tree-width. Then $MC(\mathrm{FO})|_D$ is in* FPT.

For $\lambda : \mathbb{N} \to \mathbb{N}$ we let

$$GL(\lambda) := \big\{\mathcal{G} \in GRAPH \;\big|\; \forall \mathcal{H} \preceq \mathcal{G} :\ \mathrm{ltw}(\mathcal{H}) \leq \lambda\big\},$$

and, for $\mu \in \mathbb{N}$,

$$B(\lambda,\mu) := \big\{\mathcal{A} \in STR \;\big|\; \exists X \subseteq A\ (|X| \leq \mu \wedge \mathcal{G}(\mathcal{A} \setminus X) \in GL(\lambda))\big\}.$$

Note that the clique-number of a graph in $GL(\lambda)$ is at most $\lambda(1)+1$; thus the clique-number of a graph in $B(\lambda,\mu)$ is at most $\mu + \lambda(1) + 1$. This implies that $B(\lambda,\mu) \subseteq STR[\mu + \lambda(1) + 1]$.[3]

LEMMA 6.6. *Let $\lambda : \mathbb{N} \to \mathbb{N}$ and $\mu \in \mathbb{N}$. Then $MC(\mathrm{FO})|_{B(\lambda,\mu)}$ is in* FPT.

Proof. The class $GL(\lambda)$ of graphs is minor closed and hence PTIME-decidable by Theorem 6.3. This implies that $B(\lambda,\mu)$ is PTIME-decidable.

Then for $\mu = 0$ the statement follows from Theorem 6.5. The case $\mu > 0$ can be reduced to the case $\mu = 0$ as follows: For every $\mathcal{A} \in \mathcal{B}(\lambda,\mu)$ and sentence $\varphi \in \mathrm{FO}$, we define a structure $\mathcal{A}^* \in B(\lambda,0)$ and a sentence $\varphi^* \in \mathrm{FO}$ in such a way that $A \models \varphi$

[3] Cf. footnote 2.

if and only if $\mathcal{A}^* \models \varphi^*$, and the mappings $\mathcal{A} \mapsto \mathcal{A}^*$ and $\varphi \mapsto \varphi^*$ are computable in polynomial time.

Therefore, suppose we are given $\mathcal{A} \in \mathcal{B}(\lambda, \mu)$ and $\varphi \in$ FO. For simplicity, we assume that their vocabulary consists of a single binary relation symbol E.

Let $X \subseteq A$ such that $|X| \leq \mu$ and $\mathcal{A} \setminus X \in B(\lambda, 0)$. Such an X can be computed in polynomial time because the class $B(\lambda, 0)$ is PTIME-decidable. Say, $X = \{a_1, \ldots, a_\mu\}$.

Let $\tau := \{E, P_1, \ldots, P_\mu, Q_1, \ldots, Q_\mu, R_1, \ldots, R_\mu\}$, where the P_i, Q_i, and R_i are unary. We let $A^* := A$, $E^{\mathcal{A}^*} := E^{\mathcal{A}} \cap (A \setminus X)^2$, and, for $1 \leq i \leq \mu$,

$$\begin{aligned} P_i^{\mathcal{A}^*} &:= \{a_i\}, \\ Q_i^{\mathcal{A}^*} &:= \{b \in A \mid E^{\mathcal{A}} a_i b\}, \\ R_i^{\mathcal{A}^*} &:= \{b \in A \mid E^{\mathcal{A}} b a_i\}. \end{aligned}$$

Furthermore, we let φ^* be the sentence obtained from φ by replacing each subformula Exy by

$$Exy \vee \bigvee_{i=1}^{\mu} ((P_i x \wedge Q_i y) \vee (P_i y \wedge R_i x)).$$

Clearly, these definitions lead to the desired result. □

To complete the proof of Theorem 6.2 we use a decomposition theorem for non-trivial minor-closed classes of graphs that roughly says that all graphs in such a class are built up in a tree-like manner from graphs in a $B(\lambda, \mu)$. It is based on Robertson and Seymour's deep structure theory for graphs without $\mathcal{K}_n$-minors. The precise statement requires some new notation. Let $(\mathcal{T}, (A_t)_{t \in T})$ be a tree-decomposition of a structure $\mathcal{A}$. The *torso* of this decomposition at $t \in \mathcal{T}$, denoted by $[A_t]$, is the graph with universe A_t and an edge between two distinct vertices $a, b \in A_t$ if either there is an edge between a and b in the Gaifman graph $\mathcal{G}(\mathcal{A})$ or there exists an $s \in T \setminus \{t\}$ such that $a, b \in A_s$. $(\mathcal{T}, (A_t)_{t \in T})$ is a tree-decomposition *over* a class D of graphs if all its torsos belong to D.

THEOREM 6.7 (see [19]). *Let D be a class of graphs with an excluded minor. Then there exist $\lambda : \mathbb{N} \to \mathbb{N}$ and $\mu \in \mathbb{N}$ such that every $\mathcal{G} \in D$ has a tree-decomposition over $B(\lambda, \mu)$.*

Furthermore, given $\mathcal{G}$, such a decomposition can be computed in PTIME.

Clearly, this theorem implies the analogous statement for all structures in $STR[D]$.

The *adhesion* of a tree-decomposition $(\mathcal{T}, (A_t)_{t \in T})$ is $\max\{|A_s \cap A_t| \mid E^{\mathcal{T}} st\}$. The *clique number* of a class of structures is the maximum of the clique numbers of the Gaifman graphs of structures in D, if this maximum exists, or ∞ otherwise. Note that if $(\mathcal{T}, (A_t)_{t \in T})$ is a decomposition over a class D, then the clique-number of D is an upper bound for the adhesion of $(\mathcal{T}, (A_t)_{t \in T})$. Remembering that the clique-number of $B(\lambda, \mu)$ is $\lambda(1) + \mu + 1$, we see that the adhesion of a tree-decomposition over $B(\lambda, \mu)$ is at most $\lambda(1) + \mu + 1$.

Proof of Theorem 6.2. Let D be a PTIME-decidable class of graphs with an excluded minor and λ, μ such that every $\mathcal{G} \in D$ has a tree-decomposition over $B(\lambda, \mu)$. Let $\nu := \lambda(1) + \mu + 1$.

We shall describe an algorithm that, given $\mathcal{A} \in STR[D]$ and $\varphi \in$ FO, decides if $A \models \varphi$.

Therefore let $\mathcal{A} \in STR[D]$, say, of vocabulary τ and $\varphi \in \mathrm{FO}[\tau]$. Our algorithm starts by computing a tree-decomposition $(\mathcal{T}, (A_t)_{t \in T})$ of $\mathcal{A}$ over $B(\lambda, \mu)$. For $t \in T$

we let $A_{\geq t} := \bigcup_{u \geq t} A_u$ ($u \geq t$ if there is a path from t to u in the (directed) tree $\mathcal{T}$). In particular, $A_{\geq r} = A$ for the root $r := r^{\mathcal{T}}$ of $\mathcal{T}$.

Furthermore, we let $B_r := \emptyset$ and $B_t := A_t \cap A_s$ for $t \in T \setminus \{r\}$ with parent s. Recall that the adhesion of $(\mathcal{T}, (A_t)_{t \in T})$ is at most ν. Thus $|B_t| \leq \nu$ for $t \in T$.

The *quantifier rank* of a first-order formula is the maximal depth of nested quantifiers in this formula. Let q be the quantifier rank of φ. Simple techniques from logic show that there is an algorithm that, given a vocabulary τ and $q, m \in \mathbb{N}$, computes a finite set $\Phi_{\tau,q,m}$ of first-order formulas of vocabulary τ of quantifier rank $\leq q$ with free variables among $v_1, \dots, v_m$ such that every such formula is equivalent to a formula in $\Phi_{\tau,q,m}$. Without loss of generality we can assume that $\varphi \in \Phi_{\tau,q,m}$ (otherwise we can compute a $\varphi' \in \Phi_{\tau,q,m}$ equivalent to φ and work with φ').

A (τ, q, m)-*type* is a subset of $\Phi_{\tau,q,m}$. Given a τ-structure $\mathcal{A}'$ and a set $B = \{b_1, \dots, b_m\} \subseteq A'$ let $\operatorname{tp}_q^{\mathcal{A}'}(B)$—more precisely, $\operatorname{tp}_q^{\mathcal{A}'}(b_1, \dots, b_m)$—be the (τ, q, m)-type

$$\operatorname{tp}_q^{\mathcal{A}'}(B) := \{\psi(v_1, \dots, v_m) \in \Phi_{\tau,q,m} \mid \mathcal{A}' \models \psi(b_1, \dots, b_m)\}.$$

We come back to our structure $\mathcal{A}$ and the tree-decomposition $(\mathcal{T}, (A_t)_{t \in T})$. By induction from the leaves to the root, for every $t \in T$ we compute $\operatorname{tp}_q^{\langle A_{\geq t} \rangle^{\mathcal{A}}}(B_t)$, which for brevity we denote by $\operatorname{tp}_q^{\geq t}(B_t)$. Since $B_r = \emptyset$, $\operatorname{tp}_q^{\geq r}(B_r)$ is a set of sentences, and we have $\mathcal{A} \models \varphi$ if and only if $\varphi \in \operatorname{tp}_q^{\geq r}(B_r)$.

Therefore let t be a vertex of $\mathcal{T}$ and assume that we have already computed $\operatorname{tp}_q^{\geq u}(B_u)$ for all children u of t (if there are any). (Actually, the case that t has no children is much simpler than the following general case because it is a direct application of Lemma 6.6.)

For every (τ, q, m)-type Φ we introduce a new $(m+1)$-ary relation symbol R_Φ. Furthermore, we let $P_1, \dots, P_\nu$ be new unary relation symbols and $\tau' := \tau \cup \{R_\Phi \mid m \leq \nu, \Phi \text{ a } (\tau, q, m)\text{-type}\} \cup \{P_1, \dots, P_\nu\}$. In four steps, we define a τ'-structure $\widetilde{\mathcal{A}}_t$ that contains all the relevant information to compute $\operatorname{tp}_q^{\geq t}(B_t)$. In the first three steps we define "intermediate" structures $\mathcal{A}_t^1$, $\mathcal{A}_t^2$, $\mathcal{A}_t^3$.

(1) $\mathcal{A}_t^1$ is the induced substructure of $\mathcal{A}$ with universe A_t.

(2) Suppose that $B_t = \{b_1, \dots, b_m\}$ for an $m \leq \nu$. Then $\mathcal{A}_t^2$ is the $\tau \cup \{P_1, \dots, P_\nu\}$-expansion of $\mathcal{A}_t^1$ with $P_i^{\mathcal{A}_t^2} := \{b_i\}$ for $1 \leq i \leq m$ and $P_i^{\mathcal{A}_t^2} := \emptyset$ for $m+1 \leq i \leq \nu$.

(3) $\mathcal{A}_t^3$ is obtained from $\mathcal{A}_t^2$ by adding a new vertex c_u for every child u of t and edges from c_u to all vertices of B_u.

(4) $\widetilde{\mathcal{A}}_t$ is the τ'-expansion of $\widetilde{\mathcal{A}}_t^3$ with

$$R_\Phi^{\widetilde{\mathcal{A}}_t} := \big\{(d_1, \dots, d_{m_u}, c_u) \;\big|\; u \text{ child of } t, \; B_u = \{d_1, \dots, d_{m_u}\}, \operatorname{tp}_q^{\geq u}(B_u) = \Phi\big\}.$$

Standard Ehrenfeucht–Fraïssé-type methods show that there is a computable function that associates with every formula $\psi \in \Phi_{\tau,q,m}$ a sentence $\widetilde{\psi} \in \mathrm{FO}[\tau']$ such that $\psi \in \operatorname{tp}_q^{\geq t}(B_t)$ if and only if $\widetilde{\mathcal{A}}_t \models \widetilde{\psi}$.

We claim that $\widetilde{\mathcal{A}}_t \in B(\lambda + 1, \mu)$. To see this, observe that the Gaifman graph $\mathcal{G}(\widetilde{\mathcal{A}}_t)$ is the graph obtained from the torso $[A_t]$ by adding the vertices c_u and edges between c_u and every element of B_u. Recall that, by the definition of the torso, each B_u is a clique in $[A_t]$. It is easy to see that adding vertices and connecting them with cliques can increase the tree-width of a graph by at most one. This implies the claim.

ModelCheck$_D$($\mathcal{A} \in STR$, $\varphi \in$ FO)

1 **if** $\mathcal{G}(\mathcal{A}) \notin D$ **then** reject
2 compute tree-decomposition $(\mathcal{T}, (A_t)_{t\in T})$ of $\mathcal{A}$ over $B(\lambda, \mu)$
3 $q := \mathrm{qr}(\varphi)$, $\tau :=$ vocabulary of φ
4 **for** $m = 0$ **to** ν
5 compute $\Phi_{\tau,q,m}$
6 **for all** $t \in T$ (from the leaves to the root)
7 compute $\widetilde{\mathcal{A}}_t$
8 $\mathrm{tp}_q^{\geq t}(B_t) := \emptyset$
9 $m := |B_t|$
10 **for all** $\psi \in \Phi_{\tau,q,m}$
11 compute $\widetilde{\psi}$
12 **if** $\widetilde{\mathcal{A}}_t \models \widetilde{\psi}$
13 **then** $\mathrm{tp}_q^{\geq t}(B_t) := \mathrm{tp}_q^{\geq t}(B_t) \cup \{\psi\}$
14 **if** $\varphi \in \mathrm{tp}_q^{\geq r}(B_r)$
15 **then** accept
16 **else** reject.

Algorithm 2.

Now we can put everything together and obtain an algorithm for $MC(\mathrm{FO})_{STR[D]}$, a high-level description of which is given as Algorithm 2. Its correctness is straightforward. Let us have a look at the running time: Let n be the size of the input structure. Then lines 1 and 2 require time polynomial in n (independently of φ). The time required in lines 3–5 only depends on φ. The main loop in lines 6–13 is called $|T|$ times, which is polynomial in n. Computing $\widetilde{\mathcal{A}}_t$ is polynomial in $|A_t|$ and the number of children of t since we have already computed $\mathrm{tp}_q^{\geq u}(B_u)$ for all children u of t (with constants heavily depending on $||\varphi||$). The main task is to decide whether $\widetilde{\mathcal{A}}_t \models \widetilde{\psi}$ in line 12; by Lemma 6.6 this is fixed-parameter tractable because $\widetilde{\mathcal{A}}_t \in B(\lambda + 1, \mu)$. The time required in lines 14–16 again depends only on φ. □

A consequence of our results is that slicewise first-order definable parameterized problems are fixed-parameter tractable when restricted to classes of structures whose underlying class of graphs has an excluded minor.

Corollary 6.8. *Let D be a* PTIME*-decidable class of graphs with an excluded minor and $P \subseteq STR \times \Pi^*$ a parameterized problem that is slicewise* FO*-definable. Then $P|_{STR[D]}$ is in* FPT.

7. A logical characterization of fixed-parameter tractability. In this section we give a characterization of FPT in the spirit of descriptive complexity theory.

We briefly review some facts from this area (see [13, 22] for details). It is common in descriptive complexity theory to identify decision problems, usually modeled by languages $L \subseteq \Sigma^*$ for a finite alphabet Σ, with classes of finite structures. More precisely, one identifies problems with classes of *ordered finite structures*. An *ordered structure* is a structure whose vocabulary contains the binary relation symbol $\leq$, and this symbol is interpreted as a linear order of the universe. *ORD* denotes the class of all ordered structures. In this section, τ always denotes a vocabulary that contains $\leq$.

One of the most important results in descriptive complexity theory is the Immer-

man–Vardi theorem [21, 30] saying that a class of ordered structures is in PTIME if and only if it is definable in least-fixed point logic FO(LFP). More concisely,

$$\text{PTIME} = \text{FO(LFP)}.$$

We prove a similar result characterizing the class FPT in terms of the finite variable least fixed-point logics LFP^s, for $s \geq 1$, which were introduced by Kolaitis and Vardi [25]. Analogously to the classical setting we model parameterized problems by subsets of $ORD[\tau] \times \mathbb{N}$ for some τ.

THEOREM 7.1. *A parameterized problem $P \subseteq ORD[\tau] \times \mathbb{N}$ is in* FPT *if and only if there is an $s \geq 1$ such that P is slicewise* LFP^s*-definable. More concisely, we may write*

$$\text{FPT} = \bigcup_{s \geq 1} \text{slicewise-LFP}^s.$$

In the proof of this result we assume that the reader is familiar with descriptive complexity theory, in particular with least fixed-point logic and the proof of the Immerman–Vardi theorem. Those who are not may safely skip the rest of this section.

We first recall the definition of LFP^s: In the terminology of [13, p. 174], LFP^s-sentences are FO(LFP)-sentences in the form

$$\exists \bar{y}[\text{S–LFP}_{\bar{x}_1, X_1, \ldots, \bar{x}_m, X_m} \varphi_1, \ldots, \varphi_m]\bar{y},$$

where $\varphi_1, \ldots, \varphi_m$ are first-order formulas with at most s individual variables. (That is, LFP^s-sentences are existential closures of simultaneous fixed-points over FO^s-formulas.)

We use the following two facts, the first implicit in [31] and the second in the proof of the Immerman–Vardi theorem (cf. [13]). Fix τ and $s \in \mathbb{N}$ and let n always denote the size of the input structure.

(1) There is a computable function that associates an $O(n^{2s})$-algorithm $\mathbb{A}_\varphi$ with each $\varphi \in \text{LFP}^s[\tau]$ such that $\mathbb{A}_\varphi$ accepts a structure $\mathcal{A} \in ORD[\tau]$ if and only if $\mathcal{A}$ satisfies φ.

(2) There is a $t \in \mathbb{N}$ and a computable function that associates with every $O(n^s)$-algorithm $\mathbb{A}$ accepting a class $C \subseteq ORD[\tau]$ a sentence $\varphi_\mathbb{A} \in \text{LFP}^t[\tau]$ such that a τ-structure $\mathcal{A}$ satisfies $\varphi_\mathbb{A}$ if and only if $\mathcal{A} \in C$.

There is a slight twist in (2). When proving it, one usually assumes that all structures are sufficiently large, in particular larger than the constant hidden in $O(n^s)$. This way it can be assumed that the algorithm is actually an n^{s+1}-algorithm (without any hidden constants). Then one argues that small structures are no problem because they can be described up to isomorphism in first-order logic. When restricting the number of variables, one has to be careful with such an argument. Luckily, we are safe here because we consider only ordered structures, and there is a $t \in \mathbb{N}$ (depending on τ) such that every structure $\mathcal{A} \in ORD[\tau]$ can be characterized up to isomorphism by an FO^t-sentence.

Proof of Theorem 7.1. For the backward direction, suppose that $P \subseteq ORD[\tau] \times \mathbb{N}$ is slicewise LFP^s-definable via $\delta : \mathbb{N} \to \text{LFP}^s$. Then Algorithm 3 shows that P is in FPT. The crucial fact is that lines 1 and 2 do not depend on the input structure $\mathcal{A}$ and line 3 requires time $O(n^{2s})$.

For the forward direction, suppose that $P \subseteq ORD[\tau] \times \mathbb{N}$ is in FPT. Choose $f : \mathbb{N} \to \mathbb{N}$, $c \in \mathbb{N}$ and an algorithm $\mathbb{A}$ deciding P in time $f(k) \cdot n^c$. The algorithm

```
DECIDE-P(𝒜 ∈ ORD, k ∈ ℕ)
1  compute δ(k) ∈ LFP^s
2  compute 𝔸_δ(k) (cf. (1))
3  simulate 𝔸_δ(k) on input 𝒜
4  if 𝔸_δ(k) accepts 𝒜
5      then accept
6      else reject.
```

Algorithm 3.

$\mathbb{A}$ gives rise to a sequence $\mathbb{A}_k$ ($k \geq 1$) of algorithms, where $\mathbb{A}_k$ decides the class $\{\mathcal{A} \mid (\mathcal{A}, k) \in P\} \subseteq ORD[\tau]$ in time $O(n^c)$. Then (2) yields the desired slicewise definition of P. □

8. Beyond FPT. In this last section we discuss how logical definability is related to the classes of the W-hierarchy. We introduce another hierarchy of parameterized problems, which we call the A-hierarchy, in terms of model-checking problems for first-order logic and show that the A-hierarchy can be seen as a parametric analogue of the polynomial hierarchy. We then discuss the relation between the A-hierarchy and the W-hierarchy.

Our treatment is motivated by the following two results. They relate the class W[1] to model-checking and computations of nondeterministic Turing machines, respectively. Recall that $MC(\Sigma_1[s])$ denotes the parameterized model-checking problem for existential formulas in prenex normal form whose vocabulary contains at most s-ary relation symbols.

THEOREM 8.1 (Downey, Fellows, and Regan [11]). *$MC(\Sigma_1)|_{GRAPH}$ is* W[1]*-complete under parameterized m-reductions.*

Proof. We prove that $CLIQUE \equiv^{\mathrm{fp}}_{\mathrm{m}} MC(\Sigma_1)|_{GRAPH}$.

$CLIQUE \leq^{\mathrm{fp}}_{\mathrm{m}} MC(\Sigma_1)|_{GRAPH}$ follows from the fact that *CLIQUE* is slicewise Σ_1-definable, so we only have to prove the converse.

An *atomic k-type* (in the theory of graphs) is a sentence $\theta(x_1, \dots, x_k)$ of the form $\bigwedge_{1 \leq i < j \leq k} \alpha_{ij}(x_i, x_j)$, where $\alpha_{ij}(x_i, x_j)$ is either $x_i = x_j$ or Ex_ix_j or $(\neg Ex_ix_j \wedge \neg x_i = x_j)$ (for $1 \leq i < j \leq k$).

It is easy to see that there is a computable mapping f that associates with every EFO-sentence φ a sentence $\tilde{\varphi}$ of the form

$$\bigvee_{i=1}^{l} \exists x_1 \dots \exists x_k \theta_i(x_1, \dots, x_k), \tag{8.1}$$

where each θ_i is an atomic k-type, such that for all graphs $\mathcal{G}$ we have $\mathcal{G} \models \varphi \iff \mathcal{G} \models \tilde{\varphi}$.

For each graph $\mathcal{G}$ and each atomic k-type $\theta(\bar{x}) = \bigwedge_{1 \leq i < j \leq k} \alpha_{ij}(x_i, x_j)$ we define a graph $h(\mathcal{G}, \theta)$ as follows:

- The universe of $h(\mathcal{G}, \theta)$ is $\{1, \dots, k\} \times G$.
- There is an edge between (i, v) and (j, w), for $1 \leq i < j \leq k$ and $v, w \in G$, if $\mathcal{G} \models \alpha_{ij}(v, w)$.

Then $h(\mathcal{G}, \theta)$ contains a k-clique if and only if $\mathcal{G} \models \exists \bar{x} \theta(\bar{x})$. Now we are ready to define the reduction from $MC(\Sigma_1)|_{GRAPH}$ to *CLIQUE*. Given an instance $(\mathcal{G}, \varphi)$ of

$MC(\Sigma_1)|_{GRAPH}$, we first compute the sentence

$$\tilde{\varphi} = \bigvee_{i=1}^{l} \exists x_1 \dots \exists x_k \theta_i.$$

We let $\mathcal{G}'$ be the disjoint union of the graphs $h(\mathcal{G}, \theta_i)$ for $1 \leq i \leq l$. Then $\mathcal{G}'$ has a k-clique if and only if $\mathcal{G} \models \varphi$. □

THEOREM 8.2 (Cai et al. [4]). *The following parameterized problem SHORT TURING MACHINE ACCEPTANCE (NM)*[4] *is* W[1]*-complete:*

NM	*Input:*	A nondeterministic Turing machine M.
	Parameter:	$k \in \mathbb{N}$.
	Question:	Does M accept the empty word in at most k steps?

Downey and Fellows call Theorem 8.2 a parameterized "analogue of Cook's theorem." In our notation, we may write $[NM]^{\mathrm{fp}}_{\mathrm{m}} = \mathrm{W}[1]$. It is now very natural to define the following "parameterized analogue of the polynomial hierarchy," which we call the A*-hierarchy*, by letting $\mathrm{A}[t] := [AM_t]^{\mathrm{fp}}_{\mathrm{m}}$ for all $t \geq 1$:

AM_t	*Input:*	An alternating Turing machine M whose initial state is existential.
	Parameter:	$k \in \mathbb{N}$.
	Question:	Does M accept the empty word in at most k steps with at most t alternations?

Note that $NM = AM_1$; thus $\mathrm{W}[1] = \mathrm{A}[1]$. Our following theorem can be seen as a natural generalization of Theorem 8.1.

THEOREM 8.3. *For all $t \geq 1$, the problem $MC(\Sigma_t)|_{GRAPH}$ is* A$[t]$*-complete under parameterized m-reductions. Thus*

$$\mathrm{A}[t] = [MC(\Sigma_t)|_{GRAPH}]^{\mathrm{fp}}_{\mathrm{m}} = \bigcup_{s \geq 1} [MC(\Sigma_t[s])]^{\mathrm{fp}}_{\mathrm{m}}.$$

Proof. The second equality follows from Corollary 3.4(3).

To prove that $MC(\Sigma_t)|_{GRAPH} \leq^{\mathrm{fp}}_{\mathrm{m}} AM_t$, we first observe that, for every graph $\mathcal{G}$ and every quantifier-free formula $\theta(x_1, \dots, x_m)$, in time $p(||\theta||) \cdot |G|^2$, for a suitable polynomial p, we can construct a deterministic Turing machine $M(\mathcal{G}, \theta)$ with input alphabet G that accepts an input word $a_1 \dots a_m$ over G if and only if $\mathcal{G} \models \theta(a_1, \dots, a_m)$ and that performs at most $f(||\theta||)$ steps for some computable function $f : \mathbb{N} \to \mathbb{N}$. Only to give an example, to check whether $Ex_k x_l$ holds, say, with $k < l$, we need states $s(i)$ for $1 \leq i \leq k$ and $s(i, a)$ for $k < i \leq l, a \in G$. The machine starts in state $s(1)$ with head in position 1 in state $s(1)$. It moves its head right until it reaches position k in state $s(k)$, reads a_k, and goes to position $(k+1)$ in state $s(k+1, a_k)$. Then it moves right again until it reaches position l in state $s(l, a_k)$. From this state it can reach an accepting state if and only if $E^{\mathcal{G}} a_k a_l$.

[4] *NM* stands for nondeterministic Turing machine. This notation should be seen in connection with the AM_t below, which refers to alternating Turing machines.

Now suppose we are given an instance of $MC(\Sigma_t)|_{GRAPH}$, i.e., a graph $\mathcal{G}$ and a sentence

$$\varphi = \exists x_{11} \dots \exists x_{1k_1} \forall x_{21} \dots \forall x_{2k_2} \ \dots \ Q x_{t1} \dots Q x_{tk_t} \ \theta,$$

where θ is quantifier-free. Then the following alternating Turing machine $M(\mathcal{G}, \varphi)$ accepts the empty word if and only if $\mathcal{G} \models \varphi$: It first writes a sequence of elements of $\mathcal{G}$ on the tape using existential and universal states appropriately and then simulates $M(\mathcal{G}, \theta)$ on this input. Again $g(||\theta||)$, for some computable function g, is an upper bound for the number of steps $M(\mathcal{G}, \varphi)$ has to perform.

To finish the proof, by Corollary 3.4(3) it suffices to show that $\mathrm{AM}_t \leq^{\mathrm{fp}}_{\mathrm{m}} MC(\Sigma_t[\tau])$ for a suitable vocabulary τ. To illustrate the idea, we first consider the case $t = 1$. Suppose we are given a nondeterministic Turing machine M with alphabet Σ, set Q of states, initial state q_0, accepting state q_{acc}, and transition relation δ. Let $\Sigma_H := \{a_H \mid a \in \Sigma\}$, a_H coding the information that the head of M scans a cell containing a. Let $\tau := \{ST, AL, H, IN, ACC, R, L, S\}$ with unary ST, AL, H, IN, ACC and 4-ary R, L, S and let $\mathcal{A}_M$ be the τ-structure given by

$$\begin{aligned}
&A_M := Q \dot\cup \Sigma \dot\cup \Sigma_H,\\
&ST^{\mathcal{A}_M} := Q, \qquad AL^{\mathcal{A}_M} := \Sigma, \qquad H^{\mathcal{A}_M} := \Sigma_H,\\
&IN^{\mathcal{A}_M} := \{q_0\}, \quad ACC^{\mathcal{A}_M} := \{q_{\mathrm{acc}}\},\\
&R^{\mathcal{A}_M} := \{(q, a_H, b, q') \mid (q, a, 1, b, q') \in \delta\},\\
&L^{\mathcal{A}_M} := \{(q, a_H, b, q') \mid (q, a, -1, b, q') \in \delta\},\\
&S^{\mathcal{A}_M} := \{(q, a_H, b_H, q') \mid (q, a, 0, b, q') \in \delta\} \cup \{(q_{\mathrm{acc}}, a_H, a_H, q_{\mathrm{acc}}) \mid a \in \Sigma\},
\end{aligned}$$

where $(q, a, h, b, q') \in \delta$ means that if M is in state q and its head scans $a \in \Sigma$, then M replaces a by b, moves its head one cell to the right ($h = 1$), moves to the left ($h = -1$), or does not move its head ($h = 0$); finally, it changes to state q'.

Let k be given as parameter for AM_1. In k steps, M scans at most the first k cells. The quantifier-free formula (note that the following formulas depend only on k and not on M)

$$\varphi_{\mathrm{config}}(x, y_1, \dots, y_k) := STx \wedge \bigvee_{i=1}^{k} \left(Hy_i \wedge \bigwedge_{j \neq i} ALy_j \right)$$

states that $(x, y_1, \dots, y_k)$ is a configuration with state x, with the head facing y_i, and with y_j being the content of the jth cell ($j \neq i$). Let $\varphi_{\mathrm{start}}(x, y_1, \dots, y_k)$ be a quantifier-free formula stating that $(x, y_1, \dots, y_k)$ is the starting configuration. Similarly, we define a quantifier-free formula $\varphi_{\mathrm{step}}(x, y_1, \dots, y_k, x', y'_1, \dots, y'_k)$ stating that the configuration $(x', y'_1, \dots, y'_k)$ is the successor configuration of $(x, y_1, \dots, y_k)$ (we agree that each accepting configuration is its own successor).

Now, the equivalence

$$M \text{ stops in } \leq k \text{ steps} \iff \mathcal{A}_M \models \varphi_k$$

holds for the following existential sentence φ_k:

$$\begin{aligned}
&\exists x_1 \exists y_{11} \dots \exists y_{1k} \ \dots \ \exists x_k \exists y_{k1} \dots \exists y_{kk} (\varphi_{\mathrm{start}}(x, y_{11}, \dots, y_{1k})\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{k-1} \varphi_{\mathrm{step}}(x_i, y_{i1}, \dots, y_{ik}, x_{i+1}, y_{i+11}, \dots, y_{i+1k}) \wedge ACCx_k).
\end{aligned}$$

For $t \geq 1$, we can proceed similarly with the addition that universal quantifiers are needed to take care of universal states of the input machine. Now, τ also contains two

further unary symbols F (for the existential states) and U (for the universal states) which get the corresponding interpretations in $\mathcal{A}_M$. For example, for $t = 2$, we can take a Σ_t-sentence equivalent to

$$\begin{aligned}
&\bigvee_{l\le k} \exists x_1\, \exists y_{11} \ldots \exists y_{1k} \;\ldots\; \exists x_k \exists y_{l1} \ldots \exists y_{lk} \\
&\quad \Big(\varphi_{\text{start}}(x, y_{11}, \ldots, y_{1k}) \\
&\wedge \textstyle\bigwedge_{i=1}^{l-1} \varphi_{\text{step}}(x_i, y_{i1}, \ldots, y_{ik}, x_{i+1}, y_{i+1\,1}, \ldots, y_{i+1\,k}) \\
&\wedge Fx_1 \wedge \ldots \wedge Fx_{l-1} \wedge Ux_l \\
&\wedge \forall x_{l+1} \forall y_{l+1\,1} \ldots \forall y_{l+1\,k} \;\ldots\; \forall x_k \forall y_{k1} \ldots \forall y_{kk} \\
&\qquad \big(\big(Ux_{l+1} \wedge \ldots \wedge Ux_{k-1} \\
&\qquad \wedge \textstyle\bigwedge_{i=l}^{k-1} \varphi_{\text{step}}(x_i, y_{i1}, \ldots, y_{ik}, x_{i+1}, y_{i+1\,1}, \ldots, y_{i+1\,k}) \big) \to ACCx_k \big) \Big).
\end{aligned}$$

(Without loss of generality we assume that the accepting state is both existential and universal and that at least one transition is always possible in a universal state.) □

The following result due to Downey, Fellows, and Regan [11] allows us to compare the W- and the A-hierarchy. Let $t, u \ge 1$. A formula φ is $\Sigma_{t,u}$ if it is Σ_t and all quantifier blocks after the leading existential block have length $\le u$.

THEOREM 8.4 (see [11]). *For all $t \ge 1$,*

$$\mathrm{W}[t] = \bigcup_{\substack{\tau \text{ vocabulary} \\ u \ge 1}} [MC(\Sigma_{t,u}[\tau])]^{\mathrm{fp}}_{\mathrm{m}} = \bigcup_{u \ge 1} [MC(\Sigma_{t,u})|_{GRAPH}]^{\mathrm{fp}}_{\mathrm{m}}.$$

Note that for $t = 1$, this is Theorem 8.1. The crucial step in proving the theorem for $t \ge 2$ is to establish the W[t]-completeness of the problems *WEIGHTED MONOTONE t-NORMALIZED SATISFIABILITY* (for even t) and *WEIGHTED ANTIMONOTONE t-NORMALIZED SATISFIABILITY* (for odd $t \ge 3$). We refer the reader to [10] for the (difficult) proofs of these results. Once these basic completeness results are established, it is relatively easy to derive Theorem 8.4 (cf. also our proof of Theorem 8.7). We encourage the reader to give a purely "logical" proof of the second equality in the theorem.

Since $\Sigma_{1,u} = \Sigma_1$, we get

$$\mathrm{W}[1] = \bigcup_{\tau \text{ vocabulary}} [MC(\Sigma_1[\tau])]^{\mathrm{fp}}_{\mathrm{m}} = [MC(\Sigma_1)|_{GRAPH}]^{\mathrm{fp}}_{\mathrm{m}} = \mathrm{A}[1].$$

By Theorem 8.3, W[t] $\subseteq$ A[t] for all $t \ge 2$. The question whether W[t] = A[t] for all t remains open; in view of Theorem 8.3 this question is equivalent to W[t] = $[MC(\Sigma_t)|_{GRAPH}]^{\mathrm{fp}}_{\mathrm{m}}$ for all $t \ge 1$. In this form it is stated as an open problem in [11]. Consider, for example, the following parameterized problem:

P_0		
	Input:	Graph $\mathcal{G}$.
	Parameter:	$(k, l) \in \mathbb{N}^2$.
	Question:	Are there $a_1, \ldots, a_k \in G$ such that every clique of size l contains an a_i?

Since P_0 is slicewise Σ_2-definable, we have $P_0 \in \mathrm{A}[2]$. However, is P_0 in $\mathrm{W}[2]$?

In the definition of the W-hierarchy we can restrict the length of the nonleading quantifier-blocks to one.

PROPOSITION 8.5. *For all $t \geq 1$,*

$$\mathrm{W}[t] = \bigcup_{\tau \text{ vocabulary}} [MC(\Sigma_{t,1}[\tau])]^{\mathrm{fp}}_{\mathrm{m}}.$$

Proof. The inclusion $\supseteq$ being trivial we turn to a proof of $\subseteq$: Fix τ and $u \geq 1$. We show that $MC(\Sigma_{t,u}[\tau]) \leq^{\mathrm{fp}}_{\mathrm{m}} MC(\Sigma_{t,1}[\tau'])$ for suitable τ'. The idea is to replace the blocks of at most u quantifiers by a single quantifier ranging over the set of u-tuples of a structure. To explain this idea we first use a vocabulary containing function symbols (and sketch afterwards how one can do without). Let $\tau' := \tau \cup \{T, p_1, \dots, p_u\}$, where T is a unary relation symbol (for ordered u-tuples) and $p_1, \dots, p_u$ are unary function symbols (the projection functions). Given a τ-structure $\mathcal{A}$ let $\mathcal{A}'$ be a τ'-structure with

$$A' := A \dot\cup A^u, \;\; T^{\mathcal{A}'} := A^u,$$

where for $(a_1, \dots, a_u) \in A^u$, $p_i(a_1, \dots, a_u) = a_i$ and where the relation symbols of τ are interpreted as in $\mathcal{A}$. Now, for example, for

$$\varphi = \exists x_1 \dots \exists x_k \forall y_1 \dots \forall y_u \psi(\bar{x}, \bar{y})$$

with quantifier-free ψ, let

$$\varphi' = \exists x_1 \dots \exists x_k \forall y (\neg T x_1 \wedge \dots \wedge \neg T x_k \wedge (Ty \to \psi(\bar{x}, p_1(y) \dots p_u(y)))).$$

Then,

$$\mathcal{A} \models \varphi \iff \mathcal{A}' \models \varphi',$$

which gives the desired parameterized m-reduction.

Let us explain, for the case $t = 2$, how to proceed to avoid function symbols. One has to add to τ, besides T as above, for every relation symbol $R \in \tau$, say, r-ary, every subset $M \subseteq \{1, \dots, r\}$, and every function $\rho : M \to \{1, \dots, u\}$ a new relation symbol $R_{M,\rho}$; e.g., if $M = \{s, \dots, r\}$, then $R^{\mathcal{A}'}_{M,\rho} a_1 \dots a_{s-1} b$ if and only if $a_1 \dots a_{s-1} \in A$, $b = (b_1, \dots, b_u) \in A^u$, and $R^{\mathcal{A}} a_1 \dots a_{s-1} b_{\rho(s)} \dots b_{\rho(r)}$; and a subformula $R x_{i_1} \dots x_{i_{s-1}} y_{\rho(s)} \dots y_{\rho(r)}$ of φ is replaced by $R_{M,\rho} x_{i_1} \dots x_{i_{s-1}} y$. □

Downey, Fellows, and Regan [11] also gave a (much simpler) characterization of the W-hierarchy in terms of Fagin definability. We find it worthwhile to sketch a short proof of this result. For a class Φ of formulas we let $\mathrm{FD}(\Phi)$ be the class of all problems that are Φ-Fagin-definable. Let $\Pi_t[s]$ denote the class of all Π_t-formulas whose vocabulary is at most s-ary.

THEOREM 8.6 (see [11]). *For all $t \geq 1$ we have* $\mathrm{W}[t] = [\mathrm{FD}(\Pi_t[2])]^{\mathrm{fp}}_{\mathrm{m}} = [\mathrm{FD}(\Pi_t)]^{\mathrm{fp}}_{\mathrm{m}}$.

Proof. Recall that $\mathrm{W}[t] = \bigcup_{d \geq 1} [WSAT(C_{t,d})]^{\mathrm{fp}}_{\mathrm{m}}$, where $C_{t,d}$ is the class of all propositional formulas of the form

$$\bigwedge_{i_1} \bigvee_{i_2} \cdots \Big(\bigwedge_{i_t} / \bigvee_{i_t} \Big) \varphi_{i_1 \dots i_t}, \tag{8.2}$$

where the $\varphi_{i_1\dots i_t}$ are small formulas of depth at most d.

To prove that $\mathrm{W}[t] \subseteq [\mathrm{FD}(\Pi_t[2])]^{\mathrm{fp}}_{\mathrm{m}}$, we first transform a propositional formula φ of the form (8.2) into a propositional formula φ' of essentially the same form, but with all the $\varphi_{i_1\dots i_t}$ being disjunctions (if t is odd) or conjunctions (if t is even) of exactly d' literals, for some constant d' depending only on d. This can be done by first transforming the small formulas into equivalent formulas in conjunctive normal form or disjunctive normal form, respectively, and then repeatedly replacing disjunctions (respectively, conjunctions) γ with less than the maximum number of literals by the two clauses $\gamma \vee X$ and $\gamma \vee \neg X$ ($\gamma \wedge X$ and $\gamma \wedge \neg X$, respectively) for some variable X not appearing in γ.

We associate with φ' an $\{E, P, N, T, L\}$-structure $\mathcal{C}$ which is obtained from the tree corresponding to φ' as follows: We first remove the root. Then we identify all leaves corresponding to the same propositional variable. To indicate whether a variable occurs positively or negatively in a clause, we use the binary relations P and N. The unary relation T contains all the top level nodes, and the unary relation L contains all the (former) leaves. It is easy to write a Π_t-formula $\psi'(X)$ such that φ' has a satisfying assignment of weight k if and only if there exists a k-element subset $B \subseteq C$ such that $\mathcal{C} \models \psi'(B)$.

This can best be illustrated with a simple example: Let

$$\varphi' := (X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee Y \vee \neg Z).$$

The corresponding structure $\mathcal{C}$ is displayed in Figure 8.1.

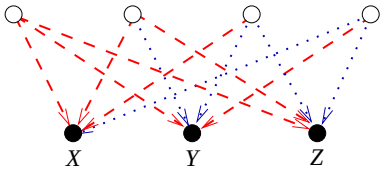


FIG. 8.1.

We let

$$\psi'(U) := \forall x(Ux \to Lx) \wedge \forall x \forall y_1 \forall y_2 \forall y_3 \Bigg(\Bigg(Tx \wedge \bigwedge_{i=1}^{3} Exy_i \wedge \bigwedge_{1 \le i < j \le 3} y_i \neq y_j \Bigg) \longrightarrow \bigvee_{i=1}^{3} \big((Pxy_i \wedge Uy_i) \vee (Nxy_i \wedge \neg Uy_i) \big) \Bigg).$$

To prove $[\mathrm{FD}(\Pi_t)]^{\mathrm{fp}}_{\mathrm{m}} \subseteq \mathrm{W}[t]$, we note only that for every formula $\psi(X) \in \Pi_t$ and every structure $\mathcal{A}$, there is a $C_{t,d}$-formula φ with the property that every assignment α for φ corresponds to a set B, whose size is the weight of the assignment, such that α satisfies φ if and only if $\mathcal{A} \models \psi(B)$. Here d is a constant that depends only on ψ. Furthermore, the transformation $(\mathcal{A}, \psi) \mapsto \varphi$ is computable in time polynomial in $\mathcal{A}$. □

We do not know of any simple proof of the equivalence between the two characterizations of the W-hierarchy in terms of slicewise $\Sigma_{t,u}$-definability (cf. Theorem 8.4) and Π_t-Fagin definability. While the proof of the previous theorem shows that Π_t-Fagin definability is actually quite close to the definition of $\mathrm{W}[t]$ in terms of the

weighted satisfiability problem for C_t-formulas, it seems that it is a significant step to get from there to slicewise $\Sigma_{t,u}$-definability.

Our last result is another characterization of the W-hierarchy in terms of Fagin definability that is much closer to the slicewise characterization. A first-order formula $\varphi(X)$ is *bounded to* the r-ary relation variable X if in $\varphi(X)$ quantifiers appear only in the form $\exists x_1 \dots \exists x_r(Xx_1 \dots x_r \wedge \psi)$ or $\forall x_1 \dots \forall x_r(Xx_1 \dots x_r \rightarrow \psi)$, which we abbreviate by $\exists \bar{x} \in X\ \psi$ and $\forall \bar{x} \in X\ \psi$, respectively. For $t \geq 1$ we let Π^b_t be the class of all formulas $\varphi(X)$ of the form

$$\forall \bar{x}_1 \exists \bar{x}_2 \dots Q\bar{x}_t \theta,$$

where $Q = \forall$ if t is odd and $Q = \exists$ otherwise and where θ is bounded to X.

For example, *CLIQUE* is Fagin-defined by the Π^b_0-formula

$$\forall x \in X \forall y \in X(x \neq y \rightarrow Exy)$$

and *DOMINATING SET* by the Π^b_1-formula

$$\forall x \exists y \in X(x = y \vee Exy).$$

THEOREM 8.7. *For $t \geq 1$, $\mathrm{W}[t] = [\mathrm{FD}(\Pi^b_{t-1})]^{\mathrm{fp}}_{\mathrm{m}}$.*

Proof. First, assume that the problem $P \subseteq STR[\tau] \times \mathbb{N}$ is Fagin-defined by $\varphi(X) \in \Pi^b_{t-1}$, say,

$$\varphi(X) := \forall \bar{y}_1 \exists \bar{y}_2 \forall \bar{y}_3 \dots Q\bar{y}_{t-1}\psi,$$

where ψ contains only bounded quantifiers. Let l be the maximum of the lengths of the tuples $\bar{y}_i$ for $1 \leq i \leq t$. For simplicity, let us assume that X is unary. Since Xy is equivalent to $\exists z \in X\, z = y$, we can assume that in ψ, the variable X occurs only in quantifier bounds. We show that $P \in \mathrm{W}[t]$. Given a parameter k set (with new variables $x_1, \dots, x_k$)

$$\varphi^k := \exists x_1 \dots \exists x_k \forall \bar{y}_1 \exists \bar{y}_2 \forall \bar{y}_3 \dots Q\bar{y}_{t-1} \left(\bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \psi^* \right),$$

where ψ^* is obtained from ψ by inductively replacing $\forall u \in X \chi(u)$ and $\exists u \in X\, \chi(u)$ by $\bigwedge_{i=1}^k \chi(x_i)$ and $\bigvee_{i=1}^k \chi(x_i)$, respectively. Note that φ^k is a $\Sigma_{t,l}$-formula and that for every structure $\mathcal{A}$,

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \varphi^k.$$

Thus, P is slicewise $\Sigma_{t,l}$-definable and hence in $\mathrm{W}[t]$.

For the converse direction, we prove that for all $t, u \geq 1$ we have

$$MC(\Sigma_{t,u})|_{GRAPH} \subseteq \mathrm{FD}(\Pi^b_{t-1}).$$

The idea is to associate with every graph $\mathcal{G}$ and $\Sigma_{t,u}$-formula $\exists x_1 \dots \exists x_k \varphi$ a structure $\mathcal{C}$ which essentially is a Boolean circuit whose satisfying assignments of size k correspond to assignments to the variables $x_1, \dots, x_k$ such that $\mathcal{G}$ satisfies φ. We can Fagin-define the weighted satisfiability problem for this circuit by a Π^b_{t-1}-formula. We leave the details to the reader.

For readers familiar with [10] (Theorem 12.6 on p. 299 is the relevant result), we state another proof. For $t = 1$, the result follows from the fact the W[1]-complete problem *CLIQUE* is Π_0^b-Fagin-definable. For odd $t \geq 2$, the problem *WEIGHTED ANTIMONOTONE t-NORMALIZED SATISFIABILITY* is W[t]-complete. It is parameterized m-reducible to the problem Fagin-defined by the Π_{t-1}^b-formula

$$\varphi(X) := \forall y_0 \forall y_1 \exists y_2 \forall y_3 \dots \exists y_{t-1} \\ ((Ey_0y_1 \wedge Ey_1y_2 \wedge \dots \wedge Ey_{t-2}y_{t-1}) \rightarrow \forall x \in X \neg Ey_{t-1}x).$$

For even t we use the completeness of *WEIGHTED MONOTONE t-NORMALIZED SATISFIABILITY* and argue similarly. □

Remark 8.8. Downey, Fellows, and Taylor [12] proved that the parameterized model-checking problem for full first-order logic is complete for the class AW[$*$], a parameterized complexity class above the W-hierarchy that is defined in terms of the satisfiability problem for quantified Boolean formulas.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[3] L. Cai and J. Chen, *On fixed-parameter tractability and approximability of NP optimization problems*, J. Comput. System Sci., 54 (1997), pp. 465–474.

[4] L. Cai, J. Chen, R. Downey, and M. Fellows, *On the parameterized complexity of short computation and factorization*, Arch. Math. Logic, 36 (1997), pp. 321–337.

[5] A. Chandra and P. Merlin, *Optimal implementation of conjunctive queries in relational data bases*, in Proceedings of the 9th ACM Symposium on Theory of Computing, Boulder, CO, 1977, pp. 77–90.

[6] C. Chekuri and A. Rajaraman, *Conjunctive query containment revisited*, in Proceedings of the 5th International Conference on Database Theory, P. Kolaitis and F. Afrati, eds., Lecture Notes in Comput. Sci. 1186, Springer-Verlag, Berlin, 1997, pp. 56–70.

[7] B. Courcelle, *Graph rewriting: An algebraic and logic approach*, in Handbook of Theoretical Computer Science, Vol. 2, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 194–242.

[8] R. G. Downey and M. R. Fellows, *Fixed-parameter tractability and completeness* I*: Basic results*, SIAM J. Comput., 24 (1995), pp. 873–921.

[9] R. Downey and M. Fellows, *Fixed-parameter tractability and completeness* II*: On completeness for W*[1], Theoret. Comput. Sci., 141 (1995), pp. 109–131.

[10] R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[11] R. Downey, M. Fellows, and K. Regan, *Descriptive complexity and the W-hierarchy*, in Proof Complexity and Feasible Arithmetics, P. Beame and S. Buss, eds., AMS-DIMACS Proceedings Series 39, 1998, pp. 119–134.

[12] R. Downey, M. Fellows, and U. Taylor, *The parameterized complexity of relational database queries and an improved characterization of W*[1], in Combinatorics, Complexity, and Logic, D. Bridges, C. Calude, J. Gibbons, S. Reeves, and I. Witten, eds., Springer-Verlag, 1996, pp. 194–213.

[13] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag, Berlin, 1995.

[14] R. Fagin, *Generalized first–order spectra and polynomial–time recognizable sets*, in Complexity of Computation, R. M. Karp, ed., Proc. Sympos. Appl. Math. 7, AMS, Providence, RI, 1974, pp. 43–73.

[15] M. Fellows and U. Stege, *An Improved Fixed-Parameter-Tractable Algorithm for Vertex Cover*, Tech. Rep. 318, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 1999.

[16] J. Flum, M. Frick, and M. Grohe, *Query evaluation via tree-decompositions*, in Proceedings of the 8th International Conference on Database Theory, J. van den Bussche and V. Vianu, eds., Lecture Notes in Comput. Sci. 1973, Springer-Verlag, Berlin, Heidelberg, New York, 2001, pp. 22–38.

[17] M. Frick and M. Grohe, *Deciding first-order properties of locally tree-decomposable graphs*, in Proceedings of the 26th International Colloquium on Automata, Languages and Pro-

gramming, J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 331–340.

[18] G. Gottlob, N. Leone, and F. Scarcello, *Hypertree decompositions and tractable queries*, in Proceedings of the 18th ACM Symposium on Principles of Database Systems, Philadelphia, PA, 1999, pp. 21–32.

[19] M. Grohe, *Local tree-width, excluded minors, and approximation algorithms*, Combinatorica, to appear.

[20] M. Grohe, T. Schwentick, and L. Segoufin, *When is the evaluation of conjunctive queries tractable*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, Crete, Greece, to appear.

[21] N. Immerman, *Relational queries computable in polynomial time*, Inform. and Control, 68 (1986), pp. 86–104.

[22] N. Immerman, *Descriptive Complexity*, Springer-Verlag, New York, 1999.

[23] P. Kolaitis and M. Thakur, *Approximation properties of NP minimization classes*, J. Comput. System Sci., 50 (1995), pp. 391–411.

[24] P. Kolaitis and M. Vardi, *Conjunctive-query containment and constraint satisfaction*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems, Seattle, WA, 1998, pp. 205–213.

[25] P. G. Kolaitis and M. Y. Vardi, *On the expressive power of variable-confined logics*, in Proceedings of the 11th IEEE Symposium on Logic in Computer Science, New Brunswick, NJ, 1996.

[26] C. Papadimitriou and M. Yannakakis, *On the complexity of database queries*, in Proceedings of the 17th ACM Symposium on Principles of Database Systems, Tucson, AZ, 1997, pp. 12–19.

[27] J. Plehn and B. Voigt, *Finding minimally weighted subgraphs*, in Graph-Theoretic Concepts in Computer Science, R. Möhring, ed., Lecture Notes in Comput. Sci. 484, Springer-Verlag, Berlin, 1991, pp. 18–29.

[28] N. Robertson and P. Seymour, *Graph minors* XIII. *The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.

[29] D. Seese, *Linear time computable problems and first-order descriptions*, Math. Structures Comput. Sci., 6 (1996), pp. 505–526.

[30] M. Vardi, *The complexity of relational query languages*, in Proceedings of the 14th ACM Symposium on Theory of Computing, San Francisco, CA, 1982, pp. 137–146.

[31] M. Vardi, *On the complexity of bounded-variable queries*, in Proceedings of the 14th ACM Symposium on Principles of Database Systems, San Jose, CA, 1995, pp. 266–276.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 146–166

APPROXIMATION TECHNIQUES FOR AVERAGE COMPLETION TIME SCHEDULING*

C. CHEKURI†, R. MOTWANI‡, B. NATARAJAN§, AND C. STEIN¶

Abstract. We consider the problem of nonpreemptive scheduling to minimize average (weighted) completion time, allowing for release dates, parallel machines, and precedence constraints. Recent work has led to constant-factor approximations for this problem based on solving a preemptive or linear programming relaxation and then using the solution to get an ordering on the jobs. We introduce several new techniques which generalize this basic paradigm. We use these ideas to obtain improved approximation algorithms for one-machine scheduling to minimize average completion time with release dates. In the process, we obtain an optimal randomized on-line algorithm for the same problem that beats a lower bound for deterministic on-line algorithms. We consider extensions to the case of parallel machine scheduling, and for this we introduce two new ideas: first, we show that a preemptive one-machine relaxation is a powerful tool for designing parallel machine scheduling algorithms that simultaneously produce good approximations and have small running times; second, we show that a nongreedy "rounding" of the relaxation yields better approximations than a greedy one. We also prove a general theorem relating the value of one-machine relaxations to that of the schedules obtained for the original m-machine problems. This theorem applies even when there are precedence constraints on the jobs. We apply this result to obtain improved approximation ratios for precedence graphs such as in-trees, out-trees, and series-parallel graphs.

Key words. approximation algorithms, scheduling, parallel machine scheduling, release dates, precedence constraints

AMS subject classifications. 68Q25, 68W20, 68W25, 68W40, 90B35

PII. S0097539797327180

1. Introduction.

We present new approximation techniques and results for *nonpreemptive* scheduling to minimize *average (weighted) completion time* (equivalently, sum of (weighted) completion times). In this problem, we are given n jobs $J_1, \ldots, J_n$, where job J_j has processing time p_j, release date r_j, and a positive weight w_j. A feasible schedule S assigns jobs nonpreemptively[1] to m machines such that each job starts after its release date. Let C_j^S denote the completion time of job J_j in schedule S. The

*Received by the editors September 16, 1997; accepted for publication (in revised form) May 9, 2000; published electronically June 5, 2001. A preliminary version [5] of this paper appeared in SODA 1997.

http://www.siam.org/journals/sicomp/31-1/32718.html

†Bell Labs, 600 Mountain Ave., Murray Hill, NJ 07974 (chekuri@research.bell-labs.com). This work was done while the author was at Stanford University where he was supported by NSF Award CCR-9357849 with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡Department of Computer Science, Stanford University, Stanford, CA 94305-9045 (rajeev@cs.stanford.edu). This author was supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, ARO MURI grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

§Arcot Systems, 2490 Sand Hill Road, Menlo Park, CA 94025 (nat@arcot.com). This work was done while the author was at Hewlett Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304.

¶Department of Computer Science, Dartmouth College, Hanover, NH 03755 (cliff@cs.dartmouth.edu). This author's research was partly supported by NSF Award CCR-9308701, NSF Career Award CCR-9624828, NSF Award DMI-9970063, and an Alfred P. Sloane Foundation Fellowship. Most of this work was done while the author was visiting Stanford University.

[1] In a nonpreemptive schedule, each job runs uninterrupted on one machine from start to finish; in a preemptive schedule, a job may be interrupted or may switch machines at any time.

objective is to minimize the *weighted* completion time $\sum_j w_j C_j^S$; if all w_j are $1/n$, the objective becomes the *average* completion time. For the single machine case, if the release dates are 0 for all jobs, then the weighted completion time problem can be solved optimally in polynomial time [30]. We are interested in a more general setting with release dates and precedence constraints and multiple machines, any of which makes the problem $\mathcal{NP}$-hard [23]. Thus we will consider approximation algorithms, or, in an on-line setting, competitive ratios.

An important motivation for studying scheduling to minimize sum of weighted completion times, aside from its intrinsic theoretical interest, comes from application to compiler optimizations. Compile-time instruction scheduling is essential for effectively exploiting the fine-grained parallelism offered in pipelined, superscalar, and very-long instruction word architectures (see, for example, [17, 35]). Current research is addressing the issue of profile-based compiler optimization. In a recent paper, Chekuri et al. [4] show that weighted completion time is the measure of interest in profile-driven code optimization; some of our results are related to the heuristics described and empirically tested therein.

Recent work has led to constant-factor approximations for weighted completion time for a variety of these $\mathcal{NP}$-hard scheduling problems [25, 16, 3, 12, 7, 28]. Most of these algorithms work by first constructing a relaxed solution, either a preemptive schedule or a linear programming relaxation. These relaxations are used to obtain an ordering of the jobs, and then the jobs are list scheduled as per this ordering.

We introduce new techniques that generalize this basic paradigm. We use these to obtain improved approximation algorithms for one-machine scheduling to minimize average completion time with release dates. Our main result here is a $\frac{e}{e-1} \approx 1.58$-approximation algorithm. This algorithm can be turned into a randomized on-line algorithm with the same bound, where an algorithm is on-line if before time r_j it is unaware of J_j, but at time r_j it learns all the parameters of J_j. This randomized on-line algorithm is particularly interesting as it beats a lower bound for deterministic on-line algorithms [21] and matches a recent lower bound for randomized on-line algorithms [33].

We then consider extensions to parallel machine scheduling and introduce two new ideas: first, we show that a preemptive one-machine relaxation is a powerful tool for designing parallel machine scheduling algorithms that simultaneously produce good approximations and have small running times; second, we show that a nongreedy "rounding" of the relaxation produces better approximations than simple list scheduling. In fact, we prove a general theorem relating the value of one-machine relaxations to that of the schedules obtained for the original m-machine problems. This theorem applies even when there are precedence constraints yielding better approximations for precedence graphs such as in-trees, out-trees, and series-parallel graphs, which are of interest in compiler applications that partly motivated our work.

The bounds in this paper derive from proving bounds on the ratio between solutions to a nonpreemptive scheduling problem and a relaxed version of this problem. The bounds on this ratio all hold when both the original and relaxed problems have weights; however, the relaxed problem with weights is typically not solvable exactly in polynomial time. Because of this, the performance ratios of our algorithms, in some cases, are not as good as those obtained directly through other techniques. However, given future improvement in state of the art for one-machine preemptive scheduling with release dates and/or precedence constraints, our results would also imply better

bounds for weighted completion time. We begin with a more detailed discussion of our results and their relation to earlier work.

One-machine scheduling with release dates. The first constant-factor approximation algorithm for an average completion time problem was the following 2-approximation algorithm of Phillips, Stein, and Wein [25] for minimizing the average completion time on one machine with release dates. First, an optimal preemptive schedule P is found using the shortest remaining processing time (SRPT) algorithm [23] which at any time runs an available job that has the least processing time left; note that this is an on-line algorithm. Given P, the jobs are ordered by increasing completion times, C_j^P, and are scheduled according to that ordering, introducing idle time as necessary to account for release dates. A simple proof shows that each job J_j completes at time no later than $2C_j^P$, implying a 2-approximation. Other 2-approximation algorithms have been discovered subsequently [21, 32, 12], and it is also known that no deterministic on-line algorithm has approximation ratio better than 2 [21, 32]. This approximation technique has been generalized to many other scheduling problems, and hence finding better approximations for this basic problem is believed to be an important step toward improved approximations for more general problems.

Our main result here is a deterministic off-line algorithm for the basic problem that gives an $\frac{e}{e-1}$-approximation ($\frac{e}{e-1} \approx 1.58$). We also obtain an *optimal* randomized on-line algorithm (in the *oblivious* adversary model) with expected competitive ratio $\frac{e}{e-1}$. This beats the deterministic on-line lower bound. Our approach is based on what we call α-*schedules* (this notion was also used by [25] and [15] in a somewhat different manner). Given a preemptive schedule P and $\alpha \in (0, 1]$, we define $C_j^P(\alpha)$ to be the time at which αp_j, an α-fraction of J_j, is completed. An α-schedule is a nonpreemptive schedule obtained by list scheduling jobs in order of increasing $C_j^P(\alpha)$, possibly introducing idle time to account for release dates. Clearly, an α-scheduler is an on-line algorithm; moreover, for $\alpha = 1$, the α-scheduler is exactly the algorithm of Phillips, Stein, and Wein [25] and hence a 2-approximation. We show that for arbitrary α, an α-scheduler has a *tight* approximation ratio of $1 + 1/\alpha$. Given that $1 + 1/\alpha \geq 2$ for $\alpha \in (0, 1]$, it may appear that this notion of α-schedulers is useless for obtaining ratios better than 2.

A key observation is that a worst-case instance that induces a performance ratio $1+1/\alpha$ for one value of α is not a worst-case instance for many other values of α. This suggests that a randomized algorithm which picks α at random, and then behaves like an α-scheduler, may lead to an approximation better than 2. Unfortunately, we show that choosing α uniformly at random gives an expected approximation ratio of 2 and that this is tight. However, this leaves open the possibility that for any given instance I, there exists a choice $\alpha(I)$ for which the $\alpha(I)$-schedule yields an approximation better than 2. We refer to the resulting deterministic off-line algorithm which, given I, chooses α to minimize $\alpha(I)$, as BEST-α.

It turns out, however, that the randomized on-line algorithm which chooses α to be 1 with probability 3/5 and 1/2 with probability 2/5 has competitive ratio 1.8; consequently, for any input I, either the 1-schedule or the $\frac{1}{2}$-schedule is no worse than an 1.8-approximation. More significantly, the nonuniform choice in the randomized version suggests the possibility of defining randomized choices of α that may perform better than 1.8 while being easy to analyze. In fact, our main result here is that it is possible to define a distribution for α that yields a randomized on-line $\frac{e}{e-1}$-approximation algorithm implying that BEST-α is an $\frac{e}{e-1}$-approximation algo-

rithm. It should be noted that BEST-α can be implemented in $O(n^2)$ time and all our other algorithms can be implemented in either $O(n)$ or $O(n \log n)$ time. Torng and Uthaisombut [34] recently showed that our analysis of BEST-α is tight by giving instances on which the approximation ratio achieved by BEST-α is arbitrarily close to $e/(e-1)$.

Our bounds are actually job-by-job, i.e., we produce a schedule N in which $E[C_j^N] \leq \frac{e}{e-1} C_j^P$ for all j where $E[C_j^N]$ is the expected completion time of J_j in the nonpreemptive schedule. Thus our conversion results generalize to weighted completion time. However, since for the weighted case even the preemptive scheduling problem is $\mathcal{NP}$-hard (given release dates), we must use an approximation for the relaxation. The current best approximation algorithm for the preemptive case [26] has a ratio of 4/3, which yields a 2.12-approximation for the nonpreemptive case. However, this does not improve earlier results.

Independently, Goemans [12] has used similar ideas to design a 2-approximation algorithm for the problem of nonpreemptive scheduling on one machine so as to minimize the average weighted completion time. His algorithm is also a BEST-α algorithm and works off a preemptive schedule that is optimal for a certain linear programming relaxation of the problem and the analysis is based on the linear programming formulation. Interestingly, Goemans proves the performance of his algorithm by choosing α uniformly at random in the interval $(0, 1]$. Further work [26, 13] based on the idea of using independent random α points for each job has resulted in an improved approximation ratio of 1.6853.

Scheduling parallel machines with release dates. We consider generalizations of the single machine problems to the case of m identical parallel machines. We first consider the problem of minimizing average completion time with release dates and no precedence constraints. Extending the techniques to the m-machine problem gives rise to two complications: the problem of computing an optimal m-machine preemptive schedule is $\mathcal{NP}$-hard [9], and the best known approximation ratio is 2 [25]; further, the conversion bounds from preemptive to nonpreemptive schedules are not as good. Chakrabarti et al. [3] obtain a bound of 7/3 on the conversion from the preemptive to the nonpreemptive case yielding a 14/3-approximation for scheduling on m machines with release dates. Several other algorithms do not use the preemptive schedule but use a linear programming relaxation. Phillips, Stein, and Wein [25] gave the first such algorithm, a 24-approximation algorithm. This has been greatly improved to $4 + \epsilon$ [15], $4 - \frac{1}{m}$ [16], and 3.5 [3]. Using a general on-line framework [3], one can obtain an algorithm with an approximation ratio of $2.89 + \epsilon$. Unfortunately, the algorithm with the best approximation is inefficient, as it uses the polynomial approximation scheme for makespan due to Hochbaum and Shmoys [20].

We give a new algorithm for this problem. First we introduce a different relaxation—a *preemptive one-machine relaxation*. More precisely, we maintain the original release dates and allow preemptions but divide all the processing times by m. We then compute a one-machine schedule. The resulting completion time ordering is then used to generate a nonpreemptive m-machine schedule that is a 3-approximation. Our algorithm has a running time of $O(n \log n)$ and in addition is also on-line. We then show that the approximation ratio can be improved to 2.83 using a general conversion algorithm that we develop. This improves on the approximation bounds of previous algorithms and gives a much smaller running time of $O(n \log n)$. Subsequent to our work, Schulz and Skutella [28] using some of our ideas have obtained an approximation ratio of 2 for the more general case of sum of weighted completion times.

Scheduling with precedence constraints. We now consider the weighted completion time problem with precedence constraints. For one-machine scheduling, minimizing the weighted completion time is $\mathcal{NP}$-hard for arbitrary precedence constraints even without release dates [11, 22]. A 2-approximation for the case of no release dates [16] and an $e \simeq 2.718$-approximation [29] with release dates are known. For arbitrary m, the problem is $\mathcal{NP}$-hard even without precedence constraints and release dates if weights are not required to be identical; on the other hand, the problem is strongly $\mathcal{NP}$-hard even when all weights are identical and the precedence graph is a union of chains [10]. An expected approximation ratio of $5.33 + \epsilon$ is achievable with release dates and precedence constraints [3]. This has been recently improved to 4 in [24].

A general conversion algorithm. We obtain a fairly general algorithm for m-machine problems with precedence constraints, release dates, and job weights. To do so, we first solve a one-machine preemptive relaxation and apply an algorithm we call Delay List to get an m-machine nonpreemptive schedule. Since, in general, the one-machine preemptive relaxation is also $\mathcal{NP}$-hard, we would have to settle for a ρ-approximation for it; then our algorithm would give a $(2\rho + 2)$-approximation for the m-machine case. In fact, we give an algorithm that gives a $(1 + \beta)\rho + (1 + 1/\beta)$-approximation for any $\beta > 0$ which when optimized for ρ yields a $(\rho + 2\sqrt{\rho} + 1)$-approximation. In the absence of release dates an optimal one-machine schedule can be computed in polynomial time when the precedence graph is a forest [19] or a series-parallel graph [22, 1]. Applying our conversion algorithm for these cases results in improved approximation results. Although the algorithm fails to obtain improved results for the most general problem, we feel that it is of independent interest and likely to find applications in the future. Further, our conversion algorithm has the advantage of being simple and combinatorial. In applications such as compilers [4], speed and simplicity are sometimes more important than getting the best possible ratio. Finally, our algorithm has a surprising property: it gives schedules that are good for both makespan and average completion time (Chakrabarti et al. [3] and Stein and Wein [31] also have shown the existence of such schedules).

2. One-machine scheduling with release dates. In this section, we present our results for one-machine scheduling with release dates to minimize average completion time. Let P be a preemptive schedule, and let C_i^P and C_i^α denote the completion time of J_i in P and in the nonpreemptive α-schedule derived from P, respectively. We begin by analyzing simple α-schedules. Techniques from [25, 16] are easily generalized to yield the following.

Theorem 2.1. *Given an instance of one-machine scheduling with release dates, for any $\alpha \in (0, 1]$, an α-schedule has $\sum_j C_j^\alpha \le (1 + 1/\alpha) \sum_j C_j^P$. Further, there are instances where the inequality is asymptotically tight.*

Proof. Index the jobs by the order of their α-points in the preemptive schedule P. Let $r_j^{\max} = \max_{1 \le k \le j} r_k$ be the latest release date among jobs with α points no greater than j's. By time $r_j^{\max}$, jobs 1 through j have all been released, and hence

$$C_j^\alpha \le r_j^{\max} + \sum_{k=1}^{j} p_k. \tag{2.1}$$

We know that $C_j^P \ge r_j^{\max}$, since only an α fraction j has finished by $r_j^{\max}$. We also know that $C_j^P \ge \alpha \sum_{k=1}^{j} p_k$, since the α fractions of jobs $1, \ldots, j$ must run before time C_j^P. Plugging these last two inequalities into (2.1) yields $C_j^\alpha \le (1 + \frac{1}{\alpha}) C_j^P$. Summing

over all j yields the lemma.

To see that this is tight, consider the following class of instances. Let ϵ be a small positive number. We will also allow jobs with processing time 0, although the proof can be modified even if these are not allowed. At time 0, we release a job with processing time 1. At time $\alpha - \epsilon$, we release a job with processing time ϵ and at time $\alpha + \epsilon$, we release x jobs of processing time 0. The optimal preemptive completion time is $\alpha + x(\alpha + \epsilon) + 1 + \epsilon$, while the completion time of the nonpreemptive α-schedule is $\alpha + (\alpha + 1) + x(1 + \alpha)$. As x gets large and ϵ goes to 0, the ratio between the two goes to $1 + \frac{1}{\alpha}$. □

This theorem, in and of itself, always yields approximation bounds that are worse than 2.

We thus introduce a new fact that ultimately yields better algorithms. We will show that the makespan of an α-schedule is within a $(1 + \alpha)$-factor of the makespan of the corresponding preemptive schedule; in fact, we will prove a stronger result in Lemma 2.3 below. Thus the idle time introduced in the nonpreemptive schedules decreases as α is reduced from 1 to 0. On the other hand, the (worst-case) bound on the completion time of any specific job increases as α goes from 1 to 0. It is the balancing of these two effects that leads to better approximations. In the following discussion we do not assume that the preemptive schedule is the optimal preemptive schedule found using SRPT. In fact, our results on converting preemptive schedules to nonpreemptive schedules apply in general, but when we want to prove upper bounds on the performance ratio for total completion time, we assume that the preemptive schedule is an optimal preemptive schedule whose value is a lower bound on the value of any optimal nonpreemptive schedule.

Let $S_i^P(\beta)$ denote the set of jobs which complete exactly β fraction of their processing time before C_i^P in the schedule P (note that J_i is included in $S_i^P(1)$). We overload notation by using $S_i^P(\beta)$ to also denote the sum of processing times of all jobs in the set $S_i^P(\beta)$; the meaning should be clear from the context. Let T_i be the total idle time in P before J_i completes.

The preemptive completion time of J_i can be written as the sum of the idle time and fractional processing times of jobs that ran before C_i^P. This yields the following lemma.

LEMMA 2.2. $C_i^P = T_i + \sum_{0<\beta\leq 1} \beta S_i^P(\beta)$.

We next upper bound the completion time of a job J_i in the α-schedule.

LEMMA 2.3.

$$C_i^\alpha \leq T_i + (1 + \alpha) \sum_{\beta \geq \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta).$$

Proof. Let $J_1, \ldots, J_{i-1}$ be the jobs that run before J_i in the α-schedule. We will give a procedure which converts the preemptive schedule into a schedule in which

(C1) jobs $J_1, \ldots, J_i$ run nonpreemptively in that order,

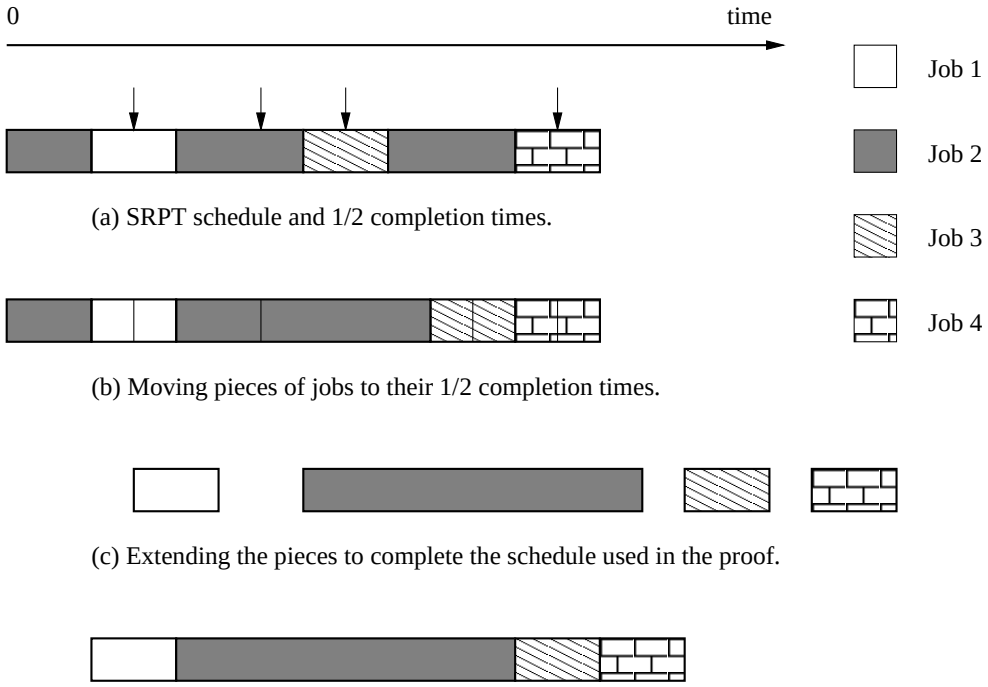
(C2) the remaining jobs run preemptively, and

(C3) the completion time of J_i obeys the bound given in the lemma.

Since the actual C_i^α is no greater than the completion time of J_i in this schedule, the lemma will be proven.

Splitting up the second term in the bound from Lemma 2.2, we get the following equation:

$$C_i^P = T_i + \sum_{\beta < \alpha} \beta S_i^P(\beta) + \sum_{\beta \geq \alpha} \alpha S_i^P(\beta) + \sum_{\beta \geq \alpha} (\beta - \alpha) S_i^P(\beta).$$



(a) SRPT schedule and 1/2 completion times.

(b) Moving pieces of jobs to their 1/2 completion times.

(c) Extending the pieces to complete the schedule used in the proof.

(d) The true 1/2-schedule.

FIG. 2.1. *Illustration of proof of Lemma* 2.3 *with* $\alpha = 1/2$ *and* $i = 4$.

Let $J^B = \bigcup_{\beta \geq \alpha} S_i^P(\beta)$ and $J^A = J - J^B$. We can interpret the four terms in the above equation as (1) the idle time in the preemptive schedule before C_i^P, (2) the pieces of jobs in J^A that ran before C_i^P, (3) for each job $J_j \in J^B$, the pieces of J_j that ran before $C_j^P(\alpha)$, and (4) for each job $J_j \in J^B$, the pieces of J_j that ran between $C_j^P(\alpha)$ and C_i^P. Let x_j be the β for which $J_j \in S_i^P(\beta)$, that is, the fraction of J_j that was completed before C_i^P. Then $\sum_{\beta \geq \alpha}(\beta - \alpha)S_i^P(\beta)$ can be rewritten as $\sum_{J_j \in J^B}(x_j - \alpha)p_j$. Observe that $(x_j - \alpha)p_j$ is the fraction of job J_j that ran between $C_j^P(\alpha)$ and C_i^P.

Let $J^C = \{J_1, \ldots, J_i\}$. Clearly J^C is a subset of J^B. Now think of schedule P as an ordered list of pieces of jobs (with sizes). For each $J_j \in J^C$ modify the list by (1) removing all pieces of jobs that run between $C_j^P(\alpha)$ and C_i^P and (2) inserting a piece of size $(x_j - \alpha)p_j$ at the point corresponding to $C_j^P(\alpha)$. In this list, we have pieces of size $(x_j - \alpha)p_j$ of jobs $J_1, \ldots, J_i$ in the correct order (plus other pieces of jobs). Now convert this ordered list back into a schedule by scheduling the pieces in the order of the list, respecting release dates. We claim that job i still completes at time C_i^P. To see this observe that the total processing time before C_i^P remains unchanged and that other than the pieces of size $(x_j - \alpha)p_j$, we moved pieces only later in time, so no additional idle time need be introduced.

Now, for each job $J_j \in J^C$, extend the piece of size $(x_j - \alpha)p_j$ to one of size p_j by adding $p_j - (x_j - \alpha)p_j$ units of processing and replace the pieces of J_j that occur earlier, of total size αp_j, by idle time. Figure 2.1 illustrates this transformation. We now have a schedule in which $J_1, \ldots, J_i$ are each scheduled nonpreemptively for p_j

units of time and in which the completion time of J_i is

$$\begin{aligned} C_i^P + \sum_{J_j \in J^C} (p_j - (x_j - \alpha)p_j) &\le C_i^P + \sum_{J_j \in J^B} (p_j - (x_j - \alpha)p_j) \\ &= C_i^P + \sum_{\beta \ge \alpha} (1 - \beta + \alpha) S_i^P(\beta) \\ &= T_i + (1 + \alpha) \sum_{\beta \ge \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta), \end{aligned}$$

where the second equality just comes from reindexing terms by β instead of j, and the third comes from plugging in the value of C_i^P from Lemma 2.2. To complete the proof, we observe that the remaining pieces in the schedule are all from jobs in $J - J^C$, and we have thus met the conditions (C1), (C2), and (C3) above. □

Although we will not use it directly, applying Lemma 2.3 to the last job to complete in the α-schedule yields the following corollary.

COROLLARY 2.4. *The makespan of the α-schedule is at most $(1 + \alpha)$ times the makespan of the corresponding preemptive schedule, and there are instances for which this bound is tight.*

Having analyzed completion times as in Lemma 2.3, we see that the approximation ratio is going to depend on the distribution of the different sets $S_i^P(\beta)$. To avoid the worst-case α, we choose α randomly according to some probability distribution. We now give a general bound on this algorithm, which we call RANDOM-α.

LEMMA 2.5. *Suppose α is chosen from a probability distribution over $(0, 1]$ with a density function f. Then for each job J_i, $E[C_i^\alpha] \le (1 + \delta)C_i^P$, where*

$$\delta = \max_{0 < \beta \le 1} \int_0^\beta \frac{1 + \alpha - \beta}{\beta} f(\alpha) \mathrm{d}\alpha.$$

It follows that $E[\sum_i C_i^\alpha] \le (1 + \delta) \sum_i C_i^P$.

Proof. We will show that the expected completion time of any job J_i is within $(1 + \delta)$ of its preemptive completion time. From Lemma 2.3 it follows that for any given α,

$$C_i^\alpha \le T_i + (1 + \alpha) \sum_{\beta \ge \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta).$$

Therefore, when α is chosen according to f, the expected completion time of J_i, $E[C_i^\alpha] = \int_0^1 f(\alpha) C_i^\alpha \mathrm{d}\alpha$, is bounded by

$$T_i + \int_0^1 f(\alpha) \left((1 + \alpha) \sum_{\beta \ge \alpha} S_i^P(\beta) + \sum_{\beta < \alpha} \beta S_i^P(\beta) \right) \mathrm{d}\alpha$$

since T_i is independent of α. We now bound the second term in the above expression:

$$\begin{aligned}
&\int_0^1 f(\alpha)\Big((1+\alpha)\sum_{\beta\geq\alpha} S_i^P(\beta) + \sum_{\beta<\alpha}\beta S_i^P(\beta)\Big)\mathrm{d}\alpha \\
&= \sum_{0<\beta\leq 1} S_i^P(\beta)\left(\int_0^\beta (1+\alpha)f(\alpha)\mathrm{d}\alpha + \int_\beta^1 \beta f(\alpha)\mathrm{d}\alpha\right) \\
&= \sum_{0<\beta\leq 1} \beta S_i^P(\beta)\left(1+\int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha)\mathrm{d}\alpha\right) \\
&\leq \left(1+\max_{0<\beta\leq 1}\int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha)d\alpha\right)\sum_{0<\beta\leq 1}\beta S_i^P(\beta) \\
&\leq (1+\delta)\sum_{0<\beta\leq 1}\beta S_i^P(\beta).
\end{aligned}$$

It follows that

$$E[C_i^\alpha] \leq T_i + (1+\delta)\sum_{0<\beta\leq 1}\beta S_i^P(\beta) \leq (1+\delta)C_i^P.$$

Using linearity of expectations, it is easy to show that the expected total completion time of the schedule is within $(1+\delta)$ of the preemptive schedule's total completion time. □

With Lemma 2.5 in place, we can simply choose different PDFs to establish different bounds.

THEOREM 2.6. *For the problem of scheduling to minimize weighted completion time with release dates,* RANDOM-α *performs as follows:*

1. *If α is chosen uniformly in $(0,1]$, the expected approximation ratio is at most 2.*
2. *If α is chosen to be 1 with probability $3/5$ and $1/2$ with probability $2/5$, the expected approximation ratio is at most 1.8.*
3. *If α is chosen from $(0,1]$ according to the density function $f(\alpha) = \frac{e^\alpha}{e-1}$, the expected approximation ratio is at most $\frac{e}{e-1} \approx 1.58$.*

Proof.

1. Choosing α uniformly corresponds to the PDF $f(\alpha) = 1$. Plugging into the bound from Lemma 2.5, we get an approximation ratio of

$$\begin{aligned}
1+\max_{0<\beta\leq 1}\int_0^\beta \frac{1+\alpha-\beta}{\beta}\mathrm{d}\alpha &= 1+\max_{0<\beta\leq 1}\frac{1}{\beta}\left((1-\beta)\beta+\frac{\beta^2}{2}\right) \\
&= 1+\max_{0<\beta\leq 1}\left(1-\frac{\beta}{2}\right) \\
&\leq 2.
\end{aligned}$$

2. Omitted.
3. If $f(\alpha) = \frac{e^\alpha}{e-1}$, then

$$\begin{aligned}
\max_{0<\beta\leq 1}\int_0^\beta \left(\frac{1+\alpha-\beta}{\beta}\right)\left(\frac{e^\alpha}{e-1}\right)\mathrm{d}\alpha &= \max_{0<\beta\leq 1}\frac{1}{\beta(e-1)}\Big(((1-\beta)+(\beta-1))e^\beta \\
&\qquad - ((1-\beta)-1)\Big) \\
&= \max_{0<\beta\leq 1}\frac{1}{e-1}
\end{aligned}$$

$$= \frac{1}{e-1}.$$

Therefore

$$1 + \max_{0<\beta\le 1} \int_0^\beta \left(\frac{1+\alpha-\beta}{\beta}\right)\left(\frac{e^\alpha}{e-1}\right) \mathrm{d}\alpha \le \frac{e}{e-1}. \qquad \square$$

It can be shown that the density function $\frac{e^\alpha}{e-1}$ minimizes the expression $\max_{0<\beta\le 1} \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha)\mathrm{d}\alpha$ over all choices of $f(\alpha)$. In the off-line setting, rather than choosing α randomly, we can try different values of α and choose the one that yields the best schedule. We call the algorithm which computes the schedule of value $\min_\alpha \sum_j C_j^\alpha$, BEST-$\alpha$.

COROLLARY 2.7. *Algorithm* BEST-α *is an* $e/(e-1)$*-approximation algorithm for nonpreemptive scheduling to minimize average completion time on one machine with release dates. It runs in* $O(n^2)$ *time.*

Proof. The approximation bound follows from Theorem 2.6. For the running time, we observe that given a preemptive SRPT schedule we can efficiently determine the best possible choice of α. The SRPT schedule preempts only at release dates. Thus it has at most $n-1$ preemptions and there are at most n "combinatorially distinct" values of α for a given preemptive schedule. The SRPT schedule can be computed in $O(n \log n)$ time using a simple priority queue and given that schedule and an α, the corresponding α-schedule can be computed in linear time by a simple scan. $\square$

In the on-line setting, we cannot implement BEST-α. However, if we choose α randomly we get the following theorem.

THEOREM 2.8. *There is a polynomial-time randomized on-line algorithm with an expected competitive ratio* $e/(e-1)$ *for the problem of minimizing total completion time in the presence of release dates.*

Proof. The randomized on-line algorithm is the following. The algorithm picks an $\alpha \in (0,1]$ at random according to the density function $f(x) = \frac{e^x}{e-1}$ before receiving any input (this is the only randomness used in the algorithm). The algorithm simulates the on-line preemptive SRPT schedule. At the exact time when a job finishes α fraction of its processing time in the simulated SRPT schedule, it is added to the queue of jobs to be executed nonpreemptively. The nonpreemptive schedule is obtained by executing jobs in the strict order of their insertion into the queue while respecting the insertion times into the queue. Observe that this rule leads to a valid on-line nonpreemptive schedule and that in fact the order of the jobs scheduled is exactly the same as in the α-schedule. The schedule respects the insertion times; therefore no job is executed in the nonpreemptive schedule before its α point in the SRPT schedule. To show the bound on the expected competitive ratio, we claim that the bounds in Lemma 2.3 (and hence Theorem 2.6 also) hold for the nonpreemptive schedule created by the on-line algorithm. The main observation is that the proof of Lemma 2.3 does not use the true α-schedule but a weaker one in which for every job J_i the first α fraction of its processing time in the SRPT schedule is left as idle time. A careful examination of the proof of Lemma 2.3 with Figure 2.1 as an illustration makes this clear. $\square$

We also give some negative results for the various algorithms.

THEOREM 2.9. *For the problem of scheduling to minimize weighted completion time with release dates,* RANDOM-α *performs as follows:*

1. *If* α *is chosen uniformly, the expected approximation ratio is at least* 2.
2. *For the* BEST-α *algorithm, the approximation ratio is at least* 4/3.

Proof. We will use a set of parameterized instances to show all the above bounds. We define an instance $I(\delta, n)$ as follows. At time 0 a job of size 1 is released and at time $\delta < 1$, n jobs of size 0 are released (we use zero length jobs for ease of exposition). The optimal preemptive schedule for this instance has a total completion time of $1+n\delta$. The optimal nonpreemptive schedule for this instance can be obtained by first completing all the small jobs and running the large job after them for a total completion time of $1+\delta+n\delta$. It is easy to see that there are only two combinatorially distinct schedules corresponding to the values of $\alpha \leq \delta$ and $\alpha > \delta$ and we can restrict our attention to those two schedules and the probability with which they are chosen. Let $S1$ and $S2$ be the two schedules and $C1$ and $C2$ be their total completion times, respectively. It is easy to see that $C1 = 1 + n$ and $C2 = 1 + \delta + n\delta$.

1. If α is chosen uniformly at random, $S1$ is chosen with probability δ and $S2$ is chosen with probability $(1 - \delta)$ and a simple calculation shows that if we choose $n \gg 1$ and $1 \gg \delta$, the expected approximation ratio approaches 2.
2. Consider an instance I in which in addition to the jobs of $I(1/2, n)$ we release n more jobs of size 0 at time 1. The optimal preemptive schedule for I consists of the preemptive schedule for $I(1/2, \epsilon, n)$ followed by the additional n small jobs. The completion time of the optimal preemptive schedule is term $1+3n/2$. An optimal nonpreemptive schedule schedules the large job after all the small jobs and has a total completion time $2+3n/2$. It is easy to see that there are only two combinatorially distinct α-schedules, one corresponding to $\alpha \leq 1/2$ and the other corresponding to $\alpha > 1/2$. In both cases it is easy to verify that the completion time of the schedule is $1 + 2n$. Thus the approximation ratio of the BEST-α cannot be better than 4/3. □

After learning of our results, Stougie and Vestjens [33] improved the lower bound for randomized on-line algorithms to $e/(e-1)$. This implies that our randomized on-line algorithm is optimal. Torng and Uthaisombut [34] have shown that there are instances on which the approximation ratio of BEST-α can be made arbitrarily close to $\frac{e}{e-1}$. This improves our lower bound of 4/3 on BEST-α's performance and also implies that our upper bound analysis is tight.

3. Parallel machine scheduling with release dates. We now turn to the problem of minimizing average completion time on parallel machines in the presence of release dates. In this section, we give a simple 3-approximation algorithm for the problem that is also an on-line algorithm. Our algorithm does not use linear programming or slow dynamic programming. It introduces the notion of a one-machine preemptive relaxation. In the next section, we will show how to improve this to a 2.83-approximation algorithm using more involved techniques.

Given an instance I for nonpreemptive scheduling on m machines, we define a one-machine preemptive relaxation $I1$ as follows. $I1$ has the same set of jobs as those of I and has one machine. The processing time of J_j in $I1$ is $p'_j = p_j/m$ and release date is $r'_j = r_j$.

LEMMA 3.1. *The value of an optimal solution to $I1$ is a lower bound on the value of an optimal solution to I.*

Proof. We show how to convert a feasible schedule N, for input I, to a feasible schedule $P1$, for input $I1$, without increasing the average completion time. Take any schedule N and consider a particular time unit t that is sufficiently small. Let the $k \leq m$ jobs that are running during that time be $J_1, \ldots, J_k$. In $P1$, at time t, run $1/m$ units of each of jobs $J_1, \ldots, J_k$, in arbitrary order. The completion time of job J_j in $P1$, C_j^{P1} is clearly no greater than C_j^N, the completion time of J_j in N. □

Given an optimal preemptive schedule $P1$ for $I1$, we form a list schedule N by ordering jobs by C_j^{P1} and then scheduling them nonpreemptively in that order, respecting release dates. Let C_j^* be the completion time of J_j in an optimal schedule for I. $I1$ may be a bad relaxation in the sense that $\sum_j C_j^{P1}$ may be much less than $\sum_j C_j^*$. However, we can still use this relaxation to obtain a good nonpreemptive schedule.

LEMMA 3.2. *The nonpreemptive list schedule N satisfies $\sum_j C_j^N \le (3-\frac{1}{m})\sum_j C_j^*$.*

Proof. We focus on a particular job J_j. For convenience, we assume that the jobs are ordered according to their completion times in $P1$. Thus J_j is the jth job to complete in $P1$. We now derive three lower bounds on C_j^{P1}. First, we have the trivial bound $C_j^{P1} \ge r'_j + p'_j$. Further, C_j^{P1} is at least as big as the processing times of the jobs that precede it. Therefore

$$C_j^{P1} \ge \sum_{k=1}^{j} p'_k = \sum_{k=1}^{j} \frac{p_k}{m}. \tag{3.1}$$

Let $r_j^{\max} = \max_{1\le k\le j} r'_k$ be the latest release date among jobs that complete before j; then $C_j^{P1} \ge r_j^{\max}$.

Now consider the list schedule N. Clearly by time $r_j^{\max}$ all jobs $J_1, \ldots, J_j$ have been released. Even if no job starts before time $r_j^{\max}$, by standard makespan arguments J_j will complete by

$$\begin{aligned} C_j^N &\le r_j^{\max} + \sum_{k=1}^{j-1} \frac{p_k}{m} + p_j \\ &\le C_j^{P1} + C_j^{P1} + p_j\left(1 - \frac{1}{m}\right), \end{aligned} \tag{3.2}$$

where the second inequality follows from (3.1) and $C_j^{P1} \ge r_j^{\max}$ above. Summing (3.2) over all jobs, we get a total completion time of

$$\sum_j C_j^N \le 2\sum_j C_j^{P1} + \left(1 - \frac{1}{m}\right)\sum_j p_j. \tag{3.3}$$

By Lemma 3.1, $\sum_j C_j^{P1} \le \sum_j C_j^*$, and trivially the optimal solution to I must have total completion time $\sum_j C_j^* \ge \sum p_j$; therefore this algorithm is a $(3 - \frac{1}{m})$-approximation. ∎

The running time is just the time to run SRPT on the one-machine relaxation and the time to list schedule for a total of $O(n \log n)$. This algorithm can be made on-line by simulating the preemptive schedule and adding a job to the list when it completes in the preemptive schedule.

4. A general conversion algorithm. In this section we develop a technique to obtain parallel machine schedules from one-machine schedules that works even when jobs have precedence constraints and release dates. Given an average weighted completion time scheduling problem, we show that if we can approximate the one-machine preemptive variant, then we can also approximate the m-machine nonpreemptive variant with a slight degradation in the quality of approximation.

Precedence constraints will be represented in the usual way by a directed acyclic graph (DAG) whose vertices correspond to jobs and whose edges represent precedence constraints.

In this section, we use a slightly different one-machine relaxation from the previous section; namely, we do not divide the processing times by m. We use the superscript m to denote the number of machines; thus S^m denotes a schedule for m machines, C^m denotes the sum of weighted completion time of S^m, and C^m_j denotes the completion time of job J_j under schedule S^m. The subscript OPT refers to an optimal schedule; thus an optimal schedule is denoted by S^m_{OPT}, and its weighted completion time is denoted by C^m_{OPT}. For a set of jobs A, $p(A)$ denotes the sum of processing times of jobs in A.

DEFINITION 4.1. *For any vertex j, recursively define the quantity κ_j as follows. For a vertex j with no predecessors $\kappa_j = p_j + r_j$. Otherwise define $\kappa_j = p_j + \max\{\max_{i \prec j} \kappa_i, r_j\}$. Any path P_{ij} from i to j where $p(P_{ij}) = \kappa_j$ is referred to as a* critical path *to j.*

4.1. Conversion algorithm DELAY LIST. We now describe the DELAY LIST algorithm. Given a one-machine schedule which is a ρ-approximation, DELAY LIST produces a schedule for $m \geq 2$ machines whose value is within a factor $(k_1\rho + k_2)$ of the optimal m-machine schedule, where k_1 and k_2 are small constants. We will describe a variant of this scheduling algorithm which yields $k_1 = (1 + \beta)$ and $k_2 = (1 + 1/\beta)$ for any $\beta > 0$. Therefore, for cases where we can find optimal one-machine schedules (trees and series-parallel without release dates), we obtain a 4-approximation for m machines by setting $\beta = 1$. To our knowledge, these are the best results for these special cases.

The main idea is as follows. The one-machine schedule taken as a list (jobs in order of their completion times in the schedule) provides some priority information on which jobs to schedule earlier.[2] Unlike with makespan, the completion time of every job is important for weighted completion time. When trying to convert the one-machine schedule into an m-machine one, precedence constraints prevent complete parallelization. Thus we may have to execute jobs out-of-order from the list to benefit from parallelism. If all p_i are identical (say 1), we can afford to use naive list scheduling.[3] If there is an idle machine and we schedule some available job on it, it is not going to delay jobs which become available soon, since it completes in one time unit. On the other hand, if not all p_i's are the same, a job could keep a machine busy, delaying more profitable jobs that become available soon. At the same time, we cannot afford to keep machines idle. We strike a balance between the two extremes: schedule a job out-of-order only if there has been enough idle time already to justify scheduling it. To measure whether there has been enough idle time, we introduce a charging scheme.

Assume, for ease of exposition, that all processing times are integers and that time is discrete. This restriction can be removed without much difficulty and we use it only in the interests of clarity and intuition. A job is *ready* if it has been released and all its predecessors are done.

DEFINITION 4.2. *The time at which job J_i is ready in a schedule S is denoted by q^S_i and the time at which it starts is denoted by s^S_i.*

We use S^m to denote the m-machine schedule that our algorithm constructs and for ease of notation the superscript m will be used in place of S^m to refer to quantities

[2] In the rest of the paper we assume without loss of generality that a *list* obeys the precedence constraints; that is, if $i \prec j$, then i comes earlier in the list than j.

[3] In this section, by *list scheduling* we mean the algorithm which schedules the first available job in the list if a machine is free. This is in contrast to another variant considered in earlier sections in which jobs are scheduled strictly in the order of the list.

of interest in this schedule. Let $\beta > 0$ be some constant. At each discrete time step t, the algorithm applies one of the following three cases:

1. *There is an idle machine M and the first job J_j on the list is ready at time t*—schedule J_j on M and charge all uncharged idle time in the interval (q_j^m, s_j^m) to J_j.
2. *There is an idle machine and the first job J_j in the list is not ready at t, but there is another ready job on the list*—focusing on the job J_k which is the first in the list among the ready jobs, schedule it if there is at least βp_k *uncharged* idle time among all machines and charge βp_k idle time to J_k.
3. *There is no idle time or the above two cases do not apply*—do not schedule any job; merely increment t.

DEFINITION 4.3. *A job is said to be scheduled* in order *if it is scheduled when it is at the head of the list. Otherwise it is said to be scheduled* out of order. *The set of jobs which are scheduled before a job J_i but which come later in the list than J_i is denoted by O_i. The set of jobs which come after J_i in the list is denoted by A_i and those which come before J_i by B_i (includes J_i).*

DEFINITION 4.4. *For each job J_i, define a path $P_i' = J_{j_1}, J_{j_2}, \ldots, J_{j_\ell}$, with $J_{j_\ell} = J_i$ with respect to the schedule S^m as follows. The job J_{j_k} is the predecessor of $J_{j_{k+1}}$ with the largest completion time (in S^m) among all the predecessors of $J_{j_{k+1}}$ such that $C_{j_k}^m \geq r_{j_{k+1}}$; ties are broken arbitrarily. J_{j_1} is the job where this process terminates when there are no predecessors which satisfy the above condition. The jobs in P_i' define a disjoint set of time intervals $(0, r_{j_1}], (s_{j_1}^m, C_{j_1}^m], \ldots, (s_{j_\ell}^m, C_{j_\ell}^m]$ in the schedule. Let κ_i' denote the sum of the lengths of the intervals.*

FACT 4.5. $\kappa_i' \leq \kappa_i$.

FACT 4.6. *The idle time charged to each job J_i is less than or equal to βp_i.*

Proof. The fact is clear if idle time is charged to J_i according to case 2 in the description of our algorithm. Suppose case 1 applies to J_i. Since J_i was ready at q_i^m and was not scheduled according to case 2 earlier, the idle time in the interval (q_i^m, s_i^m) that is charged to J_i is less than βp_i. We remark that the algorithm with discrete time units might charge more idle time due to integrality of the time unit. However, that is easily fixed in the continuous case where we schedule J_i at the first time instant when at least βp_i units of uncharged idle time have accumulated. ∎

A crucial feature of the algorithm is that when it schedules jobs, it considers only the first job in the list that is ready, even if there is enough idle time for other ready jobs that are later in the list. The proof of the following lemma makes use of this feature.

LEMMA 4.7. *For every job J_i, there is no uncharged idle time in the time interval (q_i^m, s_i^m), and furthermore all the idle time is charged only to jobs in B_i.*

Proof. By the preceding remarks, it is clear that no job in A_i is started in the time interval (q_i^m, s_i^m) since J_i was ready at q_i^m. From this we can conclude that there is no idle time charged to jobs in A_i in that time interval. Since J_i is ready at q_i^m and was not scheduled before s_i^m, from cases 1 and 2 in the description of our algorithm there cannot be any uncharged idle time. ∎

The following lemma shows that for any job J_i, the algorithm does not schedule too many jobs from A_i before scheduling J_i itself.

LEMMA 4.8. *For every job J_i, the total idle time charged to jobs in A_i, in the interval $(0, s_i^m)$, is bounded by $m(\kappa_i' - p_i)$. It follows that $p(O_i) \leq m(\kappa_i' - p_i)/\beta \leq m(\kappa_i - p_i)/\beta$.*

Proof. Consider a job J_{j_k} in P_i'. The job $J_{j_{k+1}}$ is ready to be scheduled at the

completion of J_{j_k}, that is, $q^m_{j_{k+1}} = C^m_{j_k}$. From Lemma 4.7, it follows that in the time interval between $(C^m_{j_k}, s^m_{j_{k+1}})$ there is no idle time charged to jobs in $A_{j_{k+1}}$. Since $A_{j_{k+1}} \supset A_i$ it follows that all the idle time for jobs in A_i has to be accumulated in the intersection between $(0, s^m_i)$ and the time intervals defined by P'_i. This quantity is clearly bounded by $m(\kappa'_i - p_i)$. The second part follows since the total processing time of the jobs in O_i is bounded by $1/\beta$ times the total idle time that can be charged to jobs in A_i (recall that $O_i \subseteq A_i$). □

THEOREM 4.9. *Let S^m be the schedule produced by the algorithm* DELAY LIST *using a list S^1. Then for each job J_i, $C^m_i \le (1+\beta)p(B_i)/m + (1+1/\beta)\kappa'_i - p_i/\beta$.*

Proof. Consider a job J_i. We can split the time interval $(0, C^m_i)$ into two disjoint sets of time intervals T_1 and T_2 as follows. The set T_1 consists of all the disjoint time intervals defined by P'_i. The set T_2 consists of the time intervals obtained by removing the intervals in T_1 from $(0, C^m_i)$. Let t_1 and t_2 be the sum of the times of the intervals in T_1 and T_2, respectively. From the definition of T_1, it follows that $t_1 = \kappa'_i \le \kappa_i$. From Lemma 4.7, in the time intervals of T_2, all the idle time is either charged to jobs in B_i, and the only jobs which run are from $B_i \cup O_i$. From Fact 4.6, the idle time charged to jobs in B_i is bounded by $\beta p(B_i)$. Therefore the time t_2 is bounded by $(\beta p(B_i) + p(B_i) + p(O_i))/m$. Using Lemma 4.8 we see that $t_1 + t_2$ is bounded by $(1+\beta)p(B_i)/m + (1+1/\beta)\kappa'_i - p_i/\beta$. □

4.2. One-machine relaxation. In order to use DELAY LIST, we will need to start with a one-machine schedule. The following two lemmas provide lower bounds on the optimal m-machine schedule in terms of the optimal one-machine schedule. This one-machine schedule can be either preemptive or nonpreemptive; the bounds hold in either case.

LEMMA 4.10. $C^m_{\mathrm{OPT}} \ge C^1_{\mathrm{OPT}}/m$.

Proof. Given a schedule S^m on m machines with total weighted completion time C^m, we will construct a one-machine schedule S^1 with total weighted completion time at most mC^m as follows. Order the jobs according to their completion times in S^m with the jobs completing early coming earlier in the ordering. This ordering is our schedule S^1. Note that there could be idle time in the schedule due to release dates. If $i \prec j$, then $C^m_i \le s^m_j \le C^m_j$ which implies that there will be no precedence violations in S^1. We claim that $C^1_i \le mC^m_i$ for every job J_i. Let P be the sum of the processing times of all the jobs which finish before J_i (including J_i) in S^m. Let I be the total idle time in the schedule S^m before C^m_i. It is easy to see that $mC^m_i \ge P + I$. We claim that $C^1_i \le P + I$. The idle time in the schedule S^1 can be charged to idle time in the schedule S^m and P is the sum of all jobs which come before J_i in S^1. This implies the desired result. □

LEMMA 4.11. $C^m_{\mathrm{OPT}} \ge \sum_i w_i\kappa_i = C^\infty_{\mathrm{OPT}}$.

Proof. The length of the critical path κ_i is an obvious lower bound on the completion time C^m_i of job J_i. Summing up over all jobs gives the first inequality. It is also easy to see that if the number of machines is unbounded, every job J_i can be scheduled at the earliest time it is available and will finish by κ_i yielding the equality. □

4.3. Obtaining generic *m*-machine schedules. In this section we derive our main theorem relating m-machine schedules to one-machine schedules.

We begin with a corollary to Theorem 4.9.

COROLLARY 4.12. *Let S^m be the schedule produced by the algorithm* DELAY LIST *using a one-machine schedule S^1 as the list. Then for each job J_i, $C^m_i \le$*

$(1+\beta)C_i^1/m + (1+1/\beta)\kappa_i$.

Proof. Since all jobs in B_i come before J_i in the one-machine schedule, it follows that $p(B_i) \leq C_i^1$. Plugging this and Fact 4.5 into the bound in Theorem 4.9, we conclude that $C_i^m \leq (1+\beta)C_i^1/m + (1+1/\beta)\kappa_i$. ∎

THEOREM 4.13. *Given an instance I of scheduling to minimize sum of weighted completion times and a one-machine schedule for I that is within a factor ρ of an optimal one-machine schedule,* DELAY LIST *gives an m-machine schedule for I that is within a factor $(1+\beta)\rho + (1+1/\beta)$ of an optimal m-machine schedule.*

Proof. Let S^1 be a schedule which is within a factor ρ of the optimal one-machine schedule. Then $C^1 = \sum_i w_i C_i^1 \leq \rho C_{\mathrm{OPT}}^1$. By Corollary 4.12, the schedule created by the algorithm DELAY LIST satisfies

$$\begin{aligned} C^m &= \sum_i w_i C_i^m \\ &\leq \sum_i w_i \left((1+\beta)\frac{C_i^1}{m} + \left(1+\frac{1}{\beta}\right)\kappa_i \right) \\ &= \frac{1+\beta}{m} \sum_i w_i C_i^1 + \left(1+\frac{1}{\beta}\right) \sum_i w_i \kappa_i. \end{aligned}$$

From Lemmas 4.10 and 4.11 it follows that

$$\begin{aligned} C^m &\leq \frac{(1+\beta)\rho C_{\mathrm{OPT}}^1}{m} + \left(1+\frac{1}{\beta}\right) C_{\mathrm{OPT}}^{\infty} \\ &\leq \left((1+\beta)\rho + \left(1+\frac{1}{\beta}\right) \right) C_{\mathrm{OPT}}^m. \end{aligned}$$ ∎

COROLLARY 4.14. *There is an $O(n \log n)$ time 4-approximation algorithm for weighted completion time on parallel machines when the precedence graphs are restricted to be series-parallel graphs.*

Proof. The optimal single machine schedule with release dates ignored can be computed in $O(n \log n)$ time for series-parallel graphs [1]. Applying the DELAY LIST algorithm with $\beta = 1$ to this schedule gives the desired result. ∎

REMARK 4.15. *Since the bounds in our conversion algorithm are job-by-job, the algorithm is applicable to a more general class of metrics as well.*

There is an interesting property of the conversion algorithm that is useful in its applications and worth pointing out explicitly. We explain it via an example. Suppose we want to compute an m-machine schedule with release dates and precedence constraints. From Theorem 4.13 it would appear that we need to compute a one-machine schedule for the problem that has both precedence constraints and release dates. However, we can completely ignore the release dates in computing the one-machine schedule S^1! This follows from a careful examination of the upper bound proved in Theorem 4.9 and the proof of Theorem 4.13. This is useful since the approximation ratio for the problem $1|prec|\sum_j w_j C_j$ is 2 [16] while it is 3 for $1\,|\,prec, r_j\,|\,\sum_j w_j C_j$ [16]. In another example, the problem $1\,||\,\sum_j w_j C_j$ has a very simple polynomial-time algorithm using Smith's ratio rule while $1\,|\,r_j\,|\,\sum_j w_j C_j$ is $\mathcal{NP}$-hard. Thus release dates play a role only in the conversion algorithm and not in the single machine schedule. A similar claim can be made when there are delays between jobs. In this setting a positive delay d_{ij} between jobs i and j indicates that i is a predecessor of j and that j cannot start until d_{ij} time units after i completes. We can generalize our conversion algorithm and its analysis to handle delays

and obtain the same results as those in Theorems 4.9 and 4.13. The only change required is in the definition of ready time of a job which now depends also on the delay after a predecessor finishes. As with release dates we can ignore the delay values (not the precedence constraints implied by them though) in computing the single machine schedule. Munier, Queyranne, and Schulz [24] use linear programming ideas to generalize results for problems with precedence constraints to those with delay constraints.

4.4. Applying conversion to in-tree precedence. We obtain stronger results for in-tree precedence *without* release dates. The problem is strongly $\mathcal{NP}$-hard even for this case. We analyze the standard list scheduling algorithm which starts with an ordering on the jobs (the list) and greedily schedules each successive job in the list at the earliest possible time. We use the optimal one-machine schedule for trees as the list. We show that this algorithm gives a 2-approximation for in-trees. Recall that s_i^m is the start time of J_i in the schedule S^m.

LEMMA 4.16. *If S^m is the list schedule using a one-machine schedule S^1 as the list, then for any job J_i, $C_i^m \leq \kappa_i + C_i^1/m$.*

Proof. Since there are no release dates, we can assume that the schedule S^1 has no idle time. Without loss of generality assume that $J_1, \ldots, J_n$ is the ordering of the jobs ordered according to their *start* times s_i^m in S^m (we break ties arbitrarily). We will prove the lemma by induction on i. We strengthen the hypothesis by adding the following invariant. If $C_i^m > C_j^m + p_i$, where J_j is the last predecessor of J_i to finish in S^m, then all the jobs scheduled before s_i^m in S^m are ahead of J_i in the list S^1 and there is no idle time in the schedule before time s_i^m. In this case it follows that $C_i^m \leq p_i + C_i^1/m$. The base case is trivial since $\kappa_1 = p_1$ and the first job finishes at time p_1. Suppose that the hypothesis holds for all jobs J_k, $k < i$; we will prove it holds for J_i. If J_i has no predecessor it is easily seen that there is no idle time before J_i is scheduled and that $C_i^m \leq p_i + C_i^1/m$. Among the predecessors of i, let J_j, $j < i$ be the last to finish in the schedule S^m (ties are broken arbitrarily). We consider two cases.

1. $C_i^m = C_j^m + p_i$. By the hypothesis, $C_j^m \leq \kappa_j + C_j^1/m$. It follows that $C_i^m \leq \kappa_i + C_i^1/m$ since $\kappa_i \geq \kappa_j + p_i$ and $C_j^1 < C_i^1$.
2. $C_i^m > C_j^m + p_i$. Let $t = C_j^m$. Let P be the set of jobs which finish exactly at time t and P' be the set of jobs which had their last predecessor running until time t. Note that $J_i \in P'$, $J_j \in P$, and all the jobs in P' are ready to be run at time t. In an in-tree a node has at most one immediate successor; therefore $|P'| \leq |P|$. Therefore the number of jobs that are ready at t but were not ready at t^- is at most $|P|$. If J_i was not scheduled at t there must exist a job $J_l \notin P'$ which is scheduled at t. This implies that J_l occurs before J_i in the list S^1. Since no immediate predecessor of J_l finished at t, by the induction hypothesis we conclude that there was no idle time and no job which comes later than J_l in S^1 is scheduled before time t. Since J_i was ready at time t, it follows that there is no idle time and no job later than J_i in S^1 is scheduled between time t and the s_i^m. From these observations it follows that $C_i^m \leq p_i + C_i^1/m \leq \kappa_i + C_i^1/m$.

In both cases we see that the induction hypothesis is established for J_i and this finishes the proof. □

THEOREM 4.17. *There is an $O(n \log n)$-time algorithm with approximation ratio 2 for minimizing weighted completion time on m machines for in-tree precedence without release dates.*

Proof. The proof is similar to that of Theorem 4.13 except that we use the stronger bounds from Lemma 4.16. The running time is dominated by the time to compute the optimal one-machine schedule which can be done in $O(n \log n)$ time [1]. ∎

4.5. A 2.83-approximation for scheduling without precedence constraints. We now improve the approximation bound for parallel machine scheduling with release dates to 2.83 which improves the earlier ratio of $2.89 + \epsilon$ [3]. We combine ideas of the one-machine relaxation developed in section 3 and the idea of using delay based list scheduling to derive an alternate algorithm which has worse ratio than the algorithm in section 3. However, we observe that the bounds we get from the analysis of these two algorithms can be combined to get an improved lower bound on the optimal which leads to the improvement.

Recall from section 3 that $I1$ is the one-machine relaxation for a given instance I and $P1$ is an optimal preemptive schedule for $I1$.

LEMMA 4.18. *If we apply* DELAY LIST *to* $P1$ *with parameter* β*, the resulting schedule* D *has total completion time*

$$\sum C_j^D \leq (2+\beta)C_j^* + \frac{1}{\beta}\sum_j r_j.$$

Proof. We focus on a particular job J_j. From Theorem 4.9 and Fact 4.5 we conclude that $C_j^D \leq (1+\beta)p(B_j)/m + (1+1/\beta)\kappa_j - p_j/\beta$. Since we do not have precedence constraints on the jobs, $\kappa_j = r_j + p_j$. From the definition of B_j and the fact that the list is the order in which jobs finish in $P1$, it follows that $p(B_j)/m \leq C_j^{P1}$. We therefore conclude that $C_j^D \leq (1+\beta)C_j^{P1} + p_j + r_j/\beta$. Summing this over all jobs we obtain

$$\sum_j C_j^D \leq (1+\beta)\sum_j C_j^{P1} + \sum_j p_j + \frac{1}{\beta}\sum_j r_j.$$

Since both $\sum_j C_j^{P1}$ and $\sum_j p_j$ are lower bounds on the optimal schedule value, it follows that

$$\sum_j C_j^D \leq (2+\beta)\sum_j C_j^* + \frac{1}{\beta}\sum_j r_j. \qquad \blacksquare$$

We now balance the two algorithms, list scheduling and DELAY LIST, to achieve an approximation ratio of 2.83.

LEMMA 4.19. *For any input* I*, either list scheduling from* $P1$ *or using* DELAY LIST *on* $P1$ *for an appropriate choice of* β *produces a schedule with* $\sum_j C_j \leq 2.83\sum_j C_j^*$ *and runs in* $O(n \log n)$ *time.*

Proof. By (3.3), we know that

$$\sum_j C_j^N \leq 2\sum_j C_j^{P1} + \sum_j p_j. \tag{4.1}$$

If, for some α, we know that $\sum_j p_j \leq \alpha \sum_j C_j^*$, then the list scheduling algorithm is a $(2+\alpha)$-approximation algorithm.

Now consider the case when $\sum_j p_j > \alpha \sum_j C_j^*$. If we combine this equation with the simple bound that $\sum_j C_j^* \geq \sum_j (p_j + r_j)$, we get

$$\sum_j r_j \leq (1-\alpha)\sum_j C_j^*. \tag{4.2}$$

We can now plug (4.2) into the upper bound on C_j^D from Lemma 4.18 to get

$$\sum_j C_j^D \leq \left(2 + \beta + \frac{1-\alpha}{\beta}\right) \sum_j C_j^*.$$

We do not know the value of α, but for each possible α we can choose the β that minimizes the two terms. Simple algebra and calculus show that given α, we can choose β to be $\sqrt{(1-\alpha)}$. The expression $\min\{2+\alpha, 2+2\sqrt{1-\alpha}\}$ is minimized when $\alpha = 2\sqrt{2} - 2$ thus yielding a ratio of $2\sqrt{2} \simeq 2.83$.

To obtain the guaranteed approximation the algorithm runs both the list scheduling algorithm and the DELAY LIST algorithm with $\beta = \sqrt{3 - 2\sqrt{2}}$ and chooses the better of the two schedules. The schedules can be computed in $O(n \log n)$ time each. □

5. Conclusions. As mentioned earlier, many variants of the problem of minimizing average weighted completion time were shown to have constant factor approximations. Hoogeveen, Schuurman, and Woeginger [18] investigated the hardness of approximation of average completion time scheduling and in particular showed that the problems $P|prec, p_j = 1|\sum_j C_j$ and $R|r_j|\sum_j C_j$ are APX-hard; in other words, they do not admit a polynomial-time approximation scheme (PTAS) unless $P = NP$. Recently, much progress was made in obtained improved upper bounds as well. Several groups of authors obtained efficient PTASs for problems involving only release dates. A preliminary version describing these results is [2]. The maximal cases that were shown to have a PTAS are $P|r_j|\sum_j w_j C_j$, $Rm|r_j|\sum_j w_j C_j$, and their preemptive versions. An interesting open problem is the complexity of $1|prec|\sum_j w_j C_j$. A 2-approximation for this problem is known but no APX-hardness has been established. Subsequent to the linear programming based methods [16], simple combinatorial algorithms [6, 8] were developed for this problem matching the approximation ratio of 2. By coupling these algorithms with the DELAY LIST algorithm in this paper we obtain the first efficient and combinatorial approximation algorithms even with precedence constraints and delays. The ratio achieved for $P|r_j, prec|\sum_j w_j C_j$ is 5.83 and is worse than the currently best known ratio of 4 [24]. However, the algorithm in [24] is based on solving a linear program via the ellipsoid method.

Acknowledgments. We thank Javed Aslam, Moses Charikar, Cindy Phillips, David Shmoys, Santosh Vempala, and Joel Wein for valuable discussions. We are also grateful to the two anonymous referees for their comments, corrections, and suggestions for improvements.

REFERENCES

[1] D.L. ADOLPHSON, *Single machine job sequencing with precedence constraints*, SIAM J. Comput., 6 (1977), pp. 40–54.

[2] F. AFRATI, E. BAMPIS, C. CHEKURI, D. KARGER, C. KENYON, S. KHANNA, I. MILIS, M. QUEYRANNE, M. SKUTELLA, C. STEIN, AND M. SVIRIDENKO, *Approximation schemes for minimizing average weighted completion time with release dates*, in Proceedings of the 40th Symposium on the Foundations of Computer Science, 1999, pp. 32–43.

[3] S. CHAKRABARTI, C. PHILLIPS, A. SCHULZ, D.B. SHMOYS, C. STEIN, AND J. WEIN, *Improved scheduling algorithms for minsum criteria*, in Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, Springer, 1996, pp. 646–657.

[4] C. CHEKURI, R. JOHNSON, R. MOTWANI, B.K. NATARAJAN, B.R. RAU, AND M. SCHLANSKER, *Profile-driven instruction level parallel scheduling with applications to super blocks*, in Proceeding of the 29th Annual International Symposium on Microarchitechture (MICRO-29), 1996, pp. 58–67.

[5] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, *Approximation techniques for average completion time scheduling*, in Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 1997, pp. 609–618.

[6] C. Chekuri and R. Motwani, *Precedence constrained scheduling to minimize weighted completion time on a single machine*, Discrete Appl. Math., 98 (1999), pp. 29–38.

[7] F. Chudak and D. Shmoys, *Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds*, in Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 1997, pp. 581–590.

[8] F. Chudak and D. Hochbaum, *A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine*, Oper. Res. Lett., 25 (1999), pp. 199–204.

[9] J. Du, J.Y.T. Leung, and G.H. Young, *Minimizing mean flow time with release time constraint*, Theoret. Comput. Sci., 75 (1990), pp. 347–355.

[10] J. Du, J.Y.T. Leung, and G.H. Young, *Scheduling chain structured tasks to minimize makespan and mean flow time*, Inform. and Comput., 92 (1991), pp. 219–236.

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

[12] M.X. Goemans, *Improved approximation algorithms for scheduling with release dates*, in Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 1997, pp. 591–598.

[13] M. Goemans, M. Queyranne, A. Schulz, M. Skutella, and Y. Wang, *Single machine scheduling with release dates*, submitted.

[14] R.L. Graham *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[15] L.A. Hall, D.B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, in Proceedings of the Seventh ACM-SIAM Symposium on Discrete Algorithms, Atlanta, 1996, pp. 142–151.

[16] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Offline and online algorithms*, Math. Oper. Res., 22 (1997), pp. 513–544.

[17] J.L. Hennessy and T. Gross, *Postpass code optimization of pipeline constraints*, ACM Trans. Program. Lang. and Systems, 5 (1983), pp. 422–448.

[18] J.A. Hoogeveen, P. Schuurman, and G.J. Woeginger, *Non-approximability results for scheduling problems with minsum criteria*, in Integer Programming and Combinatorial Optimization, R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado, eds., Lecture Notes in Comput. Sci., Springer, Berlin, 1998, pp. 353–366.

[19] W.A. Horn, *Single-machine job sequencing with treelike precedence ordering and linear delay penalties*, SIAM J. Appl. Math., 23 (1972), pp. 189–202.

[20] D.S. Hochbaum and D.B. Shmoys, *A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach*, SIAM J. Comput., 17 (1988), pp. 539–551.

[21] J.A. Hoogeveen and A.P.A. Vestjens, *Optimal On-line algorithms for single-machine scheduling* in Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization, Vancouver, BC, Canada, 1996, pp. 404–414.

[22] E.L. Lawler, *Sequencing jobs to minimize total weighted completion time*, Ann. Discrete Math., 2 (1978), pp. 75–90.

[23] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, in Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory, North-Holland, Amsterdam, 1990.

[24] A. Munier, M. Queyranne, and A. S. Schulz, *Approximation bounds for a general class of precedence constrained parallel machine scheduling problems*, in Integer Programming and Combinatorial Optimization, R.E. Bixby, E.A. Boyd, and R.Z. Ríos-Mercado, eds., Lecture Notes in Comput. Sci., Springer, Berlin, 1998, pp. 367–382.

[25] C. Phillips, C. Stein, and J. Wein, *Scheduling jobs that arrive over time*, in Proceedings of the Fourth International Workshop on Algorithms and Data Structures, Kingston, ON, Canada, 1995, pp. 86–97.

[26] A.S. Schulz and M. Skutella, *Scheduling-LPs Bear Probabilities: Randomized Approximations for Min-Sum Criteria*, Technical Report 533-1996, Fachbereich Mathematik, Technische Universität Berlin, Berlin, Germany, 1996.

[27] A.S. Schulz, *Scheduling to minimize total weighted completion time: Performance guarantees of lp based heuristics and lower bounds*, in Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization, Vancouver, BC, Canada, 1996, pp. 301–315.

[28] A.S. Schulz and M. Skutella, *Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria*, in Proceedings of the Fifth Annual European Symposium on Algorithms, Graz, Austria, 1997, pp. 416–429.

[29] A.S. Schulz and M. Skutella, *Random-based scheduling: New approximations and lp lower bounds*, in Randomization and Approximation Techniques in Computer Science (RANDOM), J. Rolim, ed., Lecture Notes in Comput. Sci. 1269, Springer, Berlin, 1997, pp. 119–133.

[30] W.E. Smith, *Various optimizers for single-stage production*, Naval Res. Logist. Quart., 3 (1956), pp. 59–66.

[31] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*, Oper. Res. Lett., 21 (1997), pp. 115–22.

[32] L. Stougie, *private communication* cited in [21], 1995.

[33] L. Stougie and A. Vestjens, *private communication*, 1996.

[34] E. Torng and P. Uthaisombut, *Lower Bounds for SRPT-Subsequence Algorithms for Nonpreemptive Scheduling*, manuscript, 1998.

[35] S. Weiss and J.E. Smith, *A study of scalar compilation techniques for pipelined supercomputers*, in Proceedings of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 1987, pp. 105–109.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 167–192

MARKOV CHAIN ALGORITHMS FOR PLANAR LATTICE STRUCTURES*

MICHAEL LUBY†, DANA RANDALL‡, AND ALISTAIR SINCLAIR§

Abstract. Consider the following Markov chain, whose states are all domino tilings of a $2n \times 2n$ chessboard: starting from some arbitrary tiling, pick a 2×2 window uniformly at random. If the four squares appearing in this window are covered by two parallel dominoes, rotate the dominoes 90^o in place. Repeat many times. This process is used in practice to generate a random tiling and is a widely used tool in the study of the combinatorics of tilings and the behavior of dimer systems in statistical physics. Analogous Markov chains are used to randomly generate other structures on various two-dimensional lattices. This paper presents techniques which prove for the first time that, in many interesting cases, a small number of random moves suffice to obtain a uniform distribution.

Key words. Markov chains, rapid mixing, dimer systems, domino tilings, Eulerian orientations

AMS subject classifications. 82B41, 05C70, 68Q25, 60J20, 82B20

PII. S0097539799360355

1. Introduction.

This paper is concerned with algorithmic problems of the following type: given a simply connected region S of the two-dimensional Cartesian lattice (e.g., an $n \times n$ chessboard), generate uniformly at random a tiling of S with nonoverlapping dominoes, each of which covers two adjacent squares of the lattice. This problem arises in statistical physics, where the tilings correspond to configurations of a *dimer system* on S (see, e.g., [8]). Various physical properties of the system are related to the expected value, over the uniform distribution, of some function defined over configurations, such as the number of horizontal dominoes or the correlation between the orientation of dominoes at two given squares. An algorithm for randomly generating configurations allows such expectations to be estimated to any desired accuracy. It also enables one to formulate and test more detailed properties of a "typical" configuration, such as the Arctic Circle theorem [10], which began as a conjecture based on observations of random configurations.

A host of other problems of physical and combinatorial interest center around the properties of random structures of various kinds on a two-dimensional lattice. Further examples that we shall consider in this paper are lozenge tilings of a triangular lattice (corresponding to a dimer system with a different underlying geometry), and Eulerian orientations of a Cartesian lattice, also known in the statistical mechanics community as the six-point ice model. In all cases, algorithms that randomly generate

*Received by the editors August 26, 1999; accepted for publication (in revised form) January 12, 2001; published electronically July 25, 2001. A preliminary version of this paper appeared in *Proceedings of the* 36*th IEEE Symposium on Foundations of Computer Science*, Milwaukee, WI, 1995, pp. 150–159.

http://www.siam.org/journals/sicomp/31-1/36035.html

†Digital Fountain, Inc., 600 Alabama St., San Francisco, CA 94110 (luby@digitalfountain.com). This research was done while at the International Computer Science Institute. This author was supported in part by NSF grant CCR-9304722 and US-Israel Binational Science Foundation grant 92-00226.

‡School of Mathematics and College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0160 (randall@math.gatech.edu). This author was supported in part by NSF Career grant CCR-9703206.

§Computer Science Division, University of California, Berkeley, CA 94720-1776 (sinclair@cs.berkeley.edu). This author was supported in part by NSF grant CCR-9505448, by ICSI Berkeley, and by a UC Berkeley Faculty Research grant.

configurations are the major experimental tool available to researchers interested in the properties of such systems.

Returning to our first example, here is the algorithm that is most widely used in practice to generate a random domino tiling of a region S. Starting from an arbitrary tiling, pick a 2×2 window uniformly at random. If the four squares in this window are covered by two parallel dominoes, rotate the dominoes in place (see Figure 1). Repeat this operation a large number of times. The resulting tiling should then be (almost) random.

FIG. 1. *Domino rotations.*

The fact that this process (a Markov chain on the set of tilings) is connected (i.e., that every tiling is reachable from every other one by a sequence of moves of the above kind) follows from a beautiful result of Thurston [17]. However, no nontrivial upper bound is known on the number of moves needed to achieve a random tiling. In practice, this number is decided by appealing to combinatorial intuition, or experimentally by some ad hoc stopping rule. What is lacking is an analysis of the rate of convergence of the Markov chain to the uniform distribution, which would supply an a priori bound on the number of moves. Similar Markov chains, based on analogous local moves, are used to generate other two-dimensional lattice structures in the same way. Like the dominoes chain, they have so far resisted analysis.

In this paper, we develop a combinatorial framework that allows several Markov chains of this kind to be analyzed for the first time. There are two essential ingredients. The first, which we believe to be of independent combinatorial interest, is to establish a one-to-one correspondence between the configurations on a lattice region S and objects which we call *routings* on a related lattice. Informally, a routing is a collection of vertex-disjoint (or edge-disjoint) paths crossing S from left to right. (See section 2 for precise definitions and examples.) These correspondences were already known, at least implicitly, but here they play an essential role in the analysis of the associated Markov chains.

The second ingredient is to interpret natural Markov chains like the one above on domino tilings in terms of the associated routings. As we shall see, elementary moves on configurations correspond to natural local perturbations of the routings (such as displacing one vertex along a path). The crucial feature of this translation is that, when viewed in the routings world, the rate of convergence of the Markov chain turns out to be amenable to a simple and elegant analysis using a coupling argument. In fact, this analysis leads us to generalize slightly the class of random moves allowed for routings; these in turn map back to natural nonlocal moves for the configurations themselves. As a result, we obtain new, nonlocal versions of the Markov chains which are provably "rapidly mixing" (i.e., they converge quickly to the uniform distribution).

The upshot of all this is low-degree polynomial bounds on the convergence time of these Markov chains for all three of the examples mentioned above. We therefore

provide the first rigorous justification for experiments that use short simulations of the chains in order to generate random configurations.

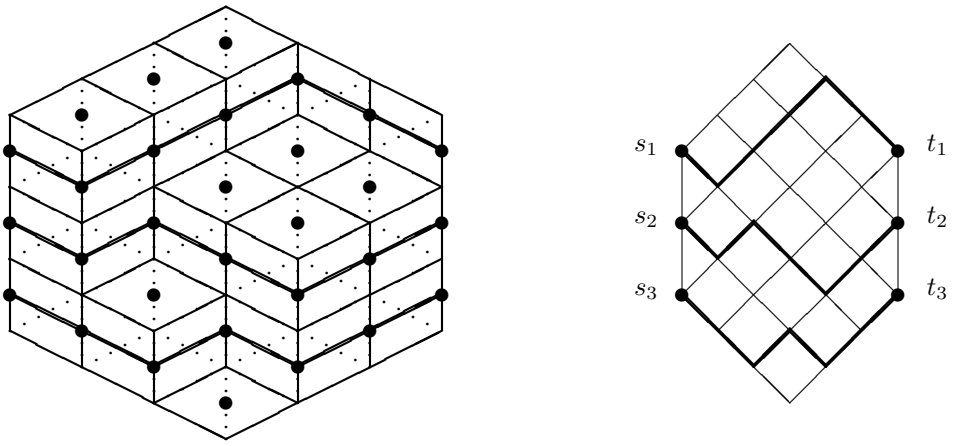
We should mention that these problems can be solved by alternative approaches. A combinatorial trick known as the Gessel–Viennot method [7] allows one to count lattice routings by evaluating a suitable determinant. In conjunction with self-reducibility properties, this allows one to generate configurations uniformly at random (see, e.g., [16]). Other Markov chain algorithms which can be applied to these structures in arbitrary graphs are given in [9, 12] (for tilings) and [13] (for Eulerian orientations). However, in the important special case of planar lattices, the algorithms in this paper have better time bounds than these other methods and are simpler, more natural, and quite widely used in practice. Moreover, our bounds are in fact quite pessimistic (our main concern is to introduce the methodology rather than to tune the bounds) and can be improved with a more detailed analysis (see the subsequent paper by Wilson [18]). We also point out that there is a simple experiment one can perform which provides a reliable estimate of the true convergence rate: this can be used to dramatically reduce the number of simulation steps required in practice, as discussed in [14].

The remainder of the paper is organized as follows. In section 2 we illustrate the correspondence between lattice configurations, routings, and height functions for each of our examples: lozenge tilings, domino tilings, and Eulerian orientations. In section 3, we show how to analyze the rate of convergence of the natural Markov chain on lozenge tilings by applying a coupling argument to the corresponding chain on routings. In the process, we will enrich the chain with nonlocal moves. In sections 4 and 5 we show how to apply the same technology to analogous Markov chains for domino tilings and Eulerian orientations.

2. Lattice routings and height functions. In this section, we consider several important examples of lattice structures and illustrate the correspondence between them and collections of paths which we call *routings*. Routings are the key for analyzing the convergence rate, or running time, of our Markov chain algorithms. As we shall see, the routings are closely related to a third representation of the lattice structures known as *height functions*, which arise from the tiling groups of Conway and Lagarias [4] and Thurston [17]. For each of our three examples we briefly outline the bijections between lattice structures, routings, and height functions. As we shall see later, this framework will make the analytical tools we develop rather generally applicable.

2.1. Lozenge tilings. The first structures we consider are lozenge tilings of a finite region of the triangular lattice: we discuss these first because the correspondence with routings is most direct here. A *lozenge* is the analogue of a domino in the Cartesian lattice: each lozenge covers two adjacent triangles in the lattice and has three possible orientations. Lozenge tilings are configurations of a dimer system on this lattice. As explained in the introduction, we are interested in the problem of generating a random lozenge tiling of the given region.

The routings corresponding to lozenge tilings are defined on an associated Cartesian lattice. Given a finite, simply connected region S of the triangular lattice, we define an associated region $\hat{S}$ of the Cartesian lattice as follows. The vertices of $\hat{S}$ correspond to the midpoints of the vertical edges in S, and two vertices in $\hat{S}$ are connected if the corresponding points in S lie on adjacent triangles. This mapping is demonstrated in Figure 2.



A lozenge tiling on S The corresponding routing on $\hat{S}$

FIG. 2. *Lozenge tilings and routings.*

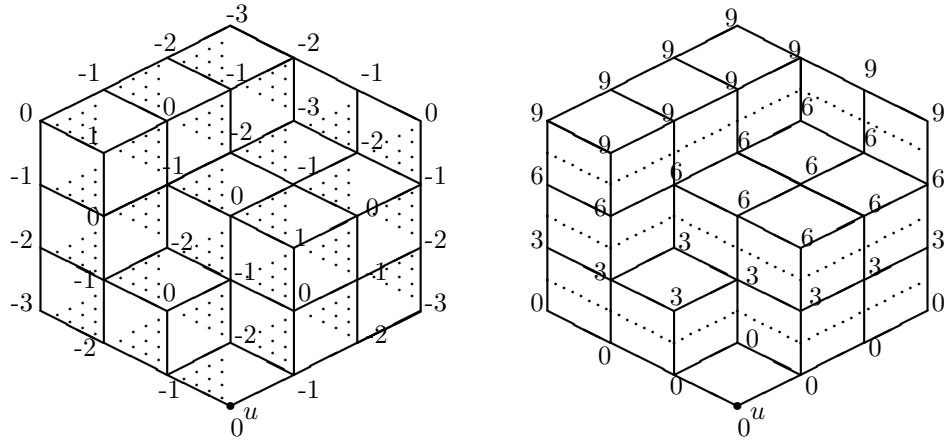
The vertices of $\hat{S}$ that correspond to edges on the boundary of S are called *sources* and *sinks*: a vertex v is a source if the interior of $\hat{S}$ lies to the right of v and a sink if the interior of $\hat{S}$ lies to its left. It is not hard to check that, if a lozenge tiling of S exists, then the numbers of sources and sinks are necessarily equal. Label the sources $s_1, \ldots, s_k$ and the sinks $t_1, \ldots, t_k$.

A *lozenge routing* of $\hat{S}$ is a set of k nonintersecting (i.e., vertex-disjoint) shortest paths on the Cartesian lattice within $\hat{S}$ from $\{s_1, \ldots, s_k\}$ to $\{t_1, \ldots, t_k\}$. It is not difficult to see that there is a bijection between lozenge tilings and lozenge routings of corresponding regions. Figure 2 provides a pictorial illustration of this correspondence for a typical region S. An easy way to see this is to "mark" the tiles containing two vertical edges with a stripe connecting the centers of these edges. Now notice that if a tile is placed next to any vertical edge of S, then it must have two vertical edges whose midpoints correspond to adjacent vertices in $\hat{S}$. Furthermore, if the vertical edge of a tile lies in the interior of S, then it must be adjacent to another tile having two vertical edges (so the markings line up). Following such a sequence of tiles, starting from each vertical edge on the boundary of S, defines a set of nonintersecting source-sink paths in $\hat{S}$, i.e., a routing. Conversely, given a routing we may invert the above construction to create a partial tiling using only marked (i.e., nonhorizontal) tiles. All vertical edges of these tiles are adjacent to another tile or to the boundary of S. The untiled portion of S therefore consists of regions bounded only by nonvertical edges. It is not hard to see that these can be tiled in only one way, using only horizontal tiles. Hence there is a bijection between the set of tilings and routings of S. Moreover, we can order the sinks so that the path from s_i always ends at t_i, for all i, and the path from s_i to t_i will have the same length in every routing.

This correspondence has been formalized by Sachs et al.

THEOREM 2.1 (see [1, 11]). *The set of lozenge tilings of S corresponds bijectively with the set of lozenge routings of $\hat{S}$.* □

The bijection between tilings and routings is closely related to the *height functions* defined by Thurston [17]. Although we don't require it for our analysis, we briefly describe this connection here because it sheds further light on the above correspondence.



The height function h_{loz} The adjusted height function $\hat{h}_{loz}$

FIG. 3. *Lozenge tilings and height functions.*

In this representation, the *height* of each vertex in the region S is determined by an underlying algebraic structure known as the tiling group, introduced by Conway and Lagarias [4]. The height function can be extended to all points within the region by a piecewise linear function, thereby defining a three-dimensional surface associated with each tiling. It turns out that, by viewing each surface from a certain orientation, the paths of a routing can be interpreted as the level sets of this surface. This is a common feature of all the structures we discuss in this paper; in the case of lozenges the surfaces are immediately apparent and are just the set of three-dimensional boxes that seem to "jump out" of the two-dimensional lozenge tiling, as in the left-hand picture in Figure 2.

The heights of the vertices of any tiling can be determined by the following simple rule. First choose a vertex u on the boundary of the region and fix its height $h_{loz}(u) = 0$. To determine the heights of all other vertices we rely on the bipartite structure of the dual lattice, which allows us to color all the triangles pointing to the right white and all those pointing to the left black. Starting at u, walk around the boundaries of tiles until an edge is traversed from a vertex v with known height $h_{loz}(v)$ to a vertex w whose height has not yet been determined. If the triangle to the left of this edge (w.r.t. the direction of traversal) is black, set $h_{loz}(w) = h_{loz}(v) + 1$; if the triangle to the left is white, set $h_{loz}(w) = h_{loz}(v) - 1$. Repeat this process until all of the vertices have been visited. For any tiling of a simply connected region, the heights are always unique and well-defined (up to translation). As an example, Figure 3 shows the height function for the tiling of Figure 2 with u being the bottom vertex. A simple consequence of the above definition is that the heights along the boundary of any tiling are determined by the region alone and are identical for all tilings of the region.

The connection between height functions and routings in the case of lozenge tilings is quite straightforward. We create an *adjusted* height function $\hat{h}_{loz}$ by letting v_y be the vertical coordinate of vertex v (i.e., the row of the lattice that it lies in) and defining $\hat{h}_{loz}(v) = h_{loz}(v) + v_y$. It is a simple exercise to verify that, for any vertex v, the adjusted height function satisfies $\hat{h}_{loz}(v) = \hat{h}_{loz}(u) + 3k$, where k is the number of paths in the routing which lie between v and u (with k being negative if these

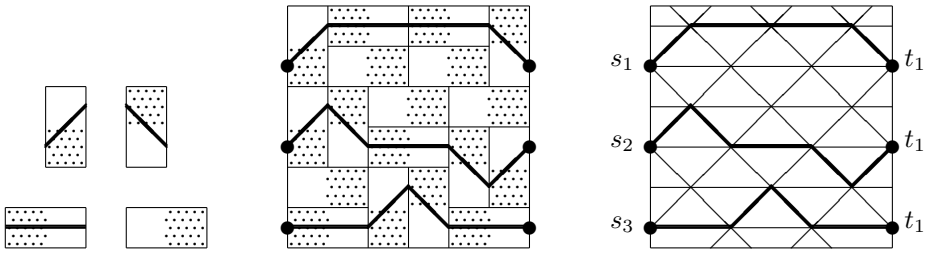


FIG. 4. *Domino tilings and routings.*

paths lie above v and positive if they lie below v); see Figure 3. This can be proved using the observation that on every lozenge the heights of the two vertices where the angle is acute must be equal. This implies that horizontal lozenges, where the two vertices with acute angles lie in the same row, will have the same adjusted height at all four corners, while on the other two types of lozenges the lower two vertices will have strictly smaller height than the upper two vertices.

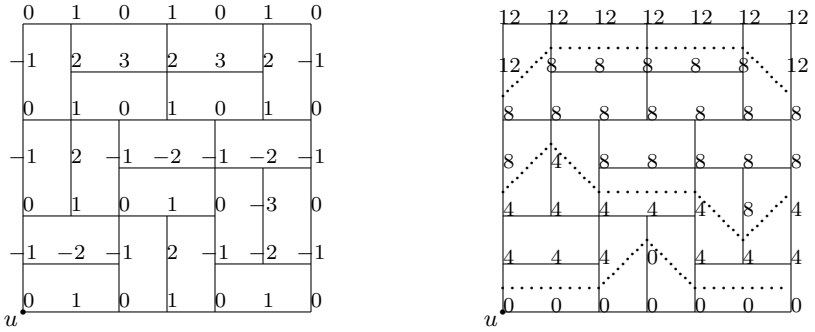
2.2. Domino tilings. A *domino tiling* is a covering of a finite region of the Cartesian lattice with dominoes, where each domino covers two adjacent squares of the region. Domino tilings are configurations of dimer systems on this lattice. As in the case of lozenge tilings, there is a family of routings which correspond bijectively to the set of domino tilings. Again, the routings are defined on an associated lattice, which in this case is triangular.

Given a finite, simply connected tileable region S in the Cartesian lattice, we define a related region $\hat{S}$ (which lies on a triangular lattice). First color the squares of the Cartesian lattice black and white as on a chessboard. The vertices of the triangular lattice can be defined as the centers of all of the vertical edges of the Cartesian lattice which have a black square to their right, where edges are defined by connecting each vertex (x, y) to $(x + 1, y + 1)$, $(x + 1, y - 1)$, and $(x + 2, y)$. The region $\hat{S}$ is the part of this triangular lattice which is defined by vertices and edges contained completely within the original region S. Sources and sinks of $\hat{S}$ are defined in similar fashion to the lozenge case: sources are boundary vertices with the interior of $\hat{S}$ to their right, and sinks are those with the interior to their left. Once again, we pair up sources $\{s_1, \ldots, s_k\}$ and sinks $\{t_1, \ldots, t_k\}$ in the obvious way. A *domino routing* of $\hat{S}$ is then a collection of nonintersecting shortest paths on the triangular lattice within $\hat{S}$ from s_i to t_i for each i.

The correspondence between domino tilings and routings is illustrated by means of an example in Figure 4: each tiling defines a unique routing using the three permitted paths through the dominoes as shown. This correspondence is formalized in the next theorem.

THEOREM 2.2. *There is a bijection between domino tilings of S and domino routings of $\hat{S}$.*

Proof. Figure 4 indicates how to use paths through dominoes to map tilings to routings as follows. Start at a source vertex s_i. By definition, there must be a black square in the interior (of the original Cartesian lattice region S) to the right of s_i. The domino occupying this square determines the first step of our path: we connect s_i to the unique point on the right boundary of the domino which is a vertex of $\hat{S}$ in the underlying triangular lattice. We now find ourselves at a new point with a



The height function h_{dom} The adjusted height function $\hat{h}_{dom}$

FIG. 5. *Domino tilings and height functions.*

black square to our right, and we can repeat this process. Since we migrate to the right in each step, we eventually hit a point on the right boundary of $\hat{S}$ which has a black square to its right and thus must be a sink. The paths must be nonintersecting because the tiles cannot overlap.

To see that the above map is bijective, we construct the inverse map from routings to tilings as follows. Each path starts at a source s_i (which has a black square to its right) and follows lattice edges to a sink t_i. As we follow the path from left to right, we tile each of the black squares to the right of the lattice points on the path (except the sink t_i). There are three possible positionings for each tile corresponding to the three possible types of edges the path can pass through. Since the paths are nonintersecting, our tiles cannot overlap and we are left with a partial tiling of S. Now there is a unique way to tile the remaining parts of the region, namely, using only horizontal tiles (whose left half covers a white square). To see this, consider any untiled black square. The white square to its left must be untiled, for if it were tiled there would be a path exiting its right boundary, and then the black square would be tiled. Therefore every black square can be tiled with the right half of a horizontal domino. This completes the tiling since there must be an equal number of black and white squares in S. The uniqueness comes from the fact that the leftmost square in each row of each untiled subregion is white, so we cannot complete the tiling if we use any vertical tiles. □

Once again, the routings defined above can be viewed as level sets of a height function. The height function which arises from the tiling group for dominoes can be summarized using a rule based on the bipartition underlying the dual lattice (i.e., the black and white squares of the chessboard). To define the height function h_{dom}, start with some point u on the boundary and assign $h_{dom}(u) = 0$. Now, walking along edges bounding the tiles starting at u, if the square to the left of an edge is black (respectively, white), increase (respectively, decrease) the height by one. An example is illustrated in Figure 5 with u being the bottom left vertex.

The connection between height functions and routings is analogous to that for lozenge tilings. For any point $v = (v_x, v_y)$, let v_y be the y coordinate of v (i.e., the row of the lattice). Let $\mathrm{par}(v) = 1$ if v is the lower left corner of a white square, and let $\mathrm{par}(v) = 0$ if v is the lower left corner of a black square. Now define the adjusted height function by $\hat{h}_{dom}(v) = -h_{dom}(v) + 2v_y + \mathrm{par}(v)$. It is easy to verify that

$\hat{h}_{dom}(v)$ is equal to $\hat{h}_{dom}(u) + 4k$, where k is the number of paths which lie between u and v in the routing (k being negative if these paths lie below u). Figure 5 illustrates this relationship.

2.3. Eulerian orientations. A third important set of structures which can be identified with lattice routings are the Eulerian orientations of a region of the Cartesian lattice with specified boundary conditions. An *Eulerian orientation* of an undirected graph is an orientation of its edges so that the in-degree of every vertex is equal to its out-degree. In this problem, the input is a finite simply connected region S of the two-dimensional Cartesian lattice, together with a fixed orientation for each of the edges that connects the boundary of S with the interior: these orientations are the *boundary conditions*. Our task is to generate uniformly at random an orientation of the edges in the interior such that all interior vertices have equal in-degree and out-degree. This is the "six-point model" in statistical mechanics, also known as the "ice model."

The correspondence between Eulerian orientations and a suitable class of routings is well known in the physics community (see, e.g., [3]) and is sketched in Figure 6. The sources and sinks in this case are defined by the boundary conditions as shown. Sources are the vertices on the boundary which are connected to the interior of the region by edges directed towards the interior and which point up or to the right; sinks are the boundary vertices which are connected to the interior by edges directed away from the interior (i.e., towards the boundary) and which also point up or to the right. A necessary condition for the existence of an Eulerian orientation is, of course, that the number of sources and sinks are equal.

An *Eulerian routing* of S is a set of shortest paths in S from sources $\{s_1, \ldots, s_k\}$ to sinks $\{t_1, \ldots, t_k\}$ on the boundary. The paths are permitted to intersect at a vertex but *not* along an edge. As illustrated in Figure 6, to get the Eulerian routing corresponding to a given Eulerian orientation, we construct the paths only from edges that are oriented up and those that are oriented to the right. It is straightforward to establish that there is a bijection between the set of Eulerian orientations of a region S and the set of Eulerian routings of S.

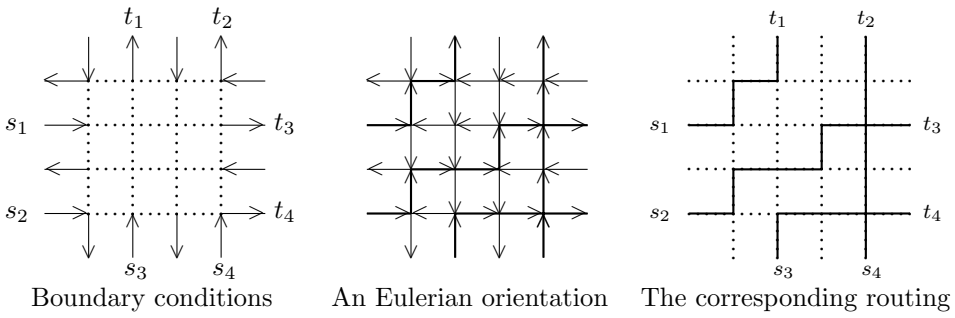
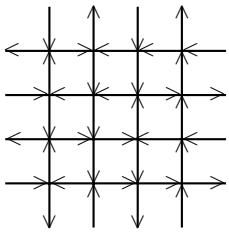


FIG. 6. *Eulerian orientations and routings.*

The height function associated with Eulerian orientations is an assignment of integers to the center of each face within a region such that neighboring faces differ in height by one. In the statistical physics community, this is known as a "solid-on-solid surface." To define the heights, start with a face adjacent to the boundary and assign to its center u the height $h_{eul}(u) = 0$. To define the heights of the other faces, walk along edges in the dual lattice. When traversing a dual edge (v, w), where the

height of v is already defined, let $h_{eul}(w) = h_{eul}(v) + 1$ if the edge of the Eulerian orientation which was crossed points to the right (relative to the path from v to w) and let $h_{eul}(w) = h_{eul}(v) - 1$ if the edge points to the left. Again this height function is unique (up to translation) and well-defined. If we define an adjusted height function $\hat{h}_{eul}(v) = h_{eul}(v) - v_x + v_y$, where v_x and v_y are the Cartesian coordinates of v, then $\hat{h}_{eul}(v) = \hat{h}_{eul}(u) + 2k$, where k is the number of paths of the routing which lie between u and v. See Figure 7 for an illustrative example.



Eulerian orientation

$\leftrightarrow$

0	1	0	1	0
1	0	1	2	1
0	1	2	1	0
1	0	1	2	1
0	1	0	1	0

Solid-on-solid surface

$\leftrightarrow$

8	8	6	6	4
8	6	6	6	4
6	6	6	4	2
6	4	4	4	2
4	4	2	2	0

The adjusted height function $\hat{h}_{eul}$

FIG. 7. *The height function representation of an Eulerian orientation.*

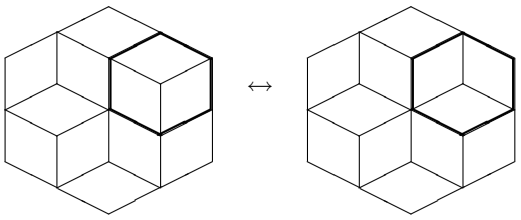
Remark 1. The solid-on-solid height function demonstrates the well-known connection between Eulerian orientations and three-colorings of a lattice region, since taking the values of the heights mod 3 always gives a valid three-coloring of the region (see, e.g., [3]). If we fix the colors of all the dual vertices on the boundary of the region (and hence the boundary conditions), then there is a bijection between three-colorings and Eulerian orientations.

3. Generating lozenge tilings. This section is devoted to an analysis of a natural Markov chain algorithm for generating random lozenge tilings. The analysis will exploit in a crucial way the correspondence with routings established in section 2.1. We present this example first because it is the most straightforward to deal with. Analogous Markov chains for generating the other structures discussed in section 2 can be analyzed by more refined applications of the same techniques, as we will demonstrate in section 4.

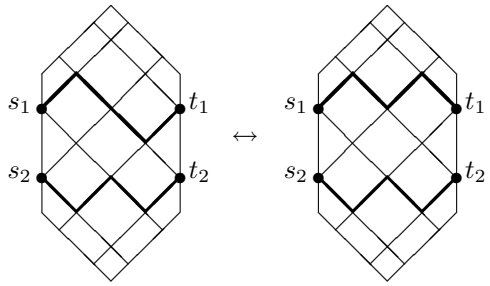
3.1. The Markov chain. In the introduction, we discussed a Markov chain on domino tilings based on a local move that rotates a pair of adjacent dominoes. The analogous Markov chain for lozenge tilings has as its local move a rotation of three neighboring lozenges (see Figure 8(a)). As in the domino case, this chain can also be shown to be connected and to converge to the uniform distribution over tilings (see section 3.2).

Let us now interpret this Markov chain in the world of routings. It is clear that a lozenge rotation induces a natural local move on the corresponding routing, in which a "peak" or a "valley" is inverted by switching two edges (see Figure 8(b)). Therefore we can think of the chain as picking a point uniformly at random on the routing and inverting it if possible.[1] It turns out that the Markov chain in the routings world

[1]Strictly speaking, this chain differs slightly from the original one in that it does not attempt to make rotations at points that are not on the current paths, which will always be rejected. This merely reduces the self-loop probabilities of the chain, and hence speeds up the mixing time by a factor that is bounded by the area of the region.



(a) Tiling rotation



(b) Routing rotation

FIG. 8. *Lozenge rotations.*

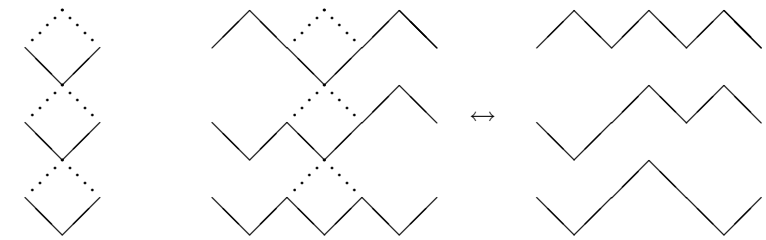
becomes considerably easier to analyze if *every* peak and valley can give rise to a rotation: note that this is not the case for the above chain, since sometimes when we try to invert a point the move will be blocked by the presence of another path. (Recall that the paths in a routing are not allowed to intersect. See, e.g., the second valley on the lower path in the left-hand routing of Figure 8(b).) This motivates the introduction of a more general set of moves in which a *tower* is rotated. The original moves will simply correspond to the special case of rotating a tower of height 1.

In the routings lattice, define the *cell at* (x, y) to be the edges connecting $(x, y), (x+1, y+1), (x, y+2)$, and $(x-1, y+1)$. A *tower of height* h is a connected set of cells at the points $(x, y), (x, y+2), \ldots, (x, y+2h-2)$, where either the points $(x, y), (x, y+2), \ldots, (x, y+2h-2)$ are all valleys and the point $(x, y+2h)$ does not lie on the routing, or the points $(x, y+2h), (x, y+2h-2), \ldots, (x, y+2)$ are all peaks and the point (x, y) does not lie on the routing. We call the points (x, y) and $(x, y+2h)$ the *bottom* and *top* of the tower, respectively (see Figure 9(a)). Provided both the top and bottom of a tower lie in the region, its peaks (respectively, valleys) can be inverted by switching pairs of edges in each of its cells. Such an operation is called a *rotation* of the tower. Figure 9(b) illustrates a rotation of a tower of height 3, and Figure 9(c) shows that tower rotations have a natural counterpart in the original tilings world.

Let R_1 and R_2 be lozenge routings of the region $\hat{S}$. We define a Markov chain $\mathcal{M}_{loz}$ in which there is a move from R_1 to R_2 iff R_1 and R_2 differ by a single tower rotation. The transition probabilities $P(\cdot, \cdot)$ of $\mathcal{M}_{loz}$ are defined by

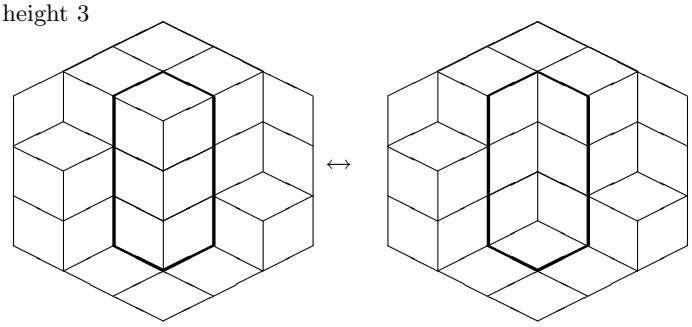
$$P(R_1, R_2) = \begin{cases} 1/2Nh & \text{if } R_1, R_2 \text{ differ by rotation of a tower of height } h; \\ 1 - \sum_{R \neq R_1} P(R_1, R) & \text{if } R_2 = R_1, \end{cases}$$

where N is the total number of internal vertices along all the paths in any routing.



(a) Tower of height 3

(b) Tower rotation in a routing



(c) Tower rotation in a tiling

FIG. 9. *A move in the Markov chain $\mathcal{M}_{loz}$ for lozenge routings (and tilings).*

Notice that we may implement a move of $\mathcal{M}_{loz}$ as follows. Given a routing R, choose an internal point p on one of the paths in R and a number $r \in [0,1]$ uniformly at random. Assume first that $r \leq 1/2$. If p is a valley, then it is the bottom of a unique tower (say, of height h); in this case, if $r \leq 1/2h$, then rotate the tower if possible (i.e., if the top of the tower lies in the region). On the other hand, if $r > 1/2$ check whether p is a peak (and hence the top of a unique tower), and if possible rotate this tower if $r \geq 1 - 1/2h$, where h is the height of the tower. In all other cases do nothing. This slightly unusual implementation is a technical device that will prove useful later when we define a coupling for the Markov chain.

As we shall see in section 3.2, this Markov chain is ergodic and converges to the uniform distribution over all lozenge routings. Therefore, we can generate a random tiling by simulating $\mathcal{M}_{loz}$ for sufficiently many steps, starting from an arbitrary routing and outputting the tiling corresponding to the final routing. The efficiency of this algorithm depends on the number of simulation steps necessary to ensure an (almost) uniform distribution, or, equivalently, on the *rate of convergence* of the Markov chain. We shall see in section 3.4 that a small number of steps suffice, or in other words that the Markov chain is "rapidly mixing." In preparation for this we will introduce some general technology in section 3.3.

3.2. Ergodicity of $\mathcal{M}_{loz}$. The fact that $\mathcal{M}_{loz}$ converges to the uniform distribution over tilings follows almost immediately from the fact that the chain is connected, i.e., every state is reachable from every other. It is actually quite straightforward to show that the Markov chain based on *simple* rotations connects the state space of all lozenge tilings. This is sufficient to show the connectedness of the Markov chain based on towers since it includes all the simple moves. Here we sketch a proof whose machinery will be useful to us in other ways.

LEMMA 3.1. *The state space of the Markov chain $\mathcal{M}_{loz}$ is connected.*

Proof. It is conceptually easier to work in the routings world. Note that there is a natural partial order on the set of all lozenge routings of a given region $\hat{S}$, defined as follows. Let R_1, R_2 be two routings, and let P_1, P_2 be a pair of corresponding paths (i.e., having the same source and sink) in R_1, R_2, respectively. We say that $P_1 \succeq P_2$ iff the ith point of P_1 lies on or above the ith point of P_2 for all i. We say that $R_1 \succeq R_2$ iff the relation $P_1 \succeq P_2$ holds for all pairs of corresponding paths P_1, P_2. Since all routings of a region $\hat{S}$ have the same number of paths, this relation is well-defined. If $R_1 \succeq R_2$, we define the distance between them to be the total area enclosed between all pairs of corresponding paths. Now it is not too hard to see that, for any finite, simply connected region $\hat{S}$, there is a unique minimum routing $R_\perp$, such that $R \succeq R_\perp$ for all routings R of $\hat{S}$.[2] We will show that every routing is connected to $R_\perp$ by a sequence of simple rotations (i.e., of towers of height 1) each of which decreases the distance to $R_\perp$ by one.

Let $R \neq R_\perp$ be an arbitrary routing. Starting with the lowest pair of corresponding paths in $R, R_\perp$ and working upwards, scan left to right along the paths until the first point x at which $R, R_\perp$ deviate from one another. Now continue to follow these two paths (say, P and $P_\perp$) until the first point y where they meet again (this must happen, at the latest, at their common sink). Notice that the union of the segments of P and $P_\perp$ that lie between x and y form a circuit whose interior is entirely contained within the interior of $\hat{S}$, since $\hat{S}$ is simply connected. In addition, since $R_\perp$ is minimal, P must lie strictly above $P_\perp$ along the segment delimited by these two points. Therefore, P must have at least one peak along this segment; let v be the leftmost such peak. It follows that v must be the top of a (rotatable) tower of height 1 in R, since v lies strictly above the corresponding point in $R_\perp$ and R coincides with $R_\perp$ on all lower paths. Therefore, if we rotate the tower at v we decrease the distance from $R_\perp$ by one unit. Applying this argument repeatedly, we arrive at $R_\perp$. □

THEOREM 3.2. *The Markov chain $\mathcal{M}_{loz}$ is ergodic and converges to the uniform distribution over lozenge tilings.*

Proof. The Markov chain is clearly aperiodic since it has a holding probability of at least $1/2$ in every state. Together with Lemma 3.1, this implies that the chain is ergodic, i.e., converges to a unique stationary distribution. That this stationary distribution is uniform follows from the fact that the transition probabilities are symmetric: for any pair of adjacent routings R_1, R_2, we have $P(R_1, R_2) = P(R_2, R_1) = 1/2Nh$, where h is the height of the tower by which R_1 and R_2 differ. □

3.3. Coupling and the convergence rate. In this subsection, we establish some general machinery for bounding the rate of convergence of Markov chains, which we shall use repeatedly in the remainder of the paper. Consider an ergodic Markov chain $\mathcal{M}$ with finite state space Ω, transition matrix P, and stationary distribution π. Following standard practice, for any given initial state x, we shall measure the deviation of the distribution $P^t(x, \cdot)$ at time t from π by the *variation distance*

$$\Delta_x(t) = \tfrac{1}{2} \sum_{y \in \Omega} |P^t(x, y) - \pi(y)|.$$

The *mixing time* of the Markov chain is defined by the function

$$\tau(\epsilon) = \max_x \min\{t : \Delta_x(t') \leq \epsilon \text{ for all } t' \geq t\}.$$

[2] A careful proof of the existence of a unique highest tiling (in the height function representation) was given by Thurston [17]; this corresponds precisely to the minimum routing defined here.

Our strategy for bounding $\tau(\epsilon)$ is to construct a *coupling* for the Markov chain, i.e., a stochastic process $(X_t, Y_t)_{t=0}^{\infty}$ on $\Omega \times \Omega$ with the following properties:

1. Each of the processes X_t and Y_t is a faithful copy of $\mathcal{M}$ (given initial states $X_0 = x$ and $Y_0 = y$).
2. If $X_t = Y_t$, then $X_{t+1} = Y_{t+1}$.

The idea here is the following. Although each of X_t, Y_t, viewed in isolation, behaves exactly like $\mathcal{M}$, they need not be independent; on the contrary, we will construct a joint distribution for the two processes in such a way that they tend to move closer together. By the second condition above, once they have met they must remain together at all future times.

The expected time taken for the processes to meet provides a good bound on the mixing time of $\mathcal{M}$. To state this formally, for initial states x, y set

$$T^{x,y} = \min\{t : X_t = Y_t \mid X_0 = x,\, Y_0 = y\}$$

and define the *coupling time* to be $T = \max_{x,y} \mathrm{E}T^{x,y}$. The following result relating the mixing time to the coupling time is standard (see, e.g., [2]).

THEOREM 3.3. $\tau(\epsilon) \le T\mathrm{e}\lceil \ln \epsilon^{-1} \rceil$. □

Next, we introduce some machinery that will help us to bound the coupling time. Suppose we have a distance function ϕ defined on $\Omega \times \Omega$ such that ϕ takes integer values in the range $[0, B]$, and $\phi(x, y) = 0$ iff $x = y$. In our examples, where the states of the Markov chain are lattice routings, ϕ will be a natural measure of the "area" between a pair of routings. We will measure the distance between a pair of processes (X_t, Y_t) using the stochastic process $\Phi(t) = \phi(X_t, Y_t)$. Our strategy will be to show that, under a suitably defined coupling, the expected change $\Delta\Phi$ in Φ is always nonpositive; intuitively, this should enable us to conclude that the coupling time is small. The following lemma makes this intuition precise.

LEMMA 3.4. *With the above notation, suppose the coupling satisfies* $\mathrm{E}\big[\Delta\Phi(t)|X_t, Y_t\big] \le 0$ *and, whenever* $\Phi(t) > 0$, $\mathrm{E}\big[(\Delta\Phi(t))^2|X_t, Y_t\big] \ge V$. *Then the expected coupling time from initial states* x, y *satisfies* $\mathrm{E}T^{x,y} \le \Phi(0)\big(2B - \Phi(0)\big)/V$.

Proof. Define the stochastic process $Z(t) = \Phi(t)^2 - 2B\Phi(t) - VT(t)$, where $T(t) = t$ if $\Phi(t) > 0$, and $T(t) = \min\{t' : \Phi(t') = 0\}$ if $\Phi(t) = 0$. Then $Z(t+1) - Z(t) = 0$ whenever $\Phi(t) = 0$, and when $\Phi(t) > 0$ we have

$$\begin{aligned}
&\mathrm{E}\big[Z(t+1)|X_t, Y_t\big] - Z(t) \\
&\quad = 2\big(\Phi(t) - B\big)\mathrm{E}\big[\Delta\Phi(t)|X_t, Y_t\big] + \big(\mathrm{E}[(\Delta\Phi(t))^2|X_t, Y_t] - V\big) \\
&\quad \ge 0.
\end{aligned}$$

Hence $Z(t)$ is a submartingale with respect to the sequence $\{(X_t, Y_t)\}_{t\ge 0}$. Moreover, the random time $T^{x,y} = \min\{t : \Phi(t) = 0\}$ (where $X_0 = x$, $Y_0 = y$) is a stopping time for $Z(t)$ with finite expectation, and the differences $|Z(t+1) - Z(t)|$ are bounded. This allows us to apply the optional stopping theorem for submartingales (see, e.g., [6, Chapter 4, Theorem 7.5] and conclude that $\mathrm{E}Z(T^{x,y}) \ge \mathrm{E}Z(0)$. From the definition of $Z(t)$, this implies that

$$-V\,\mathrm{E}T^{x,y} \ge \Phi(0)^2 - 2B\Phi(0),$$

which gives the desired upper bound on $\mathrm{E}T^{x,y}$. □

3.4. $\mathcal{M}_{loz}$ is rapidly mixing. We first consider the simplified case of routings consisting of a single path P with source s and sink t. Notice that in this case all towers have height 1.

We define a coupling as follows. Consider two copies of the Markov chain, whose states are the paths P_1 and P_2, each containing N internal points. Then in one move we choose $(i, r) \in \{1, \ldots, N\} \times [0, 1]$ uniformly at random and simultaneously move the ith points of each of P_1 and P_2 as specified by the random number r. It should be clear that this is a valid coupling. Notice that, because we are using the same value of r for both processes, the two paths will never move in opposite directions. This was the reason we implemented the Markov chain in this fashion in section 3.1.

We now proceed to bound the expected time it takes the coupled process to cause any two initial routings to agree. To simplify the analysis, we use an observation due to Propp and Wilson [14]: if the state space of a Markov chain is endowed with a partial order with unique maximum and minimum elements, and if the coupling preserves the partial order (in a sense made precise below), then the coupling time is bounded above by the expected coupling time starting from the maximum and minimum states. Recall the partial order defined in the proof of Lemma 3.1: $P_1 \succeq P_2$ iff the ith point of P_1 lies on or above the ith point of P_2 for all i. Recall also that this partial order has a unique minimum (and similarly also a unique maximum) element. To make precise the notion of preserving the partial order, we first extend the above coupling to a random function f on the entire state space; namely, pick (i, r) as above and, for any path P, let $f(P)$ be the path obtained by moving the ith point of P as specified by r. We say that the coupling is *monotone* with respect to the partial order if $P_1 \succeq P_2$ implies $f(P_1) \succeq f(P_2)$. The next lemma verifies that this condition holds for our coupling.

LEMMA 3.5. *The above coupling is monotone with respect to the partial order* $\succeq$.

Proof. Suppose $P_1 \succeq P_2$, and consider a random move defined by a pair (i, r); let $P_1' = f(P_1)$ and $P_2' = f(P_2)$ be the images of P_1, P_2, respectively. Assume that $r \geq 1/2$ (the case $r < 1/2$ is symmetric). It is straightforward to verify that if i is a peak in P_1, then either the ith point of P_1 is sufficiently far above the ith point of P_2 to ensure that $P_1' \succeq P_2$ (and therefore $P_1' \succeq P_2'$), or i is also a peak in P_2, in which case the peak is rotated in both paths. In either case we can deduce that $P_1' \succeq P_2'$. □

We now need to bound the time taken for the two extremal paths to meet. To do this, we introduce a distance function as in Lemma 3.4. For a pair of paths P_1, P_2, define the distance $\phi(P_1, P_2)$ to be the area (i.e., number of Cartesian lattice squares) of the region between P_1 and P_2. The crucial observation is that the distance $\Phi(t) = \phi(X_t, Y_t)$ will tend not to increase under our coupling, as the next lemma shows. By Lemma 3.5, we may restrict attention to the case $X_t \succeq Y_t$.

LEMMA 3.6. *Let P_1 and P_2 be any two paths such that $P_1 \succeq P_2$. Then* $\mathrm{E}[\Delta\Phi | P_1, P_2] \leq 0$.

Proof. Consider an arbitrary pair of paths $P_1 \succeq P_2$. The typical situation is as depicted in Figure 10 with path P_1 drawn as a solid line and path P_2 as a dotted line. We can partition the paths into segments $C_1, \ldots, C_\ell$ on which the two paths coincide, and segments $D_1, \ldots, D_m$ whose endpoints coincide, but for which P_1 is strictly above P_2 at all intermediate points.

First consider a segment C_i. It is clear that, if the point chosen by the coupling lies in this segment, then the point will move with the same probability on both paths, so the paths will still coincide and there will be no change in area.

Now consider a segment D_i, in which points on P_1 are strictly above the corresponding points on P_2 (except at the endpoints). On the upper path P_1, label the peaks "good" (G) and the valleys "bad" (B) and vice versa for the lower path P_2. Thus a point is good if a rotation at that point would cause the area between the paths to decrease and bad if a rotation would case the area to increase. (The boundary of the region might in fact prohibit some of these bad rotations.) It is easy to see that, on each path, the labeled points in the interior of the segment D_i are alternately good and bad with a net excess of one good point. Moreover, each endpoint of D_i contributes at most one bad point (unless D_i and D_{i+1} meet at a point, in which case there might be two bad points, but we can assign one to each segment).

Summing over all segments of the path, we see that the total number of good points is greater than or equal to the total number of bad points. Since each good or bad point is equally likely to be chosen in the coupling, the expected change in area is at most zero. □

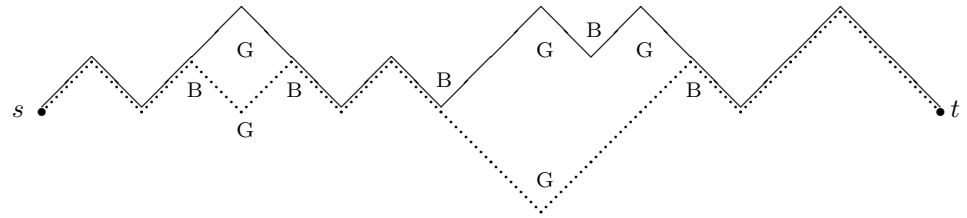


FIG. 10. *Proof of Lemma* 3.6.

THEOREM 3.7. *Let S be a region in the triangular lattice such that the corresponding region $\hat{S}$ has one source s and one sink t. Then the mixing time of the Markov chain $\mathcal{M}_{loz}$ on lozenge routings of $\hat{S}$ satisfies $\tau(\epsilon) \leq 2en^3\lceil \ln \epsilon^{-1} \rceil$, where n is the area of $\hat{S}$.*

Proof. By Theorem 3.3, it suffices to show that the coupling time T satisfies $T \leq 2n^3$. To bound T we appeal to Lemma 3.4, restricting our attention to extremal initial states following Lemma 3.5. Our distance function ϕ clearly takes integer values in the interval $[0, n]$. Moreover, we have seen in Lemma 3.6 that $\mathrm{E}[\Delta\Phi] \leq 0$. (Note that the monotonicity of the coupling ensures that any pair (P_1, P_2) that is reachable under the coupling satisfies $P_1 \succeq P_2$.) It remains only to bound $\mathrm{E}[(\Delta\Phi)^2]$, assuming that the area between the pair of paths is nonzero. In this case, there must be a segment on which the solid path is strictly above the dotted path (using the terminology in the proof of Lemma 3.6). Scanning this segment from left to right, there is a first point where at least one of the paths is good and neither is bad. If this point is chosen at the next time step, then there will be a decrease in area with probability at least $\frac{1}{2}$. Hence Φ decreases strictly with probability at least $1/2n$, so $\mathrm{E}[(\Delta\Phi)^2] \geq 1/2n$. (Clearly the number of points on a path cannot exceed n.) Plugging all of these quantities into Lemma 3.4 yields $T \leq 2n^3$. □

We now extend the above argument to the case of lozenge routings with multiple paths. The coupling we use is the following obvious generalization of our earlier one. Given a pair of routings, choose the same random point p on both, say the ith point of the jth path, and the same random bit $r \in [0, 1]$. Then update each routing by rotating at point p with the appropriate probability as determined by the random number r.

As in the single-path case, we can argue that it is sufficient to bound the expected coupling time for a pair of extremal routings. Recall the partial order defined in the proof of Lemma 3.1, in which routings R_1, R_2 satisfy $R_1 \succeq R_2$ iff the ith path of R_1 lies on or above the ith path of R_2 for all i. As before, there are unique maximum and minimum elements under $\succeq$. And once again it is not hard to check that our coupling is monotone with respect to this partial order.

LEMMA 3.8. *The above coupling for routings is monotone with respect to the partial order* $\succeq$.

Proof. Suppose $R_1 \succeq R_2$ and consider a random move defined by the pair $(p, r) \in [1, \ldots, N] \times \{0, 1\}$. Assume without loss of generality that $r \geq 1/2$. If p is the top of a rotatable tower in R_1, then either R_2 is sufficiently far away from R_1 so that rotating the tower in R_1 does not disturb the order, or p is also the top of a tower in R_2. Because $R_1 \succeq R_2$, it must be the case that the tower defined by p in R_1 has height at least as large as the tower defined by p in R_2. Thus the probability of performing a rotation in R_1 cannot exceed that of performing a rotation in R_2. Therefore, depending on the value of r, we rotate either in both routings, only in R_2, or in neither routing. In each of these cases it is clear that the partial order is preserved. □

For a pair of routings R_1, R_2, we define the distance $\phi(R_1, R_2)$ to be the sum of the areas between corresponding paths in R_1 and R_2. The next lemma, a generalization of Lemma 3.6, proves that the distance tends not to increase under the coupling. Again, in light of Lemma 3.8 we restrict our attention to the case where $R_1 \succeq R_2$.

LEMMA 3.9. *Let* R_1 *and* R_2 *be any two lozenge routings such that* $R_1 \succeq R_2$. *Then* $\mathrm{E}[\Delta\Phi | R_1, R_2] \leq 0$.

Proof. Consider a pair of routings, the upper one solid and the lower one dotted. Adopting the same terminology as in the proof of Lemma 3.6, give a point the label G_i if it defines a tower of height i and is a good point (i.e., choosing it could reduce the area between the two routings). Similarly, label a point B_i if it defines a bad tower of height i (see Figure 11). We can calculate the expected change in area by first considering the expected change if we make a move on the upper (solid) routing only and then adding to this the expected change in area from a move on the lower (dotted) routing only. Each point labeled G_i has probability $1/2Ni$ of rotating, and such a rotation decreases the area between the two routings on *each* of the i paths altered by the rotation, a total decrease of i. Summing the expected changes in area over all good and bad points, we have

$$\begin{aligned}\mathrm{E}[\Delta\Phi | R_1, R_2] &= \sum_i \sum_{B_i} \frac{i}{2Ni} - \sum_j \sum_{G_j} \frac{j}{2Nj} \\ &= \frac{1}{2N}\Big(\sum_i \#B_i - \sum_j \#G_j\Big) \leq 0,\end{aligned}$$

where $\#B_i$ is the number of points labeled B_i. The final inequality follows from the fact that on each path the number of good points is at least as large as the number of bad points, as argued in lemma 3.6. □

It is now a short step to the main theorem of this section, which confirms that simulating the Markov chain $\mathcal{M}_{loz}$ for a small number of steps suffices to generate a random lozenge tiling.

THEOREM 3.10. *Let* S *be a region of the triangular lattice. The mixing time of the Markov chain* $\mathcal{M}_{loz}$ *on lozenge routings of* $\hat{S}$ *satisfies* $\tau(\epsilon) \leq 8en^4\lceil \ln \epsilon^{-1} \rceil$, *where* n *is the area of* $\hat{S}$.

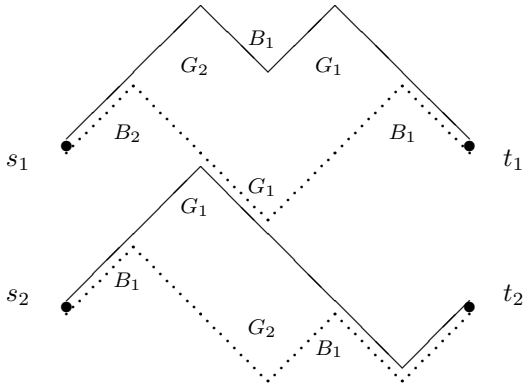


FIG. 11. *Proof of Lemma* 3.9.

Proof. By Theorem 3.3, it suffices to show that the coupling time satisfies $T \leq 8n^4$; and by Lemma 3.8 we need only consider the coupling time for the two extremal initial states. We first bound the distance between these extremal configurations.

Recall two properties of the height function h. First, the heights of all points around the boundary are fixed for all routings of a region. Second, the heights of neighboring vertices differ by at most two. For any internal point v, let k_v be the side length of the largest hexagon centered at v and fully contained within the region. This largest hexagon must share at least one vertex with the boundary of the region, and hence in any tiling the height of v can take on at most $4k_v + 1$ consecutive integer values. Finally, recall that we can walk from the lowest to the highest routing by a sequence of rotations each of which increases the height of one vertex by three (and leaves all other heights unchanged). As we perform this walk, there can be at most $\lceil \frac{4k_v+1}{3} \rceil < 2k_v$ rotations which alter the height of v. If the region being tiled has area n, then $k_v < \sqrt{n}$ for every vertex v. Therefore, summing over all internal vertices, we find that the total distance between the highest and lowest routings is at most $\sum_v 2\sqrt{n} = 2n^{3/2}$.

Lemma 3.9 confirms that $\mathrm{E}[\Delta\Phi] \leq 0$ for all pairs of states reachable under the coupling. To get a bound on $\mathrm{E}[(\Delta\Phi)^2]$, consider a pair of routings with nonzero area between them. Then there must be a pair of corresponding paths, one solid and one dotted, and a segment in which the solid path is strictly above the dotted path. Scanning this segment from left to right, call the first good point we reach on either path p. There is a $1/N$ chance of choosing the point p, and we perform the rotation at p with probability $1/2h$, where h is the height of the tower defined by p. Rotating this tower causes a decrease of h in the total area, one unit for each path included in the tower. Hence we can conclude that $\mathrm{E}[(\Delta\Phi)^2] \geq h^2/2Nh \geq 1/2n$. Putting all this together and appealing to Lemma 3.4, we see that the coupling time satisfies $T \leq 8n^4$. The result now follows from Theorem 3.3. □

Remarks. (a) We have made no attempt here to optimize the upper bound on the mixing time in Theorem 3.10; our concern has been to establish that the chain is *rapidly mixing*, in the sense that the mixing grows only *polynomially* with the area n. (Note in contrast that the size of the state space, i.e., the number of lozenge tilings, is in general *exponential* in n.) Using a more detailed analysis of our coupling, Wilson [18] has derived sharper bounds on the mixing time.

(b) The monotonicity property of our coupling allows one to determine bounds on the coupling time experimentally, simply by simulating the coupled process starting at the two extremal states. In practice, this has been found to yield significantly tighter

bounds than that of Theorem 3.10. This idea can be further extended using the technique of "coupling from the past" due to Propp and Wilson [14] to obtain a stopping rule for the simulation that eliminates all bias from the samples. Theorem 3.10 can be viewed as an a priori bound on the running time of such experiments.

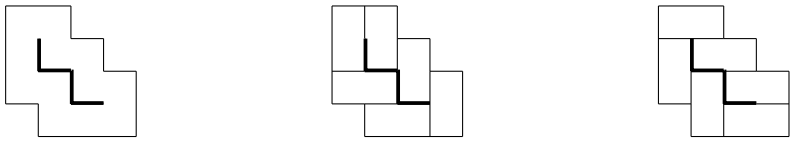
(c) Randall and Tetali [15] recently demonstrated that the comparison technique of Diaconis and Saloff-Coste [5] can be used to relate the mixing time of $\mathcal{M}_{loz}$ to that of the original Markov chain whose only moves are *simple* rotations. Their results, together with Theorem 3.10, imply that the original Markov chain is rapidly mixing as well.

4. Sampling domino tilings. The machinery presented in the last section provides a general framework which can be applied to the random generation of other lattice structures, including domino tilings and Eulerian orientations (and presumably others). In each case, the development of a provably efficient algorithm follows the same outline: we start with natural local moves connecting the space of configurations. We then interpret these moves in terms of the appropriate routings, enrich them with a small set of nonlocal moves (involving towers), and use a coupling argument to argue that the resulting Markov chain is rapidly mixing: in all these cases, the mixing time is bounded by a low-degree polynomial in the area of the region. The definition of towers is sensitive to the type of routing, and the proofs use slightly more sophisticated arguments. It is interesting that, in each case, as for lozenge tilings, the choice of towers is quite natural in the original setting as well.

Our task in this section is to construct a random domino tiling of a given region S of the Cartesian lattice. This we achieve using a Markov chain on the space of domino tilings, whose moves correspond to rotations of suitably defined towers. As before, towers are constructed so as to allow, in a single move, a series of domino rotations which enable a rotation at a particular point which would otherwise be disallowed. Consider, for example, a 2×2 block centered at a point $p = (x, y)$ which has one vertical domino to its right but a horizontal domino covering the square to its lower-left. Then we can either perform a domino rotation at the point $(x-1, y)$, which would then allow a domino rotation at p, or the square to the upper-left of p is covered by a vertical domino. Continuing in a zigzag fashion to the upper-left, we will eventually find a point $q = (x - k, y - k)$ or $q = (x - k - 1, y - k)$ where we can perform a rotation (assuming that we do not hit the boundary). Rotating dominoes along the zigzag, starting at q, we can eventually perform a rotation at p. See Figure 12. As we shall see, allowing these types of tower moves along just one of the two diagonal directions turns out to be sufficient, so we restrict our attention to those running in the NW-SE direction.

To formalize the above, a domino tower is defined by a *spine*, which is a zigzag path (staircase of unit steps along domino boundaries) in the NW-SE direction of a tiling, and the tower includes all the cells that are incident to a vertex on the spine. The endpoints of the spine are the *top* and *bottom* of the tower. Provided that all the vertices along the spine lie in the interior of the region, there are exactly two ways in which the tower can be tiled so that one of the two endpoints of the spine is the center of a 2×2 block tiled with two parallel dominoes. A *rotation* of the tower is an operation that replaces one of these two tilings with the other. There are four distinct types of domino towers, depending on whether the first and last edges of the spine are horizontal or vertical. The *height* of a tower is the number of vertices on the spine.

For the analysis it is more useful to redefine towers in terms of the corresponding routings. Recall that in Theorem 2.2 we showed that generating a random tiling



A spine defining a tower The two tilings of the tower

FIG. 12. *Towers moves for domino tilings.*

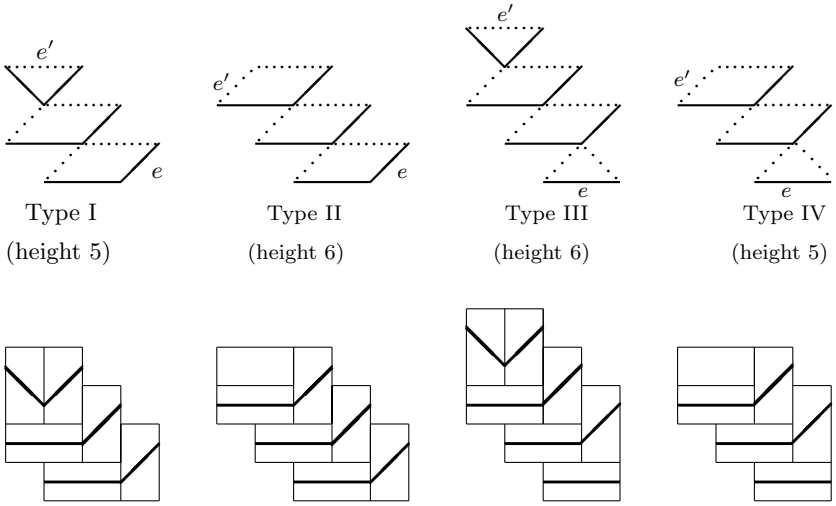


FIG. 13. *The four types of domino towers.*

is equivalent to constructing a random domino routing of a region of the triangular lattice with sources $\{s_1, \ldots, s_k\}$ and sinks $\{t_1, \ldots, t_k\}$. The four types of towers can easily be interpreted in terms of routings, as shown in Figure 13. A move of the Markov chain consists of identifying a tower and then moving between the solid and dotted local structures shown. In the routings representation, the height is just the number of unit triangles in the tower. The edges marked e in the diagram indicate the bottoms of the corresponding towers in the northwest direction, while the edges marked e' are the tops of towers in the southeast direction. (To relate this definition to tilings, if e is a diagonal edge, then it corresponds to a vertical domino such that the center of the left edge is the bottom of the spine defining the tower in the tiling; if e is horizontal, then it corresponds to a horizontal tile such that the center of the top edge is the bottom of the spine. For edges e', the center of the left or bottom edge of the corresponding domino is the top of the spine.) We refer to any of the edges e or e' as the "start" of a tower; notice that a tower can be uniquely specified by its start edge and its direction. The type of the tower merely reflects whether the top and bottom edges are horizontal or diagonal. For example, the original Markov chain based on rotating two neighboring domino tiles consists of all Type I and Type IV tower rotations of height 1.

Let N be the number of edges in any domino routing of $\hat{S}$. The transition prob-

abilities $P(\cdot,\cdot)$ of $\mathcal{M}_{dom}$ are

$$P(R_1,R_2)=\begin{cases} 1/2Nh & \text{if } R_1,R_2 \text{ differ by rotation of a tower of height } h;\\ 1-\displaystyle\sum_{R\neq R_1} P(R_1,R) & \text{if } R_2=R_1. \end{cases}$$

To implement one step of this Markov chain, starting at a routing R, choose $(e,r)\in\{1,\ldots,N\}\times[0,1]$ uniformly at random, where e is a random *edge* of R. First suppose $r\le 1/2$. If the edge e is a diagonal directed up and to the right, check whether there is a tower starting at e and extending in the northwest direction. Notice that this must be a tower of Type I or Type II and is unique. If, on the other hand, e is horizontal, check whether there is a tower of Type III or Type IV starting at e and extending northwest (again this is unique). In either case, determine the height h of the tower, and if $r\le 1/2h$ rotate the tower if possible (i.e., if the tower lies entirely within the region). The case when $r>1/2$ is similar: check whether there is a tower in the southeast direction starting at e. If e is a diagonal pointing up and to the right this would be a tower of Type II or Type IV; if e is horizontal it would be a tower of Type I or Type III. In either case rotate the tower (if possible) if $r>1-1/2h$, where h is the height of the tower.

The Markov chain $\mathcal{M}_{dom}$ can be analyzed using arguments similar to those used in the previous section for $\mathcal{M}_{loz}$. As before, we first need to show that the chain is ergodic.

THEOREM 4.1. *The Markov chain $\mathcal{M}_{dom}$ is ergodic and converges to the uniform distribution over domino tilings.*

Proof. It is easy to check that the Markov chain is symmetric and aperiodic. Therefore, it suffices to show that the Markov chain connects the state space of domino tilings of a region. We will show that the state space is connected even if we restrict the set of allowable transitions to simple domino rotations.

We start by defining a partial order on the set of all domino routings of $\hat{S}$ as follows. Let R_1,R_2 be two routings and let P_1,P_2 be a pair of corresponding paths (with the same source and sink). We say that $P_1\succeq P_2$ iff each vertical line intersecting the paths intersects P_1 at a point at least as high as the intersection with P_2. (Since the paths are piecewise linear, it suffices to check this condition at vertical lines which pass through vertices of the underlying lattice.) As before, we say $R_1\succeq R_2$ iff the relation $P_1\succeq P_2$ holds for all pairs of corresponding paths and define the distance between two routings to be the union of the area between corresponding paths.

Using an argument analogous to that in the proof of Lemma 3.1, it can be verified that there is a unique minimum routing $R_\perp$, and that for any routing $R\neq R_\perp$ we can find a tower of height 1 whose rotation will reduce the distance between R and $R_\perp$. This moves R closer to $R_\perp$ and inductively guarantees the connectedness, and thus the ergodicity, of $\mathcal{M}_{dom}$. □

It remains for us to show that the Markov chain converges rapidly to stationarity. We can define a coupling in exactly analogous fashion to that for lozenge routings in section 3.4; namely, pick a random pair (e,r) and make the appropriate move in both routings simultaneously. Once again it is not hard to check that this coupling is monotone with respect to the partial order on domino routings defined in the proof of Theorem 4.1, so when analyzing the coupling time we need consider only the two extremal routings. The key fact is the following analogue of Lemma 3.9, which says that the area $\Phi(t)$ between routings (defined in the obvious way) is nonincreasing in expectation.

LEMMA 4.2. *Let R_1 and R_2 be two domino routings such that $R_1 \succeq R_2$. Then* $\mathrm{E}[\Delta\Phi | R_1, R_2] \leq 0$.

Proof. First consider the case when R_1 and R_2 each consist of only a single path. Then, following the ideas from Lemmas 3.6 and 3.9, we can consider segments $C_1, \ldots, C_\ell$ on which the two paths coincide, and segments $D_1, \ldots, D_m$ on which the two paths coincide at the endpoints, and on which R_1 is strictly above R_2 at all intermediate points. As before, it is clear that choosing an edge on any of the C_j will not change the area between the paths.

Now consider a region D_j. Give an edge the label G_i if, when that edge is chosen by the coupling, the area between the routings could decrease by exactly i. Similarly, label an edge B_i if one of the directions could cause the area to increase. When the routings consist of a single path, i is either 1 or 2. Certain edges will receive two labels. See Figure 14.

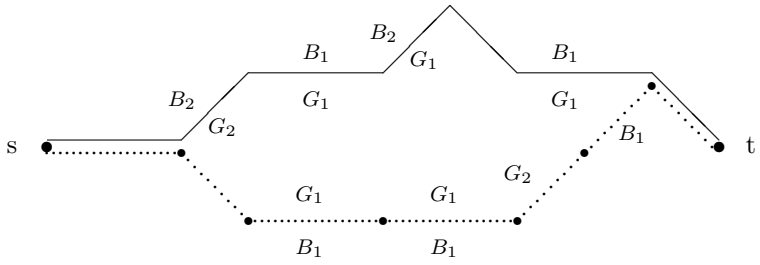


FIG. 14. *Proof of Lemma* 4.2*—single path case.*

We can match each bad edge to a unique good edge as follows. Each horizontal edge with a bad label also has a good label, so we can pair them off. Any diagonal segment which has a bad label at one end must have a good label at the other end, so we can pair these. (The moves of $\mathcal{M}_{dom}$ are defined so that we can rotate only the routing at the two extremal edges of a diagonal segment.) This tells us that the number of bad edges is less than or equal to the number of good edges. As in Lemma 3.9, the probability of rotating at a given edge is inversely proportional to the change in area caused by such a rotation. This ensures that the expected change in area is always nonpositive.

To handle multiple paths, we need to modify the above labeling slightly. It is possible that edges which would be labeled good or bad when a path is viewed in isolation might no longer be places where we can perform a rotation. In particular, if horizontal edges from adjacent paths get too close, one will not be able to rotate in the northwest direction, and the other will not be able to rotate in the southeast direction. This will have the net effect of eliminating a good and a bad label. To see this, label an edge $\hat{G}$ if it is a horizontal edge which cannot move in the good direction because of interference with another path and label it $\hat{B}$ if it cannot move in the bad direction because of interference with another path. These labels are shown in Figure 15 for the solid paths only. The crucial point is that every edge labeled $\hat{G}$ must be paired with a distinct edge labeled $\hat{B}$ and vice-versa; these are unit distance apart in the NE-SW direction. Hence $\#\hat{G} = \#\hat{B}$.

Generalizing the argument from the single path case, we have that

$$\#\hat{B} + \sum_i \#B_i \leq \#\hat{G} + \sum_j \#G_j,$$

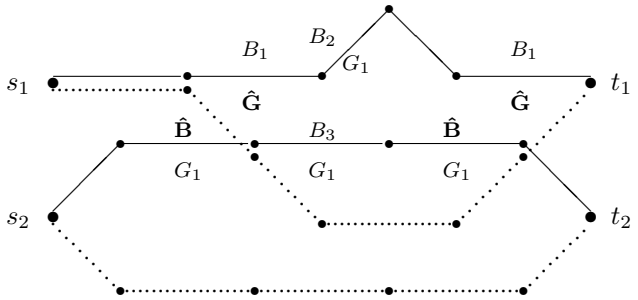


FIG. 15. *Proof of Lemma* 4.2*—multiple path case.*

whence

$$\sum_i \#B_i \le \sum_j \#G_j.$$

From this we can conclude as before that the expected change in area is always nonpositive. □

As a final ingredient, we need to bound the distance between the two extremal routings. For a fixed region with area n, the distance between the highest and lowest routings is at most $2n^{3/2}$. To see this, recall that the difference between the heights of neighboring vertices in a domino routing is at most ± 3. If k_v is the side length of the largest square centered at v which lies within the region, then the height of v is an integer value in $[h(u) - 2k_v, h(u) + 2k_v]$, where u is a vertex which lies on both the square and the boundary of the region. Each step in a walk from the lowest to the highest routing increases the height of some vertex by four, so the total number of steps is at most $\sum_v \frac{6k_v+1}{4} \le \sum_v 2\sqrt{n} = 2n^{3/2}$.

In analogous fashion to the proof of Theorem 3.10, Lemma 4.2, together with Lemma 3.4 and monotonicity of the coupling, implies that the Markov chain $\mathcal{M}_{dom}$ is rapidly mixing.

THEOREM 4.3. *Let S be a region of the Cartesian lattice, and let $\hat{S}$ be the corresponding region containing domino routings. Then the mixing time of the Markov chain $\mathcal{M}_{dom}$ on routings of $\hat{S}$ satisfies $\tau(\epsilon) \le 8en^4\lceil \ln \epsilon^{-1} \rceil$, where n is the area of S.* □

5. Sampling Eulerian orientations. Let S be a region with specified boundary conditions for Eulerian orientations: recall that these determine the sources and sinks in the routings representation. To generate a random Eulerian orientation of S, we construct a Markov chain $\mathcal{M}_{eul}$ whose state space is the set of all Eulerian routings on S with these sources and sinks. In similar fashion to the case of lozenge routings, moves will be defined in terms of "peaks" and "valleys," where in this case a valley is a vertex which is the right endpoint of a horizontal edge and the bottom endpoint of a vertical edge of the routing; a peak is a vertex which is the left endpoint of a horizontal edge and the top endpoint of a vertical edge. Note that a vertex may be both a peak and a valley at the same time. The operation of flipping a peak into a valley (or vice versa) where possible defines a simple Markov chain which we argue in Theorem 5.1 connects the state space.

For our analysis we augment this Markov chain by allowing a move between two routings if they differ by a structure which is either a vertical or horizontal tower as

depicted in Figure 16. More precisely, let the *cell at* (x, y) be the four edges of the unit square of the lattice whose lower-right corner is at (x, y). Define a *horizontal tower of height* h to be a set of cells at points $(x, y), (x-1, y), \ldots, (x-h+1, y)$, where either $(x, y), (x-1, y), \ldots, (x-h+1, y)$ are all valleys and $(x, y+1), (x-h, y)$ are not valleys, or $(x-h, y+1), \ldots, (x-1, y+1)$ are all peaks and $(x-h, y), (x, y+1)$ are not peaks. The point (x, y) is the *bottom* of the tower and $(x-h, y+1)$ is the *top* of the tower. Similarly, define a *vertical tower of height* h to be a set of cells at points $(x, y), (x, y+1), \ldots, (x, y+h-1)$, where either $(x, y), \ldots, (x, y+h-1)$ are all valleys and $(x-1, y), (x, y+h)$ are not valleys, or $(x-1, y+h), \ldots, (x-1, y+1)$ are all peaks and $(x, y+h), (x-1, y)$ are not peaks. In this case (x, y) is the bottom of the tower and $(x-1, y+h)$ is the top (see Figure 16). Note that vertical and horizontal towers of height 1 are identical and that any vertex can be the top (respectively, bottom) of at most one tower. The reader should be able to verify that any tower can be inverted in the obvious way to produce a valid routing provided all of its cells are contained within the region. As usual, we call such an operation a *rotation* of the tower.

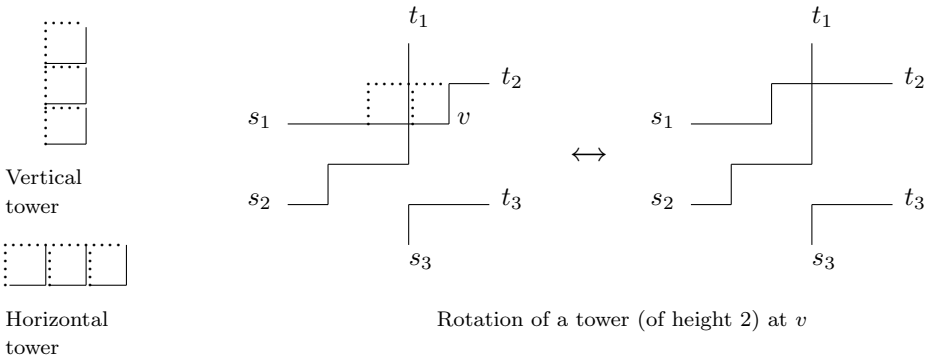


FIG. 16. *Markov chain for Eulerian orientations.*

The transition probabilities $P(\cdot\,, \cdot)$ of $\mathcal{M}_{eul}$ are defined as follows:

$$P(R_1, R_2) = \begin{cases} 1/2Nh & \text{if } R_1, R_2 \text{ differ by rotation of a tower of height } h; \\ 1 - \displaystyle\sum_{R \neq R_1} P(R_1, R) & \text{if } R_2 = R_1, \end{cases}$$

where N is the number of vertices on any routing.

To implement one step of $\mathcal{M}_{eul}$, starting at a routing R, choose $(p, r) \in \{1, \ldots, N\} \times [0, 1]$ uniformly at random, where p denotes a point on the routing. If $r \leq 1/2$, check whether p is the bottom of a tower of height h and rotate this tower if possible if $r < 1/2h$. Similarly, if $r > 1/2$ check whether p is the top of a tower of height h and rotate this tower if possible if $r > 1 - 1/2h$.

Tower rotations have a simple interpretation in terms of Eulerian orientations. A tower of height h corresponds to a directed cycle in the Eulerian orientation which is a $1 \times h$ or $h \times 1$ rectangle (with its internal edges aligned). The rotation corresponds to reversing all the orientations around this cycle, which of course produces a new Eulerian orientation.

THEOREM 5.1. *The Markov chain $\mathcal{M}_{eul}$ is ergodic and converges to the uniform distribution over Eulerian orientations.*

Proof. Since the Markov chain is symmetric and aperiodic, it is again sufficient to show that it is connected. As in the case of domino and lozenge tilings, the key to proving this fact is to define a suitable partial order on routings. For corresponding paths P_1, P_2 on two routings R_1, R_2, we say that $P_1 \succeq P_2$ if each diagonal line in the NW-SE direction intersects P_1 at a point at least as far left as P_2. We say $R_1 \succeq R_2$ iff $P_1 \succeq P_2$ holds for each pair of corresponding paths P_1, P_2. Under this partial order there will always be a unique minimum routing. An argument analogous to that in the proof of Lemma 3.1 demonstrates that, for any routing R which is not minimum, we can find a point which is the top of a rotatable tower whose rotation reduces the distance between R and the minimum routing (where again distance is defined as the sum of the area between corresponding paths). This guarantees, inductively, that the state space is connected. □

We now turn to the mixing time of the Markov chain. Note that when the boundary conditions define a set of routings consisting of a single path, the Markov chain is equivalent to the lozenge routing chain $\mathcal{M}_{loz}$ of section 3. Theorem 3.7 guarantees that this chain is rapidly mixing. We now extend this result to Eulerian routings that consist of an arbitrary number of paths. As in the previous examples, we will construct a coupling that chooses a random point p and a random number r and makes the appropriate move in both routings simultaneously. Again as before, this coupling is easily seen to be monotone with respect to the partial order defined in the proof of Theorem 5.1, so we need analyze only the coupling time starting from the two extremal routings. As usual the following lemma, which says that the area $\Phi(t)$ between routings is nonincreasing in expectation, is the key to the proof.

LEMMA 5.2. *Let R_1 and R_2 be any two Eulerian routings such that $R_1 \succeq R_2$. Then* $\mathrm{E}[\Delta\Phi | R_1, R_2] \leq 0$.

Proof. We assign labels G and B to peaks and valleys in similar fashion to the lozenge case (see Figure 17). On the lower routing, label G any corner which is the lower-right of a tower and label B any corner which is the upper-left of a tower; on the upper routing, interchange the roles of G and B. Finally we will label all the corners which do not define a tower because of interference with another path. On the lower routing, label $\hat{G}$ any unlabeled corner which goes to the right and then up if the two edges which complete the unit square are part of another path on the routing. Likewise, label $\hat{B}$ any unlabeled corner which goes up and then to the right but which is not part of a tower because the two edges completing the square are part of anther path on the routing. On the upper routing, interchange the roles of $\hat{G}$ and $\hat{B}$. Generalizing from the single path case as in Lemma 3.9, we know that $\#B + \#\hat{B} \leq \#G + \#\hat{G}$.

Next observe that each point labeled $\hat{G}$ is paired with a distinct point labeled $\hat{B}$. A point p is labeled $\hat{G}$ if the two edges which complete the unit square defined by the peak or valley at p belong to an adjacent path on the same routing. The point defining the opposite corner of the square must be labeled $\hat{B}$, and we can pair this bad point (on the same routing) with p. This pairing implies that $\#\hat{B} = \#\hat{G}$, and hence that $\#B \leq \#G$.

Finally, observe that the weights of the moves are chosen so that each bad and each good point contributes equally to the expected change in area. Hence, if N is the total number of points on the routing, we have $\mathrm{E}[\Delta\Phi] = (\#B - \#G)/2N \leq 0$. □

A simple calculation based on the height function representation reveals that the distance between extremal Eulerian routings of a region with area n is at most $n^{3/2}$.

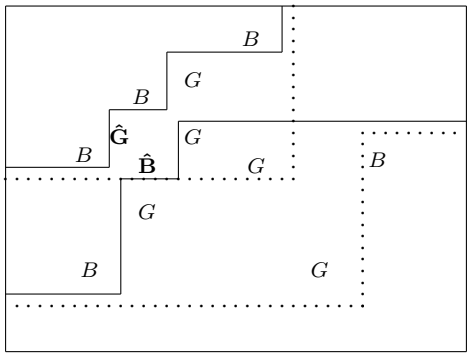


FIG. 17. *Proof of Lemma* 5.2.

Combining Lemmas 5.2 and 3.4 and the fact that the coupling is monotone, we can deduce in by now familiar fashion that the Markov chain $\mathcal{M}_{eul}$ is rapidly mixing.

THEOREM 5.3. *Let S be a region of the Cartesian lattice with specified boundary conditions. Then the mixing time of the Markov chain $\mathcal{M}_{eul}$ on Eulerian routings of S satisfies $\tau(\epsilon) \leq 2en^4\lceil \ln \epsilon^{-1} \rceil$, where n is the area of S.* □

Acknowledgment. We would like to thank James Akao for useful discussions and for implementing the Eulerian orientations algorithm.

REFERENCES

[1] K. AL-KHNAIFES, AND H. SACHS, *Graphs, linear equations, determinants, and the number of perfect matchings*, in Contemporary Methods in Graph Theory, R. Bodendiek, ed., Wissenschaftsverlag, Mannheim, 1990, pp. 47–71.

[2] D. ALDOUS, *Random walks on finite groups and rapidly mixing Markov chains*, in Séminaire de Probabilités XVII, 1981/82, Lecture Notes in Math. 986, Springer-Verlag, Berlin, 1983, pp. 243–297.

[3] R. J. BAXTER, *Exactly Solved Models in Statistical Mechanics*, Academic Press, London, 1982.

[4] J. CONWAY AND J. LAGARIAS, *Tilings with polyominoes and combinatorial group theory*, J. Combin. Theory Ser. A, 53 (1990), pp. 183–208.

[5] P. DIACONIS AND L. SALOFF-COSTE, *Comparison theorems for reversible Markov chains*, Ann. Appl. Probab., 3 (1993), pp. 696–730.

[6] R. DURRETT, *Probability: Theory and Examples*, Wadsworth & Brooks/Cole, Belmont, CA, 1991.

[7] I. GESSEL AND G. VIENNOT, *Binomial determinants, paths, and hook length formulae*, Adv. in Math., 58 (1985), pp. 300–321.

[8] O. J. HEILMANN AND E. H. LIEB, *Theory of monomer-dimer systems*, Comm. Math. Phys., 25 (1972), pp. 190–232.

[9] M. JERRUM AND A. SINCLAIR, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[10] W. JOCKUSCH, J. PROPP, AND P. SHOR, *Random Domino Tilings and the Arctic Circle Theorem*, preprint, 1995.

[11] P. JOHN AND H. SACHS, *Calculating the numbers of perfect matchings and of spanning trees, Pauling's orders, the characteristic polynomial, and the eigenvectors of a benzenoid system*, in Topics in Current Chemistry, M. J. S. Dewar et al., eds., Advances in the Theory of Benzenoid Hydrocarbons 153, Springer-Verlag, Berlin, 1990, pp. 145–179.

[12] C. KENYON, D. RANDALL, AND A. SINCLAIR, *Approximating the number of dimer coverings of a lattice*, J. Statist. Phys., 83 (1996), pp. 637–659.

[13] M. MIHAIL AND P. WINKLER, *On the number of Eulerian orientations of a graph*, in Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 138–145.

[14] J. Propp and D. Wilson, *Exact sampling with coupled Markov chains and applications to statistical mechanics*, Random Structures Algorithms, 9 (1996), pp. 223–252.
[15] D. Randall and P. Tetali, *Analyzing Glauber dynamics by comparisons of Markov chains*, J. Math. Phys., 41 (2000), pp. 1598–1615.
[16] A. Sinclair, *Counting and Generating Combinatorial Structures: A Markov Chain Approach*, Birkhäuser, Boston, 1993.
[17] W. Thurston, *Conway's tiling groups*, Amer. Math. Monthly, 97 (1990), pp. 757–773.
[18] D. B. Wilson, *Mixing Times of Lozenge Tiling and Card Shuffling Markov Chains*, manuscript, 1997; also available online from http://research.microsoft.com/∼dbwilson/nlm.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 193–198

THERE ARE NO SPARSE NP_W-HARD SETS*

FELIPE CUCKER† AND DIMA GRIGORIEV‡

Abstract. In this paper we prove that, in the context of weak machines over $\mathbb{R}$, there are no sparse NP-hard sets.

Key words. structural complexity, real number computations

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

PII. S0097539700379346

1. Introduction. In [2] Berman and Hartmanis conjectured that all NP-complete sets are polynomially isomorphic. That is, that for all NP-complete sets A and B, there exists a bijection $\varphi : \Sigma^* \to \Sigma^*$ such that $x \in A$ if and only if $\varphi(x) \in B$. In addition, both φ and its inverse are computable in polynomial time. Here Σ denotes the set $\{0,1\}$ and Σ^* the set of all finite sequences of elements in Σ.

Should this conjecture be proved, we would have as a consequence that no "small" NP-complete set exists in a precise sense of the word "small." A set $S \subseteq \Sigma^*$ is said to be *sparse* when there is a polynomial p such that for all $n \in \mathbb{N}$, the subset S_n of all elements in S having size n has cardinality at most $p(n)$. If the Berman–Hartmanis conjecture is true, then there are no sparse NP-complete sets.

In 1982 Mahaney [11] proved this weaker conjecture by showing that there exist sparse NP-hard sets if and only if P = NP. After this, a whole stream of research developed around the issue of reductions to "small" sets (see [1]).

In a different line of research, Blum, Shub, and Smale (BSS) introduced in [4] a theory of computability and complexity over the real numbers with the aim of modelling the kind of computations performed in numerical analysis. The computational model defined in that paper deals with real numbers as basic entities and performs arithmetic operations on them as well as sign tests. Inputs and outputs are vectors in $\mathbb{R}^n$ and decision problems are subsets of $\mathbb{R}^\infty$, the disjoint union of $\mathbb{R}^n$ for all $n \geq 1$. The classes $\mathrm{P}_\mathbb{R}$ and $\mathrm{NP}_\mathbb{R}$—which are analogous to the well-known classes P and NP—are then defined, and one of the main results in [4] is the existence of natural $\mathrm{NP}_\mathbb{R}$-complete problems.

Clearly, the sparseness notion defined above for sets over $\{0,1\}$ will not define any meaningful class over $\mathbb{R}$ since now the set of inputs of size n is $\mathbb{R}^n$, and this is an infinite set. A notion of sparseness over $\mathbb{R}$ capturing the main features of the discrete one (independence of any kind of computability notion and capture of a notion of "smallness"), however, was proposed in [6]. Let $S \subseteq \mathbb{R}^\infty$. We say that S is *sparse* if, for all $n \geq 1$, the set

$$S_n = \{x \in S \mid x \in \mathbb{R}^n\}$$

*Received by the editors October 12, 2000; accepted for publication (in revised form) March 12, 2001; published electronically July 25, 2001.
http://www.siam.org/journals/sicomp/31-1/37934.html

†Department of Mathematics, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (macucker@math.cityu.edu.hk). This author was partially supported by CERG grant 9040393.

‡IMR, Université de Rennes I, Campus de Beaulieu, Rennes 35042, France (dima@maths.univ-rennes1.fr).

has dimension at most $\log^q n$ for some fixed q. Here dimension is the dimension, in the sense of algebraic geometry, of the Zariski closure of S_n. Note that this notion of sparseness parallels the discrete one in a very precise way. For a subset $S_n \subseteq \{0,1\}^n$ its cardinality gives a measure of its size, and a sparse set is one for which, for all n, this cardinality is polylogarithmic in the largest possible (i.e., 2^n the cardinality of $\{0,1\}^n$). For a subset $S_n \subseteq \mathbb{R}^n$, we take the dimension to measure the size of S_n and again define sparseness by the property of having this measure be polylogarithmic in the largest possible (which is now n, the dimension of $\mathbb{R}^n$).

Using this definition of sparseness for subsets of $\mathbb{R}^\infty$, the main result of [6] proves that there are no sparse NP-complete sets in the context of machines over $\mathbb{R}$ which do not perform multiplications or divisions and branch only on equality tests. Note that this result is not conditioned to the inequality $\mathrm{P} \neq \mathrm{NP}$ since this inequality is known to be true in this setting (cf. [12]).

A variation on the BSS model attempting to get closer to the Turing machine (in the sense that iterated multiplication is somehow penalized) was introduced by Koiran in [10]. This model, which Koiran called *weak*, takes inputs from $\mathbb{R}^\infty$ but no longer measures the cost of the computation as the number of arithmetic operations performed by the machine. Instead, the cost of each individual operation $x \circ y$ depends on the sequences of operations which lead to the terms x and y from the input data and the machine constants.

In this paper we extend Mahaney's theorem to machines over $\mathbb{R}$ endowed with the weak cost. Again, this is not a conditional result since it is known that $\mathrm{P} \neq \mathrm{NP}$ in this context too (cf. [7]). If NP_W denotes the class of sets decided in nondeterministic polynomial cost, our main result is the following.

THEOREM 1.1. *There are no sparse* NP_W*-hard sets.*

2. The weak cost. Let M be a machine over $\mathbb{R}$, let $\alpha_1, \ldots, \alpha_s$ be its constants, and let $a = (\alpha_1, \ldots, \alpha_s) \in \mathbb{R}^s$. Let $x = (x_1, \ldots, x_n) \in \mathbb{R}^\infty$. At any step ν of the computation of M with input x, the intermediate value $z \in \mathbb{R}$ produced in this step can be written as a rational function of a and x, $z = \varphi(a, x)$. This rational function depends only on the computation path followed by M up to ν (i.e., on the sequence steps previously performed by M) and is actually a coordinate of the composition of the arithmetic operations performed along this path (see [3] for details). Let $\varphi = \frac{g_\nu}{h_\nu}$ be the representation of φ obtained by retaining numerators and denominators in this composition. For example, the representation of the product $\frac{g}{h} \cdot \frac{r}{s}$ is always $\frac{gr}{hs}$, and the representation of the addition $\frac{g}{h} + \frac{r}{s}$ is always $\frac{gs+hr}{hs}$. We will now use g_ν and h_ν to define weak cost.

DEFINITION 2.1. *The* weak cost *of any step ν is defined to be the maximum of* $\deg(g_\nu)$, $\deg(h_\nu)$, *and the maximum bit size of the coefficients of g_ν and h_ν. For any $x \in \mathbb{R}^\infty$ the* weak cost of M on x *is defined to be the sum of the costs of the steps performed by M with input x.*

The class P_W of sets decided within *weak polynomial cost* is now defined by requiring that for each input of size n the weak cost of its computation is bounded by a polynomial in n. A set S is decided in *weak nondeterministic polynomial cost* (we write $S \in \mathrm{NP}_W$) if there is a machine M working within weak polynomial cost satisfying the following: for each $x \in \mathbb{R}^\infty$, $x \in S$ if and only if there is $y \in \mathbb{R}^\infty$ with size polynomial in n such that M accepts the pair (x, y).

REMARK 1. *The definitions above do not fully coincide with those given in* [10] *since this reference requires the representation of the rational functions φ above to be relatively prime. The definitions we give here, which are taken from* [3]*, are essentially*

equivalent. For if a set is in P_W *with the definition above, it is clearly in* P_W *with Koiran's. The converse is more involved to prove. Roughly speaking, any machine can be simulated by another which keeps "programs" instead of performing the arithmetic operations at the computation nodes. When the computation reaches a branch node, the program for the register whose value is tested for positivity is evaluated at the pair* (a, x) *to decide such positivity. Now note that one can use algorithms of symbolic computation to make the numerator and denominator of the rational function computed by the program relatively prime* before *evaluating.*

3. Proof of the main result. Let $n \geq 1$. Consider the polynomial

$$f_n = x_1^{2^n} + \cdots + x_n^{2^n} - 1$$

and let $C_n = \{x \in \mathbb{R}^n \mid f_n(x) = 0\}$. The polynomial f_n is irreducible and the dimension of C_n is $n-1$. Let $C \subset \mathbb{R}^\infty$ be given by $C = \cup C_n$. We know (cf. [7]) that $C \in \mathrm{NP}_W$ but $C \notin \mathrm{P}_W$.

Let $S \subset \mathbb{R}^\infty$ be a NP$_W$-hard set. Then C reduces to S. That is, there exists a function $\varphi : \mathbb{R}^\infty \to \mathbb{R}^\infty$ computable with polynomial cost such that, for all $x \in \mathbb{R}^\infty$, $x \in C \iff \varphi(x) \in S$. For each $n \geq 1$, the restriction of φ to $\mathbb{R}^n$ is a piecewise rational function. Our first result, Proposition 3.2, gives some properties of this function. It uses the following simple fact in real algebraic geometry whose proof can be found in Chapter 19 of [3].

PROPOSITION 3.1. *Let* $f \in \mathbb{R}[x_1, \ldots, x_n]$ *be an irreducible polynomial such that the dimension of its zero set* $\mathcal{Z}(f) \subseteq \mathbb{R}^n$ *is* $n-1$. *Then, for any polynomial* $g \in \mathbb{R}[x_1, \ldots, x_n]$, g *vanishes on* $\mathcal{Z}(f)$ *if and only if* g *is a multiple of* f. □

PROPOSITION 3.2. *Let* n *be sufficiently large. There exist* $x \in C_n$ *and* $U \subset \mathbb{R}^n$, *an open ball centered at* x *such that the restriction of* φ *to* U *is a rational map* $h : U \to \mathbb{R}^m$ *for some* m *bounded by a polynomial in* n. *In addition, if* $h_1, \ldots, h_m$ *are the coordinates of* h, *then the degrees of the numerator and denominator of* h_i *are also bounded by a polynomial in* n *for* $i = 1, \ldots, m$.

Proof. Let M be a machine computing φ within weak polynomial cost. By unwinding the computation of M in a standard manner we obtain an algebraic computation tree of depth polynomial in n. To each branch η in this tree, one associates a set $D_\eta \subseteq \mathbb{R}^n$ such that the D_η partition $\mathbb{R}^n$ (i.e., $\cup D_\eta = \mathbb{R}^n$ and $D_\eta \cap D_\gamma = \emptyset$ for $\eta \neq \gamma$). In addition, each branch η computes a rational map h_η and $\varphi_{|D_\eta} = h_\eta$. The set D_η is the set of points in $\mathbb{R}^n$ satisfying a system

$$\bigwedge_{i=1}^{s_\eta} q_i(x_1, \ldots, x_n) \geq 0 \wedge \bigwedge_{i=s_\eta+1}^{t_\eta} q_i(x_1, \ldots, x_n) < 0, \tag{3.1}$$

where the $q_i(X_1, \ldots, X_n)$ are the rational functions tested along the branch. Since M works within weak cost, the numerators and denominators of the q_i, as well as those of h_η, have degrees bounded by a polynomial in n.

Everything we need to see now is that for some branch η, D_η contains an open neighborhood of a point $x \in C_n$.

To do so, first notice that, by replacing each q_i by the product of its numerator and denominator, we can assume that the q_i are polynomials. Also, by writing $q_i \geq 0$ as $q_i = 0 \vee q_i > 0$ and distributing the disjunctions in (3.1), we can express D_η as a finite union of sets satisfying a system

$$\bigwedge_{i=1}^{s} q_i(x_1, \ldots, x_n) = 0 \wedge \bigwedge_{i=s+1}^{t} q_i(x_1, \ldots, x_n) < 0. \tag{3.2}$$

We have thus described $\mathbb{R}^n$ as a union of sets which are solutions of systems like (3.2). Since this union is finite there exists one such set D containing a subset H of C_n of dimension $n-1$. Let D be the solution of a system like (3.2). We claim that there are no equalities in such a system. Assume the contrary. Then there is a polynomial q such that $H \subset \mathcal{Z}(q)$. Since $\dim H = n-1$ and C_n is irreducible, this implies that $q(C_n) = 0$ and, by Proposition 3.1, that q is a multiple of f_n. Since $\deg f_n = 2^n$, this is not possible for sufficiently large n.

The above implies that D is an open set from which the statement follows. □

For the next result we keep the notation of the statement of Proposition 3.2.

PROPOSITION 3.3. *Let $k = \dim h(U)$.*

(i) *There exist indices $i_1, \ldots, i_k \in \{1, \ldots, m\}$, a polynomial $g \in \mathbb{R}[y_1, \ldots, y_k]$, and a rational function $q \in \mathbb{R}(x_1, \ldots, x_n)$ with both numerator and denominator relatively prime with f_n such that*

$$g(h_{i_1}, \ldots, h_{i_k}) = f_n^\ell q$$

for some $\ell > 0$.

(ii) *Let n be sufficiently large. Then $k \geq n$.*

Proof. For part (i), first notice that since $\dim(h(U)) = k$, there exist $i_1, \ldots, i_k \in \{1, \ldots, m\}$ such that the functions $h_{i_1}, \ldots, h_{i_k}$ are algebraically independent. We want to show that $\dim(U \cap C_n) < k$. To do so let $X = h(U), Y = h(U - C_n)$, and $Z = h(U \cap C_n)$. We have that all X, Y, and Z are semialgebraic subsets of $\mathbb{R}^m$. In addition, Z is contained in the closure of Y with respect to the Euclidean topology relative to X since h is continuous, and $Y \cap Z = \emptyset$ since h is the restriction of φ to U and φ is a reduction.

From here it follows that Z is included in the boundary of Y relative to X. Hence, $\dim Z < \dim Y = \dim X$ (see, e.g., Proposition 2.8.12 of [5]).

The above shows that $\dim h(U \cap C_n) < k$. Therefore, there exists $g \in \mathbb{R}[y_1, \ldots, y_k]$ such that, for all $x \in U \cap C_n$, $g(h_{i_1}(x), \ldots, h_{i_k}(x)) = 0$. Write this as a rational function $g(h) = a/b$ with $a, b \in \mathbb{R}[x_1, \ldots, x_n]$ relatively prime. Then $a(C_n) = 0$ and $a \neq 0$ (since $h_{i_1}, \ldots, h_{i_k}$ are algebraically independent). By Proposition 3.1 this implies that there exists $r \in \mathbb{R}[x_1, \ldots, x_n]$ such that $a = r f_n$. If ℓ is the largest power of f_n dividing a, then the result follows by taking $q = \frac{r'}{b}$, where r' is the quotient of r divided by $f_n^{\ell-1}$.

We now proceed to part (ii). To simplify notation, assume that $i_j = j$ for $j = 1, \ldots, k$. Also, let d be a bound for the degrees of the numerators and denominators of the h_j. Recall from Proposition 3.2 that d is bounded by a polynomial in n.

By part (i) there exists $q \in \mathbb{R}(x_1, \ldots, x_n)$ relatively prime with f_n such that

$$f_n^\ell q = g(h_1, \ldots, h_k)$$

for a certain $\ell \geq 1$. Taking derivatives on both sides we obtain that, for all $x \in \mathbb{R}^n$,

$$\nabla(f_n^\ell q)(x) = \nabla(g)(h(x)) \circ Dh(x), \tag{3.3}$$

where ∇ denotes the gradient and $Dh(x)$ is the Jacobian matrix of h at x.

Assume that $k < n$. Transposing (3.3) one sees that $\nabla(f^\ell q)(x)$ is the image of a vector of dimension k. Thus, there exists a linear dependency among the first $k+1$ coordinates of $\nabla(f^\ell q)(x)$,

$$\sum_{i=1}^{k+1} \lambda_i \frac{\partial f_n^\ell q}{\partial x_i} = 0, \tag{3.4}$$

and the coefficients λ_i of this linear dependency are the determinants of some minors of $Dh(x)$. Thus, for $i = 1, \ldots, k+1$, λ_i is a rational function of x whose numerator and denominator have degrees bounded by kd. Since the submatrix of $Dh(x)$ obtained by keeping its first $k+1$ rows contains at most $k(k+1)$ different denominators, multiplying (3.4) by the product of all of them allows one to assume that the λ_i are polynomials with degree at most $kd(k+1)$.

By the product rule we get

$$\sum_{i=1}^{k+1} \lambda_i \left(\ell f_n^{\ell-1} q \frac{\partial f_n}{\partial x_i} + f_n^{\ell} \frac{\partial q}{\partial x_i} \right) = 0,$$

i.e.,

$$\ell f_n^{\ell-1} q \sum_{i=1}^{k+1} \lambda_i \frac{\partial f_n}{\partial x_i} + f_n^{\ell} \sum_{i=1}^{k+1} \lambda_i \frac{\partial q}{\partial x_i} = 0.$$

Since f_n^{ℓ} divides the second term above, it must also divide the first from which, using that f_n and q are relatively prime, it follows that f_n divides $\sum_{i=1}^{k+1} \lambda_i \frac{\partial f_n}{\partial x_i}$. That is, there exists a polynomial p such that

$$f_n p = \sum_{i=1}^{k+1} \lambda_i \frac{\partial f_n}{\partial x_i},$$

i.e.,

$$p \left(\sum_{i=1}^{n} x_i^{2^n} - 1 \right) = 2^n \sum_{i=1}^{k+1} \lambda_i x_i^{2^n - 1}.$$

Now, for n large enough, the degrees of the λ_i are smaller than $2^n - 1$ since $kd(k+1)$ is polynomial in n. This implies that the degree of p must also be bounded by $kd(k+1)$. Then, however, for each $i \leq k+1$, $px_i^{2^n} = \lambda_i x_i^{2^n - 1}$, i.e., $px_i = \lambda_i$. And from here it follows that $-p = 0$, a contradiction. □

Theorem 1.1 now readily follows. For all $n \in \mathbb{N}$, Proposition 3.2 ensures the existence of an open ball $U \subset \mathbb{R}^n$ whose image by the reduction φ is included in $\mathbb{R}^m$ with m polynomially bounded on n. However, for all n sufficiently large, this image, by Proposition 3.3(ii), has dimension at least n and therefore it cannot be polylogarithmic on m.

REMARK 2. *The result of Theorem* 1.1*, together with that in* [6]*, supports the conjecture that there are no sparse* NP*-hard sets over the reals unless* P = NP*. There are two main settings where this remains to be proved. On the one hand, there are machines which do not multiply nor divide but which branch over sign tests. On the other hand, there is the unrestricted case in which the machine can multiply or divide (and branch over sign tests) with unit cost. In these two cases, the result seems harder since there is no proof that* P ≠ NP*. In the first case, we would like to remark that if many-one reductions are replaced by Turing reductions and we assume that* P ≠ NP*, then Mahaney's conjecture is false. This is due to a result of Fournier and Koiran* [9] *proving that any* NP*-complete set in the Boolean setting (i.e., over* $\{0,1\}$*) is* NP*-complete over the reals with addition and order for Turing reductions. Since the subsets of elements of size n of any such set S have dimension* 0*, the sparseness of S is immediate. For more on this see* [8].

REFERENCES

[1] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf, *Reductions to sets of low information content*, in Complexity Theory: Current Research, K. Ambos-Spies, S. Homer, and U. Schöning, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 1–45.

[2] L. Berman and J. Hartmanis, *On isomorphism and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.

[3] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer-Verlag, New York, 1998.

[4] L. Blum, M. Shub, and S. Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*, Bull. Amer. Math. Soc. (N.S.), 21 (1989), pp. 1–46.

[5] J. Bochnak, M. Coste, and M.-F. Roy, *Géométrie algébrique réelle*, Springer-Verlag, Berlin, 1987.

[6] F. Cucker, P. Koiran, and M. Matamala, *Complexity and dimension*, Inform. Process. Lett., 62 (1997), pp. 209–212.

[7] F. Cucker, M. Shub, and S. Smale, *Complexity separations in Koiran's weak model*, Theoret. Comput. Sci., 133 (1994), pp. 3–14.

[8] H. Fournier, *Sparse NP-complete problems over the reals with addition*, Theoret. Comput. Sci., 255 (2001), pp. 607–610.

[9] H. Fournier and P. Koiran, *Lower bounds are not easier over the reals: Inside PH*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 832–843.

[10] P. Koiran, *A weak version of the Blum, Shub and Smale model*, J. Comput. System Sci., 54 (1997), pp. 177–189.

[11] S. Mahaney, *Sparse complete sets for NP: Solution of a conjecture by Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.

[12] K. Meer, *A note on a* $\mathrm{P} \neq \mathrm{NP}$ *result for a restricted class of real machines*, J. Complexity, 8 (1992), pp. 451–453.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 199–211

RANDOMNESS AND RECURSIVE ENUMERABILITY*

ANTONÍN KUČERA† AND THEODORE A. SLAMAN‡

Abstract. One recursively enumerable real α dominates another one β if there are nondecreasing recursive sequences of rational numbers $(a[n] : n \in \omega)$ approximating α and $(b[n] : n \in \omega)$ approximating β and a positive constant C such that for all n, $C(\alpha - a[n]) \geq (\beta - b[n])$. See [R. M. Solovay, *Draft of a Paper (or Series of Papers) on Chaitin's Work,* manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1974, p. 215] and [G. J. Chaitin, *IBM J. Res. Develop.*, 21 (1977), pp. 350–359]. We show that every recursively enumerable random real dominates all other recursively enumerable reals. We conclude that the recursively enumerable random reals are exactly the Ω-numbers [G. J. Chaitin, *IBM J. Res. Develop.*, 21 (1977), pp. 350–359]. Second, we show that the sets in a universal Martin-Löf test for randomness have random measure, and every recursively enumerable random number is the sum of the measures represented in a universal Martin-Löf test.

Key words. random real, Chaitin, Kolmogorov, Ω-number

AMS subject classifications. 68Q30, 03D15

PII. S0097539799357441

1. Introduction.

When is a real number effectively random? To a large extent, this question was answered by the collective efforts of Chaitin [4], Kolmogorov [11], Martin-Löf [14], Schnorr [15], Solomonoff [16], [17], and Solovay [18], among others. We present a brief historical account, based in the most part on [19]. One could also consult [1] or [13].

1.1. Characterizations of effective randomness.

To fix some notation, Σ^* denotes the set of finite binary sequences. For $a \in \Sigma^*$, $|a|$ denotes the length of a and $\langle a \rangle$ denotes the rational number with binary expansion $0.a$. We order Σ^* lexicographically.

Σ^ω denotes the set of all infinite binary sequences. As above, $\langle \alpha \rangle$ denotes the real number with binary expansion $0.\alpha$. We extend the lexicographic ordering of Σ^* to that on Σ^ω.

For $A \subseteq \Sigma^*$, $A\Sigma^\omega$ denotes the open subset of Σ^ω whose elements have an initial segment in A, and $\mu(A\Sigma^\omega)$ denotes the measure of $A\Sigma^\omega$.

We have chosen to work with Σ^* and Σ^ω, as that seemed to work best notationally. We could have worked with $\mathbb{Q}$ and $\mathbb{R}$ just as well and come to the same conclusions. We will refer to elements of $\mathbb{R}$ and to elements of Σ^ω as real numbers.

Characterization by measure. Our first characterizations of effective randomness are based on the hypothesis that an effectively random real should avoid every effectively presented set of measure 0.

DEFINITION 1.1 (Martin-Löf [14]).

1. *A* Martin-Löf randomness test *is a uniformly recursively enumerable sequence* $(A_n : n \geq 1)$ *of subsets of* Σ^* *such that for each* n, $\mu(A_n\Sigma^\omega) \leq 1/2^n$.

*Received by the editors June 21, 1999; accepted for publication (in revised form) March 12, 2001; published electronically July 25, 2001.

http://www.siam.org/journals/sicomp/31-1/35744.html

†Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic (kucera@ksi.mff.cuni.cz).

‡Department of Mathematics, University of California, Berkeley, CA 94720-3840 (slaman@math.berkeley.edu). This author was partially supported by National Science Foundation grant DMS-9500878.

2. *An x in Σ^ω is* Martin-Löf-random *if for every Martin-Löf test $(A_n : n \geq 1)$, $x \notin \bigcap_{n\geq 1} A_n\Sigma^\omega$.*
3. *A Martin-Löf test $(U_n : n \geq 1)$ is* universal *if for every $x \in \Sigma^\omega$, if $x \notin \bigcap_{n\geq 1} U_n\Sigma^\omega$, then x is Martin-Löf-random.*

Note that when we speak of Martin-Löf tests, we will always be referring to tests which are applied to infinite binary sequences. Such tests are also known as sequential Martin-Löf tests to distinguish them from tests applied to finite strings.

A second measure theoretic criterion was proposed by Solovay.

DEFINITION 1.2 (Solovay [18]).

1. *A* Solovay randomness test *is a uniformly recursively enumerable sequence $(A_n : n \geq 1)$ such that the sum $\sum_{n\geq 1} \mu(A_n\Sigma^\omega)$ is convergent.*
2. *An x in Σ^ω is* Solovay-random *if and only if for every Solovay randomness test $(A_n : n \geq 1)$, $\{n : x \in A_n\Sigma^\omega\}$ is finite.*

It is immediate that every Solovay-random real is Martin-Löf-random, and Solovay proved the converse.

THEOREM 1.3 (Solovay [18]). *Every x in Σ^ω which is Martin-Löf-random is also Solovay-random.*

Characterization by algorithmic complexity.. Our second characterization of effective randomness is based on the hypothesis that an effectively random sequence should be unpredictable.

Suppose that f is a partial recursive function from Σ^* to Σ^*. We say that f is *self-delimiting* if for all a and b in Σ^*, if f is defined on a and on b, then a and b are incompatible; that is to say that they are not equal and neither string extends the other.

DEFINITION 1.4 (Levin [12], Chaitin [4]). *Suppose that f is a self-delimiting recursive function. We write $f(a)\downarrow$ to indicate that f is defined on argument a.*

1. *The* halting probability *of f is $\sum_{f(a)\downarrow} 1/2^{|a|}$.*
2. *If b is in the range of f, then the* f-complexity *of b is the least length of a string a such that $f(a) = b$. If b is not in the range of f, then the f-complexity of b is ∞. Let $H_f(b)$ denote the f-complexity of b.*

Note the halting probability of a self-delimiting function is a real number between 0 and 1. Consequently, we can use its binary expansion to identify it with an element of Σ^ω. This identification is unique for irrational reals.

CONVENTION 1.5. *In the following, we will make implicit use of the identification between $\mathbb{R}$ and Σ^ω whenever we say that a real number has a property defined only on Σ^ω.*

DEFINITION 1.6 (Chaitin [4]). *A recursive function u is* Chaitin-universal *if and only if the following conditions hold:*

1. *u is self-delimiting.*
2. *For any self-delimiting recursive function f, there is a constant C such that for all a, $H_u(a)$ is less than or equal to $H_f(a) + C$.*

PROPOSITION 1.7 (Chaitin [4]). *There is a recursive function which is Chaitin-universal.*

DEFINITION 1.8 (Chaitin [4]). *An $x \in \Sigma^\omega$ is* Chaitin-random *if there is a recursive function u which is Chaitin-universal and a constant C such that for all n, $H_u(x \upharpoonright n) > n - C$. (Here $x \upharpoonright n$ is the sequence given by the first n values of x.)*

It is straightforward to check that every Martin-Löf-random real is Chaitin-random. Schnorr proved the converse; see [3].

THEOREM 1.9 (Schnorr [15]). *For every $x \in \Sigma^\omega$, if x is Chaitin-random, then x is Martin-Löf-random.*

Since all of the preceding notions of effective randomness coincide, except for historical references, we will drop the prefixes and speak of a real's being random.

Natural examples. Chaitin provided a natural class of random reals.

DEFINITION 1.10 (Chaitin [4]). *A* Chaitin Ω-number *is the halting probability of a universal function u as above.*

THEOREM 1.11.

1. (Chaitin [4]). *Every Ω-number is Chaitin-random.*
2. (Solovay [18]). *Every Ω-number is Solovay-random.*

Consequently, every Ω-number is random.

1.2. Recursive enumerability.

DEFINITION 1.12. *An α in Σ^ω is* recursively enumerable *if there is a nondecreasing sequence $(a[n] : n \in \omega)$ from Σ^* such that $\lim_{n\to\infty} a[n] = \alpha$.*

The Ω-numbers provide natural examples of recursively enumerable reals.

Solovay formulated the following notion for recursive increasing sequences of rational numbers converging to real numbers. We take the liberty of presenting his definition in terms of recursive increasing sequences from Σ^* converging to elements of Σ^ω.

DEFINITION 1.13 (Solovay [18]). *Let $(a[n] : n \in \omega)$ and $(b[n] : n \in \omega)$ be recursive monotonically (lexicographically) increasing sequences from Σ^* which converge to α and β, respectively.*

1. *$(a[n] : n \in \omega)$* dominates *$(b[n] : n \in \omega)$ if there is a positive constant C such that for all n in ω, $C(\langle\alpha\rangle - \langle a[n]\rangle) \geq (\langle\beta\rangle - \langle b[n]\rangle)$.*
2. *$(a[n] : n \in \omega)$ is* universal *if it dominates every recursive monotonically increasing sequence from Σ^*.*
3. *α is Ω-like if it is the limit of a universal monotonically increasing recursive sequence from Σ^*.*

Solovay showed that every Ω-number is Ω-like. Additionally, Solovay's proof that every Ω-number is Solovay-random generalizes to Ω-like reals.

THEOREM 1.14 (Solovay [18]). *If α is Ω-like, then α is random.*

Calude et al. [2] sharpened Theorem 1.14 as follows.

THEOREM 1.15 (Calude et al. [2]). *If α is Ω-like, then α is an Ω-number.*

Thus, every Ω-like number is an Ω-number.

Calude et al. [2] posed the natural question, "Is every recursively enumerable random real an Ω-number?" In Theorem 2.1, we show that every recursively enumerable random real is Ω-like and conclude from Theorem 1.15 that the answer to this question is yes.

A second natural class of random reals. Chaitin's Ω-numbers come from universal objects in the complexity theoretic formulation of randomness. Calude et al. [2] raised the question whether the universal objects in the measure theoretic formulation of randomness are also random. They asked, "If $(U_n : n \geq 1)$ is a universal Martin-Löf test, then is $\sum_{n\geq 1} \mu(U_n\Sigma^\omega)$ random?" In Theorem 3.1, we show that the answer is yes. As we discuss below, an equivalent form of this fact was known in the context of recursive analysis.

A dual statement is also true. Theorem 3.3 states that every random recursively enumerable real number is the sum of the measures in a some universal Martin-Löf test.

This paper represents work conducted independently by the authors. Where appropriate, we have indicated places where they came to their solutions to these problems differently.

2. Random recursive enumerability implies Ω-like.

THEOREM 2.1. *Suppose that α is a random recursively enumerable element of Σ^ω. Then α is Ω-like.*

Proof. Let $(a[n] : n \in \mathbb{N})$ be a recursive nondecreasing sequence from Σ^* which converges to α. Let β be recursively enumerable, and let $(b[n] : n \in \mathbb{N})$ be a recursive lexicographically nondecreasing sequence from Σ^* which converges to β.

We show that one of the following two conditions must hold:

1. There is a uniformly recursively enumerable sequence of sets $(A_n : n \in \mathbb{N})$ such that for each n, $A_n \subseteq \Sigma^*$, $\mu(A_n\Sigma^\omega) \leq 1/2^n$, and $\alpha \in A_n\Sigma^\omega$.
2. There is a C such that for all i, $C(\langle\alpha\rangle - \langle a[i]\rangle) \geq (\langle\beta\rangle - \langle b[i]\rangle)$.

Theorem 2.1 follows. If the first condition holds, then α is not random and Theorem 2.1 is verified. Otherwise, the second condition holds and the pair β and $(b[n] : n \in \mathbb{N})$ is not a counterexample to α's being Ω-like. Since β and $(b[n] : n \in \mathbb{N})$ were arbitrary, Theorem 2.1 is verified.

We enumerate A_n by recursion on stages s. Let $A_n[s]$ be the finite set of strings that have been enumerated into A_n during earlier stages than s. Let $s^-[s]$ be the last stage during which we enumerated an element into A_n, or equal to 0, if there was no such earlier stage. If $a[s]$ has an initial segment in $A_n[s]$ or $b[s] = b[s^-[s]]$, then we let $A_n[s+1] = A_n[s]$. Otherwise, let $a[s] + (b[s] - b[s^-[s]])/2^n$ denote the string c such that $\langle c\rangle$ is equal to $\langle a[s]\rangle + (\langle b[s]\rangle - \langle b[s^-[s]]\rangle)/2^n$. We choose a finite antichain $d_1, \ldots, d_k$ from Σ^* such that for every d in $[a[s], a[s] + (b[s] - b[s^-[s]])/2^n]$, there is an i such that d is compatible with d_i . We enumerate $d_1, \ldots, d_k$ into A_n. In other words, we add the interval from $a[s]$ to $a[s] + (b[s] + b[s^-[s]])/2^n$ to $A_n\Sigma^\omega$. Our intention is that if the approximation to β changed by ϵ, then either α will belong to $A_n\Sigma^\omega$ or the approximation to α must change by an additional amount greater than or equal to $\epsilon/2^n$.

First, we calculate that $\mu(A_n\Sigma^\omega) \leq (\langle\beta\rangle - \langle b[0]\rangle)/2^n$: $A_n\Sigma^\omega$ is a union of a disjoint set of intervals, and the measure of $A_n\Sigma^\omega$ is the sum of the lengths of those intervals. That sum has the form

$$(\langle b[t_1]\rangle - \langle b[0]\rangle)/2^n + (\langle b[t_2]\rangle - \langle b[t_1]\rangle)/2^n + (\langle b[t_3]\rangle - \langle b[t_2]\rangle)/2^n + \ldots,$$

where $t_1, t_2, \ldots$ is the sequence of stages during which we enumerate intervals into $A_n\Sigma^\omega$. This is a collapsing sum with limit less than or equal to $(\langle\beta\rangle - \langle b[0]\rangle)/2^n$. The inequality could be strict when there are only finitely many terms in the sum. In any event, $\mu(A_n\Sigma^\omega) \leq 1/2^n$.

If α belongs to each $A_n\Sigma^\omega$, then we have condition 1.

Therefore, suppose that n is fixed so that α is not in $A_n\Sigma^\omega$. By our construction, if we enumerate the interval $[a[s], a[s] + (b[s] - b[s^-[s]])/2^n]$ into $A_n\Sigma^\omega$ during stage s, then there is a stage t greater than s such that $\langle a[t]\rangle$ is greater than $\langle a[s]\rangle + (\langle b[s]\rangle - \langle b[s^-[s]]\rangle)/2^n)$.

We claim that for all s, $2^n(\langle\alpha\rangle - \langle a[s]\rangle) \geq (\langle\beta\rangle - \langle b[s]\rangle)$. Fix s and let t_0 be the greatest stage t less than s such that we enumerate something into A_n during stage t or be 0 if there is no such stage. Let $t_0, t_1, \ldots$ be the sequence of stages, beginning with stage t_0, during which we enumerate intervals into A_n. Then t_1 is greater than or equal to s and $\langle\alpha\rangle - \langle a[t_1]\rangle$ is greater than the sum $\Sigma_{k=1}^{\infty}(\langle b[t_k]\rangle - \langle b[t_{k-1}]\rangle)/2^n$. This is

another collapsing sum and is equal to $(\langle\beta\rangle - \langle b[t_0]\rangle)/2^n$. Consequently, $\langle\alpha\rangle - \langle a[s]\rangle \geq \langle\alpha\rangle - \langle a[t_1]\rangle \geq (\langle\beta\rangle - \langle b[t_0]\rangle)/2^n \geq (\langle\beta\rangle - \langle b[s]\rangle)/2^n$, as required. □

3. Universal Martin-Löf tests have random measure. Subsequent to our having proven Theorem 3.1, the first author observed that a version of it appears in [6], set in the context of recursive analysis. See Remark 3.5.

THEOREM 3.1. *Let $(U_n : n \geq 1)$ be a universal Martin-Löf test. Then, for each $n \geq 1$, $\mu(U_n\Sigma^\omega)$ is random.*

Proof. We show that for each n, $\mu(U_n\Sigma^\omega)$ is Ω-like and therefore random.

Let U be one of the elements of $(U_n : n \geq 1)$. We note that $\mu(U\Sigma^\omega)$ is less than or equal to $1/2$. Let $U[s]$ denote the set consisting of the first s elements in the enumeration of U. Let $\beta \in \Sigma^\omega$ be recursively enumerable, and let $(b[s] : s \geq 1)$ be a recursive increasing sequence from Σ^* which converges to β.

We will construct a Martin-Löf test $(A_n : n \geq 1)$ so that for all n, $A_{n+1}\Sigma^\omega \subseteq A_n\Sigma^\omega$ and so that one of the following conditions holds:

1. For each n, A_n is finite and $\mu(A_n\Sigma^\omega \setminus U\Sigma^\omega) > 0$.
2. There is a C such that for each s, $C(\mu(U\Sigma^\omega) - \mu(U[s]\Sigma^\omega)) > (\langle\beta\rangle - \langle b[s]\rangle)$.

In the first case, we will obtain a contradiction by showing that $(U_n : n \geq 1)$ is not universal. In the second case, we will show that $(\mu(U[s]\Sigma^\omega) : s \geq 1)$ dominates $(b[s] : s \geq 1)$. Since β and $(b[s] : s \geq 1)$ were arbitrary, $\mu(U\Sigma^\omega)$ is Ω-like, as required.

We construct the sets A_n and several auxiliary functions by recursion on stages s. Our continuing convention is to use the suffix $[s]$ to denote the values of these objects during stage s. For example, $A_n[s]$ denotes the finite subset of Σ^* whose elements were enumerated into A_n before stage s.

In our recursion, if the recursion variable i goes to infinity, then we verify the first disjunct above. If i does not go to infinity in the limit, then its limit infimum i^* is the least index for an infinite element of $(A_n : n \geq 1)$. In this case, U must cover a nonzero fraction of the measure of A_{i^*}. We add measure to each A_n so that we can verify the second disjunct above (where C depends on i^*; see below).

We begin the construction with each A_n empty. During stage 0, we define $m_0[0] = 1/2$, define $A_0 = \{\,()\,\}$, the set whose only element is the null sequence, and say that 0 is active during stage 0. During stage s greater than 0, we begin in step 1 and follow the instructions below until reaching one which requires the end of stage s. Upon the end of stage s, we begin stage $s + 1$.

1. Let $m_0[s] = 1/2$, let $A_0[s] = \{\,()\,\}$, and let $i = 1$. Go to step 2.
2. (a) If i has not been active during any previous stage or if all of its previous actions have been canceled, then let $s_i^-[s]$ equal 0.
   (b) Otherwise, let $s_i^-[s]$ be the most recent stage during which i was active.
   Go to step 3.
3. (a) If $s_i^-[s] = 0$ or if $\mu(A_i[s]\Sigma^\omega \setminus U[s]\Sigma^\omega)$ is less than or equal to $d_i[s_i^-[s]]m_{i-1}[s]/2$, then take the following actions.
      i. Set $d_i[s] = (\langle b[s]\rangle - \langle b[s_i^-[s]]\rangle)$ and $m_i[s] = d_i[s]m_{i-1}[s]/2$.

ii. Choose a finite set of strings $F_i[s]$ so that $\mu(F_i[s]\Sigma^\omega)$ is equal to $d_i[s]m_{i-1}[s]$, $F_i[s]\Sigma^\omega$ is a subset of $A_{i-1}[s]\Sigma^\omega$, and $F_i[s]\Sigma^\omega$ is disjoint from $U[s]\Sigma^\omega$. Enumerate the elements of $F_i[s]$ into A_i.

iii. Say that i is active during stage s. For each $j > i$, cancel all of the previous actions for the sake of j.

iv. End the stage s of the recursion.

(b) Otherwise, let $d_i[s] = d_i[s_i^-[s]]$ and $m_i[s] = d_i[s]m_{i-1}[s]/2$. Go to step 4.

4. (a) If i is less than s, then increase the value of i by 1, and go to step 2.

(b) Otherwise, end stage s of the recursion.

Suppose that we reach step 3(a) with $i = n$. If n is equal to 1, then we are required to find a set $F_1[s]$ such that $F_1[s]\Sigma^\omega \subset (\Sigma^\omega \setminus U[s]\Sigma^\omega)$ and $\mu(F_1[s]\Sigma^\omega) = d_1[s]m_0[s]$. Of course, $m_0[s] = 1/2$ and $d_1[s]$ is less than 1. Therefore we must find a set of measure less than $1/2$ in $\Sigma^\omega \setminus U[s]\Sigma^\omega$. Since U belongs to a Martin-Löf test, $\mu(U\Sigma^\omega) \leq 1/2$ and it is possible to find the set $F_1[s]$. If n is greater than 1, then at an earlier point in stage s, we noted that $\mu(A_{n-1}[s]\Sigma^\omega \setminus U[s]\Sigma^\omega)$ is greater than $d_{n-1}[s_{n-1}^-[s]]m_{n-2}[s]/2$. We defined $d_{n-1}[s] = d_{n-1}[s_{n-1}^-[s]]$ and defined $m_{n-1}[s] = d_{n-1}[s]m_{n-2}[s]/2$. Then $d_n[s]m_{n-1}[s] = d_n[s](d_{n-1}[s]m_{n-2}[s]/2)$. Since $d_n[s]$ is less than or equal to 1, this quantity is less than $d_{n-1}[s_{n-1}^-[s]]m_{n-2}[s]/2$, and again it is possible to find the set $F_n[s]$.

We say that n is *injured* during stage s if we cancel all of the previous actions for the sake of n during stage s. Note that 1 is never injured.

Let M_n be the set of stages during which n is active. M_n is naturally divided into intervals by injury to n. If M_n is not empty, then start by letting $\{q_j : j \in Q_n\}$ be an increasing enumeration of the stages s in M_n such that $s_n^-[s]$ is equal to 0. Note that Q_n may be finite or may be all of $\mathbb{N}$. In the case that Q_n is finite with greatest element j, we let q_{j+1} denote infinity and use it to refer to the semi-infinite interval of stages coming after the final injury to n.

To calculate a bound on the measures of the sets $A_n\Sigma^\omega$, we now compute $\sum_{s\in M_n} d_n[s]m_{n-1}[s]$, when n is greater than or equal to 1.

Divide M_n into intervals.

$$\sum_{s\in M_n} d_n[s]m_{n-1}[s] = \sum_{j\in Q_n} \sum_{s\in M_n\cap[q_j,q_{j+1})} d_n[s]m_{n-1}[s].$$

Note that $m_{n-1}[s]$ is constant between q_j and q_{j+1}.

$$= \sum_{j\in Q_n} \left(m_{n-1}[q_j] \sum_{s\in M_n\cap[q_j,q_{j+1})} d_n[s] \right).$$

Identify the collapsing sum.

$$\begin{aligned}
&= \sum_{j \in Q_n} \left(m_{n-1}[q_j] \sum_{s \in M_n \cap [q_j, q_{j+1})} (\langle b[s] \rangle - \langle b[s_n^-[s]] \rangle) \right) \\
&\leq \sum_{j \in Q_n} m_{n-1}[q_j](\langle \beta \rangle - \langle b[0] \rangle) \\
&\leq \sum_{j \in Q_n} m_{n-1}[q_j].
\end{aligned}$$

Note that $m_{n-1}[q_j]$ equal to $m_{n-1}[s]$, where s is the greatest stage less than q_j during which $n-1$ was active.

$$\leq \sum_{s \in M_{n-1}} m_{n-1}[s].$$

The last inequality could be strict, as there may be stages during which $n-1$ is active which are followed by an injury to $n-1$ before the next stage during which n is active.

We now check by induction that $\sum_{s \in M_n} m_n[s]$ is less than or equal to $1/2^{n+1}$.

Consider the case when n is equal to 0. Then, M_0 is equal to $\{0\}$ and $m_0[0]$ is equal to $1/2$. Consequently, $\sum_{s \in M_0} m_0[s] = 1/2$, as required.

Now, suppose that n is greater than 0. Then, $\sum_{s \in M_n} m_n[s]$ is given by the following:

$$\sum_{s \in M_n} m_n[s] = \sum_{s \in M_n} d_n[s] m_{n-1}[s]/2.$$

Move the factor $1/2$ out of the sum, and apply the previous calculation.

$$\leq \frac{1}{2} \sum_{s \in M_{n-1}} m_{n-1}[s].$$

Apply induction.

$$\begin{aligned}
&\leq \frac{1}{2}(1/2^n) \\
&= 1/2^{n+1}.
\end{aligned}$$

We have the required inequality.

Now, $\mu(A_n \Sigma^\omega)$ is less than or equal to the sum of the measures of the sets $F_n[s]\Sigma^\omega$ for $s \in M_n$. Each $F_n[s]\Sigma^\omega$ has measure $d_n[s] m_{n-1}[s]$. Therefore, $\mu(A_n \Sigma^\omega)$ is less than or equal to $\sum_{s \in M_n} d_n[s] m_{n-1}[s]$, which is less than or equal to $\sum_{s \in M_{n-1}} m_{n-1}[s]$, and hence less than or equal to $1/2^n$, as above.

Thus, $(A_n : n \geq 1)$ is a Martin-Löf test.

Suppose that for each n, n is active only finitely often. Then for each n, there is a stage s during which we execute step 3(a) for $i = n$ for the final time. Therefore, for each n, A_n is finite and $A_n \Sigma^\omega \setminus U \Sigma^\omega$ is a closed set of positive measure. Further,

for each n, $A_{n+1}\Sigma^\omega \subseteq A_n\Sigma^\omega$. Since Σ^ω is compact, $\bigcap_{n\geq 1} A_n\Sigma^\omega \setminus U\Sigma^\omega$ is not empty. Thus, $\bigcap_{n\geq 1} A_n\Sigma^\omega$ is not a subset of U, contradicting the universality of $(U_n : n \geq 1)$.

Consequently, there are numbers which are active infinitely often, and we let i^* be the least such number.

The first possibility is that i^* is equal to 1. Consider the action during a stage $s \in M_1$. We add strings to A_1 so that the measure of $A_1\Sigma^\omega \setminus U[s]\Sigma^\omega$ is greater than or equal to $d_1[s]m_0[s]$, where $d_1[s]$ is the amount that the approximation to β has increased since the most recent stage $s_1^-[s]$ during which 1 was active. At the next stage $s_1^+[s]$ in M_1 after s, the measure of $A_1[s_1^+]\Sigma^\omega \setminus U[s]\Sigma^\omega$ is less than $d_1[s]m_0[s]/2 = d_1[s]m_0[0]/2$. Thus, for s in M_1, if the approximation to β increases by $d_1[s]$ during the interval $[s_1^-[s], s)$, then the measure of $U[s_1^+]\Sigma^\omega \setminus U[s]\Sigma^\omega$ is greater than or equal to $d_1[s]m_0[s]/2$. It follows that for every s, $(\langle\beta\rangle - \langle b[s]\rangle) \leq (2/m_0[0])(\mu(U\Sigma^\omega) - \mu(U[s]\Sigma^\omega))$. Thus, every increase in the approximation to β is followed by a proportional increase in the approximation to the measure of U, and so $\mu(U[s]\Sigma^\omega : s \geq 1)$ dominates $(b[s] : s \geq 1)$.

Second, i^* may be larger than 1, but the analysis is completely parallel to that of the previous case. We start from the first stage $s[0]$ in M_{i^*} after i^* is injured for the last time, we add strings to A_{i^*} so that the measure of $A_{i^*}\Sigma^\omega \setminus U[s]\Sigma^\omega$ is greater than or equal to $d_{i^*}[s]m_{i^*-1}[s] = d_{i^*}[s]m_{i^*-1}[s_0]$, and we observe that the measure of $U\Sigma^\omega$ increases by at least half that much during the interval from s to the next stage in M_n. It follows that for every m, $(\langle\beta\rangle - \langle b[m]\rangle) \leq (2/m_{i^*-1}[s_0])(\mu(U\Sigma^\omega) - \mu(U[m]\Sigma^\omega))$.

In either case, U is Ω-like and therefore random. □

The following corollary follows easily.

COROLLARY 3.2. *Let $(U_n : n \geq 1)$ be a universal Martin-Löf test. Then $\sum_{n\geq 1} \mu(U_n\Sigma^\omega)$ is random.*

THEOREM 3.3. *For each recursively enumerable random r in Σ^ω there is a universal Martin-Löf test $(U_n : n \geq 1)$ such that $\langle r\rangle$ is equal to $\sum_{n\geq 1} \mu(U_n\Sigma^\omega)$.*

Proof. We fix a universal Martin-Löf test $(A_n : n \geq 1)$, and construct another $(U_n : n \geq 1)$ based on it so that $\langle r\rangle = \sum_{n\geq 1} \mu(U_n\Sigma^\omega)$. Let $A_n[s]$ denote the finite set of sequences which enter A_n during the first s steps of its enumeration. We may assume that for all n and s, if $s < n$, then $A_n[s]$ is empty. With analogous notation, we will make use of a universal Martin-Löf test $(V_n : n \geq 1)$ and a nondecreasing recursive sequence $(r[s] : s \geq 1)$ with limit r such that for all s, $\sum_{n\geq 1} \mu(V_n\Sigma^\omega) - \sum_{n\geq 1} \mu(V_n[s]\Sigma^\omega)$ is less than $\langle r\rangle - \langle r[s]\rangle$. We first argue that there are such sequences.

For s greater than or equal to 1, let $b[s]$ be the binary string such that the following condition holds:

$$\langle b[s]\rangle = \sum_{s\geq i\geq 1} 2^i \sum_{s\geq j\geq 1} \mu(A_{2i+j+1}[s]\Sigma^\omega).$$

Note that

$$\begin{aligned}
\sum_{s\geq i\geq 1} 2^i \sum_{s\geq j\geq 1} \mu(A_{2i+j+1}[s]\Sigma^\omega) &\leq \sum_{i\geq 1} 2^i \sum_{j\geq 1} \mu(A_{2i+j+1}\Sigma^\omega) \\
&\leq \sum_{i\geq 1} 2^i \sum_{j\geq 1} (1/2^{2i+j+1}) \\
&\leq \sum_{i\geq 1} 1/2^{i+1} \\
&\leq 1/2,
\end{aligned}$$

and so there is such a $b[s]$. Let β be $\lim_{s\to\infty} b[s]$. Since r is random, Theorem 2.1 applies and we may let $(r[s] : s \geq 1)$ be a recursive nondecreasing sequence from Σ^* with limit r and let C be constant such that for all s, $\langle\beta\rangle - \langle b[s]\rangle$ is less than $C(\langle r\rangle - \langle r[s]\rangle)$. Now let k be fixed so that 2^k is greater than C. Then for all s,

$$\begin{aligned} 2^k(\langle r\rangle - \langle r[s]\rangle) &> C(\langle r\rangle - \langle r[s]\rangle) \\ &\geq \langle\beta\rangle - \langle b[s]\rangle \\ &= \sum_{i\geq 1} 2^i \sum_{j\geq 1} \mu(A_{2i+j+1}\Sigma^\omega) - \sum_{s\geq i\geq 1} 2^i \sum_{s\geq j\geq 1} \mu(A_{2i+j+1}[s]\Sigma^\omega) \\ &\geq 2^k \sum_{j\geq 1} \mu(A_{2k+j+1}\Sigma^\omega) - 2^k \sum_{s\geq j\geq 1} \mu(A_{2k+j+1}[s]\Sigma^\omega). \end{aligned}$$

Consequently, for each s,

$$(\langle r\rangle - \langle r[s]\rangle) > \sum_{j\geq 1} \mu(A_{2k+j+1}\Sigma^\omega) - \sum_{s\geq j\geq 1} \mu(A_{2k+j+1}[s]\Sigma^\omega).$$

Then, $(A_{2k+j+1} : j \geq 1)$ is a universal Martin-Löf test such that for all s, $\sum_{j\geq 1} \mu(A_{2k+j+1}\Sigma^\omega) - \sum_{s\geq j\geq 1} \mu(A_{2k+j+1}[s]\Sigma^\omega)$ is less than $\langle r\rangle - \langle r[s]\rangle$.

We first handle the case in which $\langle r\rangle$ is less than $1/2$. Choose m so that

$$\langle r\rangle + \mu(A_{2k+m+1}\Sigma^\omega) < 1/2$$

and so that

$$\langle r\rangle > \sum_{j\geq 1} \mu(A_{2k+m+j+1}\Sigma^\omega).$$

For $n \geq 1$, let $V_n = A_{2k+m+n+1}$. For s greater than or equal to 1, let $v[s]$ be $\sum_{s\geq n\geq 1} \mu(V_n[s]\Sigma^\omega)$, and let v be $\sum_{n\geq 1} \mu(V_n\Sigma^\omega)$. By the estimates given above, for each s, $v - v[s]$ is less than or equal to $\langle r\rangle - \langle r[s]\rangle$.

We now construct our Martin-Löf test $(U_n : n \geq 1)$ so that $V_1 \subseteq U_1$ and for all n greater than 1, $V_n = U_n$.

Assuming that we establish $\sum_{n\geq 1} \mu(U_n\Sigma^\omega) = \langle r\rangle$, then since $\mu(U_1\Sigma^\omega)$ is less than or equal to $\langle r\rangle$ and $\langle r\rangle$ is less than or equal to $1/2$, $(U_n : n \geq 1)$ is a Martin-Löf test. Further, $\cap_{n\geq 1} A_n$ is a subset of $\cap_{n\geq 1} U_n$ and so $(U_n : n \geq 1)$ is universal.

We enumerate U_1 by recursion on stages s. Let $U_1[s]$ be set of strings enumerated into U_1 during stages less than s. Let $u[s]$ be $\mu(U_1[s]\Sigma^\omega) + \sum_{n>1} \mu(V_n[s]\Sigma^\omega)$, and let u be the limit of $u[s]$, as s goes to infinity.

During stage s, if $u[s]$ is less than $\langle r[s]\rangle$, then we enumerate a finite set of strings $F[s]$ into U_1 so that $F[s]\Sigma^\omega \cap U_1[s]\Sigma^\omega = \emptyset$ and $\mu(F[s]\Sigma^\omega)$ is equal to $\langle r[s]\rangle - u[s]$. (Not to ignore a fine point, since $\langle r[s]\rangle$ and $u[s]$ have finite binary expansions, there is such a finite set.) We then enumerate all of the strings that enter V_1 during stage s into U_1.

It remains to check that $u = \sum_{n\geq 1} \mu(U_n\Sigma^\omega)$ is equal to $\langle r\rangle$.

By the construction, for every s, $u[s+1]$ is greater than or equal to $\langle r[s]\rangle$. Consequently, $u \geq \langle r\rangle$.

Since $\sum_{n\geq 1} \mu(V_n\Sigma^\omega) < \langle r\rangle$, there must be a stage s such that $\langle r[s]\rangle \geq v(s)$. At the first such stage, $\langle r[s]\rangle \geq u(s)$ as well. If there are infinitely many stages s during which $\langle r[s]\rangle \geq u[s]$, then $u = \langle r\rangle$, as required. Otherwise, there are only finitely many

such stages. We argue that $\langle r\rangle \geq u$ as follows. Let s_0 be the greatest stage s during which $\langle r[s]\rangle \geq u[s]$. At the beginning of stage s_0+1, we add a finite set of elements $F[s_0+1]$ to U_1 so that the measure of $U_1\Sigma^\omega$ is momentarily equal to $\langle r[s_0]\rangle$. Since $u[s] \geq \langle r[s]\rangle$ during every stage s after s_0, we do not add any further elements to U_1 other than those in V_1. Consequently U_1 is equal to $U_1[s_0] \cup F[s_0+1] \cup V_1$. Further, u, which is equal to $\sum_{n\geq 1} \mu(U_n\Sigma^\omega)$, can be written as

$$u = \mu\left((U_1[s_0] \cup F[s_0+1])\,\Sigma^\omega\right) + \sum_{n>1} \mu(V_n[s_0]\Sigma^\omega) + \mu(V_1\Sigma^\omega \setminus (U_1[s_0] \cup F[s_0+1])\Sigma^\omega) + \sum_{n>1} \mu(V_n\Sigma^\omega \setminus V_n[s_0]\Sigma^\omega).$$

By the choice of $F[s_0+1]$,

$$\mu((U_1[s_0] \cup F[s_0+1])\Sigma^\omega) + \sum_{n>1} \mu(V_n[s_0]\Sigma^\omega) = \langle r[s_0]\rangle.$$

Further, $V_1[s_0] \subseteq U_1[s_0]$ so

$$\begin{aligned}
&\mu(V_1\Sigma^\omega \setminus (U_1[s_0] \cup F[s_0+1])\Sigma^\omega) + \sum_{n>1} \mu(V_n\Sigma^\omega \setminus V_n[s_0]\Sigma^\omega) \\
&\quad\leq \mu(V_1\Sigma^\omega \setminus V_1[s_0]\Sigma^\omega) + \sum_{n>1} \mu(V_n\Sigma^\omega \setminus V_n[s_0]\Sigma^\omega) \\
&\quad\leq \sum_{n\geq 1} \mu(V_n\Sigma^\omega) - \sum_{n\geq 1} \mu(V_n[s_0]\Sigma^\omega) \\
&\quad\leq (v - v[s_0]).
\end{aligned}$$

Then, however, $u \leq \langle r[s_0]\rangle + (v - v[s_0])$. By the above, $(v - v[s_0])$ is less than or equal to $(\langle r\rangle - \langle r[s_0]\rangle)$. We conclude that u is less than or equal to $\langle r[s_0]\rangle + (\langle r\rangle - \langle r[s_0]\rangle)$; that is, $u \leq \langle r\rangle$, as required.

Next we consider the case when $\langle r\rangle$ is greater than $1/2$. Again, let $(A_n : n \geq 1)$ be a universal Martin-Löf test. Choose $m > 1$ so that $1/2 + \sum_{n>m} \mu(A_n\Sigma^\omega)$ is less than $\langle r\rangle$. Let 0^n denote the sequence with n many 0's. For $n \geq 1$, let V_n be a subset of Σ^* such that $A_{m+n}\Sigma^\omega \cup \{0^{n+1}\}\Sigma^\omega$ is equal to $V_n\Sigma^\omega \cup \{0^{n+1}\}\Sigma^\omega$, and each element of V_n is incompatible with 0^{n+1}. For each n, $\mu(A_{m+n}\Sigma^\omega \cup \{0^{n+1}\}\Sigma^\omega)$ is less than or equal to $\mu(A_{m+n}\Sigma^\omega) + 1/2^{n+1}$, which is less than or equal to $1/2^n$. Now we use the method in the previous construction to find $(U_n : n \geq 1)$ such that the following conditions hold: $\sum_{n\geq 1} \mu(U_n\Sigma^\omega) = \langle r\rangle - 1/2$; for each n, $V_n \subseteq U_n$; and every element of U_n is incompatible with 0^{n+1}.

The last constraint is only relevant to the construction of U_1. In the notation of the previous construction, we may be asked during step $s+1$ to find a set of finite sequences $F[s+1]$ such that the measure of $F[s+1]\Sigma^\omega$ is equal to $(\langle r[s]\rangle - 1/2) - u[s]$ and $F[s+1]\Sigma^\omega \cap U_1[s]\Sigma^\omega = \emptyset$. For $n = 1$, the complement of $\{0^{n+1}\}\Sigma^\omega$ has measure $3/4$, so the measure available for the choice of $F[s+1]$ is greater than or equal to $3/4 - u[s]$. Thus it is always possible to find the set $F[s]$ as required.

Finally, we let U_n^* be $U_n \cup \{0^{n+1}\}$.

Then, $\sum_{n\geq 1} \mu(U_n^*\Sigma^\omega)$ is evaluated as follows:

$$\sum_{n\geq 1} \mu(U_n^*\Sigma^\omega) = \sum_{n\geq 1} \mu(U_n\Sigma^\omega \cup \{0^{n+1}\}\Sigma^\omega).$$

Note that $U_n\Sigma^\omega \cap \{0^{n+1}\}\Sigma^\omega$ is empty.

$$\begin{aligned}
&= \sum_{n\geq 1} \mu(U_n\Sigma^\omega) + \sum_{n\geq 1} \mu(\{0^{n+1}\}\Sigma^\omega) \\
&= \sum_{n\geq 1} \mu(U_n\Sigma^\omega) + 1/2 \\
&= (\langle r\rangle - 1/2) + 1/2 \\
&= \langle r\rangle. \qquad \square
\end{aligned}$$

Remark 3.4. Theorem 3.1 can also be proved by using the following idea. We first observe that approximating $\mu(U_n\Sigma^\omega)$ by an open set with measure less than ϵ is at least as difficult as approximating $\langle L\rangle$, where L is the least element in Σ^ω which is not in $U_n\Sigma^\omega$. More precisely, let $\langle\alpha\rangle$ denote $\mu(U_n\Sigma^\omega)$. If we were given a Martin-Löf test $(A_n : n \geq 1)$ such that α belongs to $\bigcap_{n\geq 1} A_n\Sigma^\omega$, then we could construct another Martin-Löf test $(B_n : n \geq 1)$ such that L belongs to $\bigcap_{n\geq 1} B_n\Sigma^\omega$. By virtue of L's passing the universal Martin-Löf test $(U_n : n \geq 1)$, L is random, a contradiction. Theorem 3.1 follows.

We sketch the enumeration of $(B_n : n \geq 1)$. For any σ and s, if s is the least such that

$$\sigma \in A_n[s] \text{ and } \langle\sigma\rangle \leq \mu(U_n[s]\Sigma^\omega) < \langle\sigma\rangle + 2^{-|\sigma|},$$

take the following actions. Choose a finite set of strings $G_n[s]$ such that $G_n[s]\Sigma^\omega$ is disjoint from $U_n[s]\Sigma^\omega$, $\mu(G_n[s]\Sigma^\omega) = \langle\sigma\rangle + 2^{-|\sigma|} - \mu(U_n[s]\Sigma^\omega)$; if β is the least element in Σ^ω which is not in $U_n[s]\Sigma^\omega \cup G_n[s]\Sigma^\omega$, then

$$\langle\beta\rangle = \mu(U_n[s]\Sigma^\omega \cap \{\gamma : \gamma < \beta\}) + \mu(G_n[s]\Sigma^\omega).$$

(Observe that such set $G_n[s]$ exists.) Enumerate $G_n[s]$ into B_n.

Roughly speaking, enumerate into B_n a finite set of strings $G_n[s]$ such that $G_n[s]\Sigma^\omega$ presents the leftmost part of the complement of $U_n[s]\Sigma^\omega$ (not necessarily connected) of a total length $\langle\sigma\rangle + 2^{-|\sigma|} - \mu(U_n[s]\Sigma^\omega)$.

Remark 3.5. As we mentioned above, a recursive-analysis version of Theorem 3.1 was proven by [6]. Demuth worked in the Markov/Russian style of constructive mathematical analysis. He studied a behavior of everywhere defined constructive functions of a real variable and, among others, questions of differentiation of such functions. Since random reals from the closed unit interval

1. form a set of measure one,
2. arise by avoiding sets of measure zero from a special class, and
3. can be viewed as "generic,"

one could expect that they would be important in recursive analysis. This is the case and Demuth devoted a considerable amount of effort toward understanding their role there. He started with a rather finitistic approach, and he used a more standard terminology only in his last papers. We briefly survey Demuth's work here, taking the liberty to reformulate his definitions and results into a contemporary terminology.

Demuth [6] studied reals recursive in $\emptyset'$ and defined π_1 and π_2 numbers in that context. According to Demuth, a real x recursive in $\emptyset'$ is a π_1 number if and only if for some (equivalently, for any) recursive sequence of rational numbers $(a[n] : n \in \omega)$ converging to x there is a recursive sequence of finite recursive sets $(C_m : m \in \omega)$

such that $\mu(\bigcup_{s \notin C_m} (\min(a[s], a[s+1]), \max(a[s], a[s+1])))$ is less than 2^{-m}. Dually, x recursive in $\emptyset'$ is a π_2-number if x is not a π_1-number.

In his own terminology, Demuth constructed a universal Martin-Löf test [6, Theorem 2] and showed [6, Theorem 5] that for all x recursive in $\emptyset'$, x is a π_2 number if and only if x is random in the sense of Martin-Löf. We are omitting some details here. Later, Demuth [7] worked with arithmetical reals and defined $\mathcal{A}_1$ and $\mathcal{A}_2$ numbers as natural extensions of π_1 and π_2 numbers. Demuth was not aware that Martin-Löf had formulated these notions earlier.

Finally, Demuth [9] extended these notions to all reals under a different terminology (still not using "randomness"). In [6], he proved, among other things, the following.

(Demuth [6, Lemma 3]). If $r = \sum_{n \in \omega} r_n$, for nonnegative rationals r_n, is a π_1-number (i.e., nonrandom), then $\sum_{n \in C} r_n$ for any recursively enumerable set C is again a π_1-number (no proof was given).

(Demuth [6, Corollary]). Let Q be a recursively enumerable set of strings. If $\mu(Q\Sigma^\omega)$ is a π_1-number and $\mu(Q[s]\Sigma^\omega)$ is less than 1 for all s, then there is a π_1-number x with $0 \leq x \leq 1$ such that $x \notin Q\Sigma^\omega$ (no proof was given).

In other words, if $\mu(Q\Sigma^\omega)$ is not random, then there is a nonrandom real not in $Q\Sigma^\omega$. It follows that if U_n appears as one of the sets in a universal Martin-Löf test, then $\mu(U_n\Sigma^\omega)$ is random.

For more on the massive work of Demuth on recursive analysis one could also consult [10], [8], or [9]. Finally, we note that Demuth also proved several interesting results from a more recursion theoretic point of view in his last papers; see [8], [9], [10]. He also studied various modifications of randomness, again motivated by problems arising in recursive analysis.

Acknowledgments. Slaman wishes to thank Cristian S. Calude and Robert M. Solovay for their advice on this project.

REFERENCES

[1] C. Calude, *Information and Randomness. An Algorithmic Perspective*, Springer-Verlag, Berlin, 1994.

[2] C. S. Calude, P. H. Hertling, B. Khoussainov, and Y. Wang, *Recursively enumerable reals and Chaitin Ω numbers*, in Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS 98), Springer, Berlin, 1998, pp. 596–606.

[3] G. J. Chaitin, *A theory of program size formally identical to information theory*, J. Assoc. Comput. Mach., 22 (1975), pp. 329–340.

[4] G. J. Chaitin, *Algorithmic information theory*, IBM J. Res. Develop., 21 (1977), pp. 350–359, 496. Reprinted in [5].

[5] G. J. Chaitin, *Information, Randomness, and Incompleteness. Papers on Algorithmic Information Theory*, 2nd ed., World Scientific, River Edge, NJ, 1990.

[6] O. Demuth, *On constructive pseudonumbers*, Comment. Math. Univ. Carolin., 16 (1975), pp. 315–331 (in Russian).

[7] O. Demuth, *On some classes of arithmetical real numbers*, Comment. Math. Univ. Carolin., 23 (1982), pp. 453–465 (in Russian).

[8] O. Demuth, *Reducibilities of sets based on constructive functions of a real variable*, Comment. Math. Univ. Carolin., 29 (1988), pp. 143–156.

[9] O. Demuth, *Remarks on the structure of tt-degrees based on constructive measure theory*, Comment. Math. Univ. Carolin., 29 (1988), pp. 233–247.

[10] O. Demuth, *Remarks on Denjoy sets*, in Proceedings of the Summer School Conference Dedication of the 90th Anniversary of Heyting, Plenum Press, New York, 1990, pp. 267–280.

[11] A. N. Kolmogorov, *Three approaches to the definition of the concept "quantity of information,"* Problemy Peredači Informacii, 1 (1965), pp. 3–11.

[12] L. A. Levin, *Laws of information conservation (non-growth) and aspects of the foundation of probability theory*, Problems Inform. Transmission, 10 (1974), pp. 206–210.
[13] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, New York, 1997.
[14] P. Martin-Löf, *The definition of random sequences*, Information and Control, 9 (1966), pp. 602–619.
[15] C.-P. Schnorr, *Process complexity and effective random tests*, J. Comput. System Sci., 7 (1973), pp. 376–388.
[16] R. J. Solomonoff, *A formal theory of inductive inference.* I, Information and Control, 7 (1964), pp. 1–22.
[17] R. J. Solomonoff, *A formal theory of inductive inference.* II, Information and Control, 7 (1964), pp. 224–254.
[18] R. M. Solovay, *Draft of a Paper (or Series of Papers) on Chaitin's Work ...* , manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1974, p. 215.
[19] R. M. Solovay, *private communication,* 1999.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 212–232

TREEWIDTH AND MINIMUM FILL-IN: GROUPING THE MINIMAL SEPARATORS*

VINCENT BOUCHITTÉ† AND IOAN TODINCA†

Abstract. We use the notion of potential maximal clique to characterize the maximal cliques appearing in minimal triangulations of a graph. We show that if these objects can be listed in polynomial time for a class of graphs, the treewidth and the minimum fill-in are polynomially tractable for these graphs. We prove that for all classes of graphs for which polynomial algorithms computing the treewidth and the minimum fill-in exist, we can list their potential maximal cliques in polynomial time. Our approach unifies these algorithms. Finally we show how to compute in polynomial time the potential maximal cliques of weakly triangulated graphs for which the treewidth and the minimum fill-in problems were open.

Key words. graph algorithms, treewidth, minimum fill-in, weakly triangulated graphs

AMS subject classifications. 05C85, 68Q20, 68R10

PII. S0097539799359683

1. Introduction.

The notion of *treewidth* was introduced by Robertson and Seymour in [28]. It plays a major role in graph algorithm design. Indeed, it has been shown that many classical NP-hard problems become polynomial and even linear when restricted to graphs with small treewidth. These algorithms often use a tree decomposition or a *triangulation* of the input graph, which is a chordal supergraph, i.e., all the cycles with at least four vertices of the supergraph have a chord. Computing the treewidth consists of finding a triangulation of minimum cliquesize. A related problem is the *minimum fill-in* problem, which consists of finding a triangulation of a graph such that the number of added edges is minimum. This parameter is used in sparse matrix factorization.

Both the treewidth and the minimum fill-in problems are NP-complete. Nevertheless, these parameters can be computed in polynomial time for several classes of graphs such as chordal bipartite graphs [20, 9], circle and circular-arc graphs [16, 32, 24], and AT-free graphs with a polynomial number of separators [23]. Most of these algorithms use the fact that these classes of graphs have a polynomial number of *minimal separators*. It was conjectured in [18, 19] that the treewidth and the minimum fill-in should be tractable in polynomial time for all the graphs having a polynomial number of minimal separators. The conjecture is still open.

A potential maximal clique of a graph is a set of vertices which induces a maximal clique in some minimal triangulation of the graph. Although this seems a purely combinatorial definition, a potential maximal clique corresponds to a local grouping of some minimal separators of the graph. This will lead to a local characterization of a potential maximal clique, and in particular to a polynomial algorithm that, given a graph G and a set of vertices K, decides if K is a potential maximal clique of G. We also show in this paper that if one can list in polynomial time all the potential

*Received by the editors August 2, 1999; accepted for publication (in revised form) November 8, 2000; published electronically July 25, 2001. Preliminary versions of this paper were presented at the European Symposium on Algorithms, Venice, Italy, 1998 and at the Symposium on Theoretical Aspects of Computer Science, Trier, Germany, 1999.
http://www.siam.org/journals/sicomp/31-1/35968.html

†LIP-École Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France (Vincent.Bouchitte@ens-lyon.fr, Ioan.Todinca@ens-lyon.fr).

maximal cliques of some class of graphs, then the treewidth and the minimum fill-in of those graphs can be computed in polynomial time.

We prove that the potential maximal cliques can be enumerated in polynomial time for all the classes of graphs previously mentioned, unifying in this way the cited algorithms.

The class of *weakly triangulated graphs*, introduced in [13], is a class of graphs with a polynomial number of separators, probably the only one for which the treewidth and minimum fill-in problems were still open. We give an algorithm computing the potential maximal cliques of these graphs. Consequently, the treewidth and the minimum fill-in of weakly triangulated graphs are computable in polynomial time.

2. Chordal graphs and minimal separators. Throughout this paper we consider connected, simple, finite, undirected graphs.

A graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e., an edge between two nonconsecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. H is a *minimal triangulation* if for any intermediate set E'' with $E \subseteq E'' \subset E'$, the graph (V, E'') is not triangulated. For example, the graph of Figure 2.1(b) is a minimal triangulation of the graph of Figure 2.1(a).

Another characterization of minimal triangulations is provided in [29].

LEMMA 2.1. *Let H be a triangulation of a graph G. Then H is a minimal triangulation of G if and only if, for any edge e of $E(H) - E(G)$, the graph $H - e$ is not triangulated.*

Now let us define the treewidth and the minimum fill-in of a graph.

DEFINITION 2.2. *Let G be a graph. The* treewidth *of G, denoted by* $\mathrm{tw}(G)$*, is the minimum, over all triangulations H of G, of $\omega(H) - 1$, where $\omega(H)$ is the maximum cliquesize of H.*

DEFINITION 2.3. *The* minimum fill-in *of a graph G, denoted by* $\mathrm{mfi}(G)$*, is the smallest value of $|E(H) - E(G)|$, where the minimum is taken over all triangulations H of G.*

In other words, computing the treewidth of G means finding a triangulation with the smallest cliquesize, while computing the minimum fill-in consists in finding a triangulation with the smallest number of edges. In both cases we can restrict our work to minimal triangulations.

Now let a and b be two nonadjacent vertices of a graph G. A set of vertices $S \subseteq V$ is an *a, b-separator* if the removal of S from the graph separates a and b in different connected components. S is a *minimal a, b-separator* if no proper subset of S separates a and b. We say that S is a *minimal separator* of G if there are two vertices a and b such that S is a minimal a, b-separator. Notice that a minimal separator can be strictly included in another one. We denote by Δ_G the set of all minimal separators of G.

Let us recall some known results about the minimal separators of a chordal graph. We will use the representation of chordal graphs provided by *clique trees*. For an extensive survey of these notions, see [2, 12]. A *clique* is a complete subgraph of G. Now consider the set $\mathcal{K}_G = \{\Omega_1, \ldots, \Omega_p\}$ of maximal cliques of G. Let $\mathcal{T}$ be a tree on $\mathcal{K}_G$, i.e., every maximal clique $\Omega \in \mathcal{K}_G$ corresponds to exactly one node of $\mathcal{T}$. We also say that the nodes of $\mathcal{T}$ are *labeled* by the cliques of $\mathcal{K}_G$ and we will simply denote by Ω the node of the tree labeled by a maximal clique Ω. We say that $\mathcal{T}$ is a *clique tree* of G if it satisfies the *clique-intersection property*: for every pair of distinct cliques $\Omega, \Omega' \in \mathcal{K}$, the set $\Omega \cap \Omega'$ is contained in every clique on the unique path connecting Ω

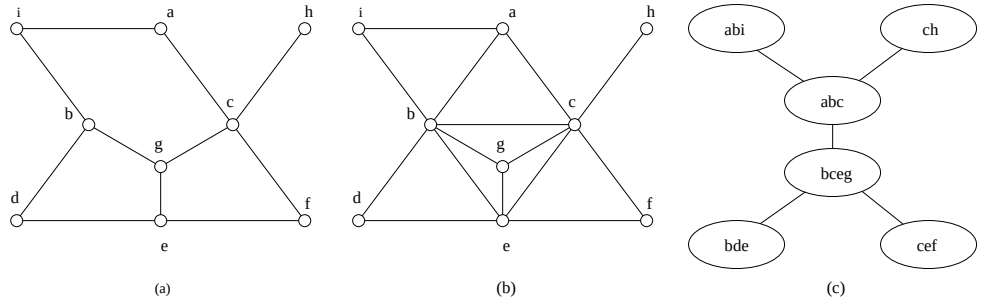


FIG. 2.1. *Minimal triangulation and clique tree.*

and Ω' in the tree $\mathcal{T}$. It is well known (see [2] for a proof) that a graph G is chordal if and only if it has a clique tree.

Figure 2.1(c) presents a clique tree of the chordal graph of Figure 2.1(b).

The following crucial property is proved in [2].

PROPOSITION 2.4. *Let H be a chordal graph and $\mathcal{T}$ be any clique tree of H. A set of vertices S is a minimal separator of H if and only if $S = \Omega \cap \Omega'$ for some maximal cliques Ω and Ω' of H adjacent in the clique tree $\mathcal{T}$.*

In particular, all minimal separators of a chordal graph are cliques. Actually, Dirac [10] has shown that a graph is chordal if and only if all its minimal separators are cliques.

The following proposition (see [2]) gives another relation between a minimal separator and a clique tree of H.

PROPOSITION 2.5. *Let $\mathcal{T}$ be any clique tree of a chordal graph H and let Ω, Ω' be two maximal cliques of H, adjacent in $\mathcal{T}$. Consider the two subtrees of $\mathcal{T}$ obtained by removing the edge between the nodes Ω and Ω'. Let $\mathcal{T}_\Omega$ be the subtree containing Ω and $\mathcal{T}_{\Omega'}$ the subtree containing Ω'. We denote by V_Ω and $V_{\Omega'}$ the union of the labels of $\mathcal{T}_\Omega$ (respectively, $\mathcal{T}_{\Omega'}$). Then the minimal separator $S = \Omega \cap \Omega'$ separates in H every vertex of $V_\Omega - S$ from every vertex of $V_{\Omega'} - S$.*

Consider, for example, the chordal graph H of Figure 2.1(b) and its clique tree of Figure 2.1(c). If we take the adjacent cliques $\{a, b, c\}$ and $\{b, c, e, g\}$, then the minimal separator $\{b, c\}$ separates in H every vertex of $\{a, h, i\}$ from every vertex of $\{d, e, f, g\}$.

Let G be a graph and S a minimal separator of G. We note $\mathcal{C}_G(S)$ to be the set of connected components of $G - S$. A component $C \in \mathcal{C}_G(S)$ is a *full* component associated with S if every vertex of S is adjacent to some vertex of C. We denote by $\mathcal{C}^*_G(S)$ the set of all full components associated with S. For the following lemma, we refer to [12].

LEMMA 2.6. *A set S of vertices of G is a minimal a, b-separator if and only if a and b are in different full components associated with S.*

If $C \in \mathcal{C}(S)$, we say that $(S, C) = S \cup C$ is a *block* associated with S. A block (S, C) is called *full* if C is a full component associated with S.

DEFINITION 2.7. *Two separators S and T* cross*, denoted by $S \sharp T$, if T intersects at least two distinct components of $G - S$. If S and T do not cross, they are called* parallel*, denoted by $S \| T$.*

It is easy to prove that these relations are symmetric (see [26]). Remark that two minimal separators S and T are parallel if and only if T is contained in some block (S, C) associated with S.

Using the fact that a separator cannot separate two adjacent vertices we deduce the following lemma.

LEMMA 2.8. *Let G be a graph, S a minimal separator, and Ω a clique of G. Then Ω is included in some block associated with S. In particular, the minimal separators of a chordal graph are pairwise parallel.*

Let $S \in \Delta_G$ be a minimal separator. We denote by G_S the graph obtained from G by completing S, i.e., by adding an edge between every pair of nonadjacent vertices of S. If $\Gamma \subseteq \Delta_G$ is a set of separators of G, G_Γ is the graph obtained by completing all the separators of Γ. The results of [22], concluded in [27], establish a strong relation between the minimal triangulations of a graph and its minimal separators.

THEOREM 2.9. *Let $\Gamma \subseteq \Delta_G$ be a maximal set of pairwise parallel separators of G. Then $H = G_\Gamma$ is a minimal triangulation of G and $\Delta_H = \Gamma$.*

Conversely, let H be a minimal triangulation of a graph G. Then Δ_H is a maximal set of pairwise parallel separators of G and $H = G_{\Delta_H}$.

In other terms, every minimal triangulation of a graph G is obtained by considering a maximal set Γ of pairwise parallel separators of G and completing the separators of Γ. The minimal separators of the triangulation are exactly the elements of Γ. For example, if G is the graph of Figure 2.1(a) and H is the graph of Figure 2.1(b), then $\Gamma = \{\{a,b\},\{b,c\},\{c\},\{b,e\},\{c,e\}\}$.

It is important to know that the elements of Γ, which become the minimal separators of H, have strictly the same behavior in H as in G. Indeed, the connected components of $H - S$ are exactly the same in $G - S$ for every $S \in \Gamma$. Moreover, the full components are the same in the two graph: $\mathcal{C}^*_H(S) = \mathcal{C}^*_G(S)$.

3. Potential maximal cliques. The previous theorem gives a characterization of the minimal triangulations of a graph by means of minimal separators, but it gives no algorithmic information about how we should construct a minimal triangulation in order to minimize its cliquesize or the fill-in. We will prove that the *potential maximal cliques* of a graph suffice to compute its treewidth and its minimum fill-in.

DEFINITION 3.1. *A set of vertices Ω of a graph G is called a* potential maximal clique *if there is a minimal triangulation H of G such that Ω is a maximal clique of H.*

In this section, we give several characterizations of the potential maximal cliques of a graph, which will allow us to recognize a potential maximal clique in polynomial time and also to enumerate these objects for several classes of graphs.

3.1. Potential maximal cliques and minimal separators. If K is a set of vertices of G, we denote by $\Delta_G(K)$ the minimal separators of G included in K. Our aim is to give a strong relation between a potential maximal clique Ω and the minimal separators of $\Delta_G(\Omega)$.

DEFINITION 3.2. *Let G be a graph and $\mathcal{S} \subseteq \Delta_G$ a set of pairwise parallel separators such that for any $S \in \mathcal{S}$, there is a block $(S, C(S))$ containing all the separators of $\mathcal{S}$. Suppose that $\mathcal{S}$ ordered by inclusion has no greatest element. We define the* piece between *the elements of $\mathcal{S}$ by*

$$P(\mathcal{S}) = \bigcap_{S \in \mathcal{S}} (S, C(S)).$$

Notice that for any $S \in \mathcal{S}$ the block $(S, C(S))$ containing all the separators of $\mathcal{S}$ is unique: if $T \in \mathcal{S}$ is not included in S, there is a unique connected component of S containing $T - S$.

LEMMA 3.3. *Let H be a chordal graph and let Ω be a maximal clique of H. Then either $\Delta_H(\Omega)$ has a greatest element, or $P(\Delta_H(\Omega))$ exists and contains Ω.*

Proof. According to Lemma 2.8, for any minimal separator $S \in \Delta_H(\Omega)$, the clique Ω is contained in some block $(S, C(S))$ of S. It follows that if $\Delta_H(\Omega)$ has no greatest element, then Ω is contained in $P(\Delta_H(\Omega))$, by definition of the piece between. □

THEOREM 3.4. *Let Ω be a maximal clique of a chordal graph H. If all the separators of $\Delta_H(\Omega)$ are contained in some $S \in \Delta_H(\Omega)$, then Ω is a block (S, C) associated with S. Otherwise, $\Omega = P(\Delta_H(\Omega))$.*

Proof. Suppose that $\bigcup_{T \in \Delta_H(\Omega)} T = S$ with $S \in \Delta_H(\Omega)$. Ω is included in some block (S, C) of S. If $x \in C$ is not in Ω, we take in a clique tree of H a path $\Omega, \Omega', \ldots, \Omega^{(i)}$ of adjacent cliques such that $x \in \Omega^{(i)}$. Then $T = \Omega \cap \Omega'$ is a minimal separator of H included in Ω, so it must be an element of $\Delta_H(\Omega)$. In particular, T is included in S. According to Proposition 2.5, T separates x and any vertex of $\Omega - T$. Since $T \subseteq S$, S clearly separates x and any vertex of $\Omega - S$, contradicting the fact that x and Ω are in the same block associated with S. It follows that $(S, C) = \Omega$.

Now suppose that no separator $S \in \Delta_H(\Omega)$ contains all the others. According to the Lemma 3.3, $P(\Delta_H(\Omega))$ exists. On the one hand, $\Omega \subseteq P(\Delta_H(\Omega))$ by Lemma 3.3. On the other hand, consider $y \in P(\Delta_H(\Omega)) - \Omega$. In a clique tree of H, we consider a path $\Omega, \Omega', \ldots, \Omega^{(i)}$ of adjacent cliques such that $y \in \Omega^{(i)}$. Then $S = \Omega \cap \Omega'$ is a minimal separator that belongs to $\Delta_H(\Omega)$ and S separates y from every vertex of $\Omega - S$ by Proposition 2.5. Let T be a separator of $\Delta_H(\Omega)$ not included in S. Then S separates y and a vertex of $T - S$, so y is not in the block $(S, C(S))$ containing Ω, contradicting our choice. It follows that $\Omega = P(\Delta_H(\Omega))$. □

Theorem 3.4 gives a relation between a maximal clique of a chordal graph and the minimal separators contained in the clique. We extend this result to the potential maximal cliques of a graph. We also establish the reverse of Theorem 3.4, which will allow us to recognize a potential maximal clique. For this we need some easy lemmas.

LEMMA 3.5. *Let H be a minimal triangulation of a graph G and T be a minimal separator of G such that $H[T]$ is a clique. Then T is also a minimal separator of H.*

Proof. We know that $H = G_\Gamma$ for a maximal set of pairwise parallel separators $\Gamma \subseteq \Delta_G$ and $\Delta_H = \Gamma$ (Theorem 2.9). Let $S \in \Gamma$ be any minimal separator of H and G. By Lemma 2.8, the clique T will be in some block (S, C) of H. Since the blocks associated with S in G are the same as in H (Theorem 2.9 and the related remarks) we deduce that S and T are parallel in G. Since Γ is a *maximal* set of pairwise parallel separators, T must be in $\Gamma = \Delta_H$. □

COROLLARY 3.6. *In particular, if T is a minimal separator of G contained in S with $S \in \Delta_H$, then $T \in \Delta_H$.*

Consider a graph G and a minimal triangulation H of G. An important consequence of Lemma 3.5 is that if Ω is a clique of H, then the minimal separators of H contained in Ω are the same as in G.

PROPOSITION 3.7. *Let G be a graph and H be a minimal triangulation of G. Then for any clique Ω of H, we have $\Delta_H(\Omega) = \Delta_G(\Omega)$.*

Proof. By Theorem 2.9, every minimal separator of H is also a minimal separator of G, so $\Delta_H(\Omega) \subseteq \Delta_G(\Omega)$. According to Lemma 3.5, since every minimal separator $S \in \Delta_G(\Omega)$ of S induces a clique in H, S is also a minimal separator of H. □

The next step is to prove that if Ω is a maximal clique of a minimal triangulation H of G, then Ω is also a clique in the graph $G_{\Delta_G(\Omega)}$, obtained by completing in G the minimal separators included in Ω.

LEMMA 3.8. *Let H be a chordal graph, Ω a maximal clique, and S a minimal*

separator of H intersecting Ω. Then $S \cap \Omega$ is contained in some minimal separator T with $T \subset \Omega$.

Proof. Consider a clique tree $\mathcal{T}$ of H. We can write $S = \Omega_1 \cap \Omega_2$ for some maximal cliques Ω_1, Ω_2 of H, adjacent in $\mathcal{T}$ (Proposition 2.4). Let us suppose that Ω_1 is closer to Ω in the clique tree than Ω_2. Let Ω' be the clique next to Ω on the path from Ω to Ω_2 in $\mathcal{T}$, i.e., the path is $\Omega, \Omega', \ldots, \Omega_1, \Omega_2$.

Then $\Omega_1 \cap \Omega \subset \Omega'$ and $\Omega_2 \cap \Omega \subset \Omega'$ by definition of a clique tree. Since $S = \Omega_1 \cap \Omega_2$, we deduce that $S \cap \Omega \subset \Omega'$ and, in particular, $S \cap \Omega$ is contained in $\Omega \cap \Omega'$, which is a minimal separator by Proposition 2.4. □

PROPOSITION 3.9. *Let Ω be a potential maximal clique of G. Then Ω is a clique in the graph $G_{\Delta_G(\Omega)}$.*

Proof. Let H be a minimal triangulation of G such that Ω is a maximal clique of H. Notice that H exists by definition of a potential maximal clique. By Theorem 2.9, $H = G_{\Delta_H}$, so for any edge $\{x, y\} \in E(H) - E(G)$, there is a minimal separator of H containing both x and y. Now let $\{x, y\}$ be an edge of $H[\Omega]$; we want to prove that $\{x, y\}$ is also an edge of $G_{\Delta_G(\Omega)}[\Omega]$. If $\{x, y\} \in E(G)$, the assertion is clearly true. Otherwise, let S be a minimal separator of H such that $x, y \in S$. By Lemma 3.8, there is a minimal separator T of H contained in Ω and containing x and y. We deduce that $\{x, y\}$ is also an edge of $G_{\Delta_G(\Omega)}$. □

We can give now a characterization of the potential maximal cliques Ω of a graph using the minimal separators of $\Delta_G(\Omega)$.

THEOREM 3.10. *Let Ω be a set of vertices of G and suppose that $\Delta_G(\Omega)$ has a maximum element S, i.e., every T in $\Delta_G(\Omega)$ is included in S. Then Ω is a potential maximal clique if and only if Ω is some block (S, C) and $G_{\Delta_G(\Omega)}[\Omega]$ is a clique.*

Proof. "⇒." Let H be a minimal triangulation of G such that Ω is a maximal clique of H. By Proposition 3.7, $\Delta_H(\Omega) = \Delta_G(\Omega)$, so S is an element of $\Delta_H(\Omega)$, maximum by inclusion. According to Theorem 3.4, Ω is a block (S, C) of S in H. Since the blocks of S are the same in H and in G, (S, C) is also a block in the graph G.

By Proposition 3.9, Ω is a clique in the graph $G_{\Delta_G(\Omega)}$. Notice that $G_{\Delta_G(\Omega)}$ is identical to G_S.

"⇐." Notice that the separators of $\Delta_G(\Omega)$ are pairwise parallel. Indeed, let H be a minimal triangulation of G such that $S \in \Delta_H$. Then each $T \in \Delta_G(\Omega)$ is a clique in H because $T \subseteq S$. Consequently $\Delta_G(\Omega) \subseteq \Delta_H$, so by Theorem 2.9, the elements of $\Delta_G(\Omega)$ are pairwise parallel in G.

We prove that $S \cup C$ is a maximal clique of H. Let Ω' be a clique of H including $S \cup C$; Ω' must be in some block associated with S in H (Lemma 2.8), and the only choice is $\Omega \subseteq (S, C)$. We conclude that Ω is a maximal clique of the minimal triangulation H of G, and therefore Ω is a potential maximal clique of G. □

THEOREM 3.11. *Let Ω be a set of vertices of G and suppose that $\Delta_G(\Omega)$ ordered by inclusion has no greatest element. Then Ω is a potential maximal clique if and only if $\Omega = P(\Delta_G(\Omega))$ and $G_{\Delta_G(\Omega)}[\Omega]$ is a clique.*

Proof. The proof is very similar to the one of the previous theorem.

"⇒." We consider a minimal triangulation H of G such that Ω is a maximal clique of H. According to Theorem 3.4, $\Omega = P(\Delta_H(\Omega))$ in H. By Proposition 3.7, $\Delta_H(\Omega) = \Delta_G(\Omega)$ and the minimal separators of $\Delta_H(\Omega)$ induce the same connected components in G and in H. Consequently, $P(\Delta_H(\Omega))$ is the same as $P(\Delta_G(\Omega))$ in G, so $\Omega = P(\Delta_G(\Omega))$ in the graph G.

By Proposition 3.9, Ω is a clique in $G_{\Delta_G(\Omega)}$.

"⇐." Since $P(\Delta_G(\Omega))$ exists, the minimal separators of $\Delta_G(\Omega)$ are pairwise parallel in G. As in the previous theorem, we prove that Ω is a maximal clique in any minimal triangulation $H = G_\Gamma$ of G with $\Delta_G(\Omega) \subseteq \Gamma$.

Since Ω is a clique in $G_{\Delta_G(\Omega)}$, Ω is also a clique in H. Consider a maximal clique Ω' of H containing Ω. Let x be a vertex of Ω'. For any $S \in \Delta_H(\Omega)$, x must be in the block $(S, C_\Omega(S))$ of H containing Ω. Therefore $x \in P(\Delta_H(\Omega))$ in the graph H. However, $\Delta_H(\Omega) = \Delta_G(\Omega)$ by Proposition 3.7, and the piece between these separators is the same in H and in G. It follows that $x \in P(\Delta_G(\Omega)) = \Omega$, so Ω is a maximal clique of H. □

Notice that Theorems 3.10 and 3.11 lead to an algorithm that, given a graph G, its minimal separators Δ_G, and a set of vertices K, verifies in polynomial time if K is a potential maximal clique of G.

3.2. A simpler characterization of potential maximal cliques. We are going to give a strong characterization of potential maximal cliques here, which does not use minimal separators. We start with some easy observations, following directly from the previous results.

COROLLARY 3.12. *Let Ω be a potential maximal clique of G and let $S \in \Delta_G(\Omega)$. Then S is strictly contained in Ω and $\Omega - S$ is in a full connected component associated with S.*

Proof. Consider a minimal triangulation H of G such that Ω is a maximal clique of H. By Lemma 3.5, S is a minimal separator of H. S is an intersection of two distinct maximal cliques of H by Proposition 2.4, so S cannot be a maximal clique of H. It follows that S is strictly contained in Ω.

Now let us prove that $\Omega - S$ is in a full connected component associated with S in G. Clearly $\Omega - S$ is in a full component associated with S in H. Since the full components associated with S in G are the same as in H, we conclude that $\Omega - S$ is in a full component associated with S in G. □

COROLLARY 3.13. *Let Ω be a potential maximal clique of a graph G and let a be any vertex of $V - \Omega$. There is a minimal separator $S \subset \Omega$ that separates a and $\Omega - S$.*

Proof. If we are in the case of Theorem 3.10, $\Delta_G(\Omega)$ has a greatest element S and $\Omega = (S, C)$. Then clearly S separates any vertex $a \in V - (S, C)$ from any vertex of $C = \Omega - S$. Now suppose that we are under the conditions of Theorem 3.11, so $\Delta_G(\Omega)$ has no greatest element and $\Omega = P(\Delta_G(\Omega))$. By definition of $P(\Delta_G(\Omega))$, since $a \notin P(\Delta_G(\Omega))$, we deduce that there is some $S \in \Delta_G(\Omega)$ separating a from $P(\Delta_G(\Omega)) - S$. □

Now let K be a set of vertices of a graph G. We denote by $C_1(K), \ldots, C_p(K)$ the connected components of $G - K$. We denote by $S_i(K)$ the vertices of K adjacent to at least one vertex of $C_i(K)$. When no confusion is possible we will simply speak of C_i and S_i. If $S_i(K) = K$, we say that $C_i(K)$ is a *full component* associated with K.

LEMMA 3.14. *Let Ω be a potential maximal clique of a graph G and let $\Delta_G(\Omega)$ be the set of all minimal separators contained in Ω. Then the elements of $\Delta_G(\Omega)$ are exactly the sets $S_i(\Omega)$.*

Proof. We prove that for any i, $1 \leq i \leq p$, S_i is a minimal a, b-separator for some $a \in C_i$ and $b \in \Omega - S_i$. Corollary 3.13 tells us that there is some minimal separator $S \in \Delta_G(\Omega)$ that separates a from $\Omega - S$; recall that $\Omega - S$ is not empty. Since every vertex in S_i has a neighbor in C_i, if S does not contain a vertex $x \in S_i$, S cannot separate $x \in \Omega - S$ from a, so we get $S_i \subseteq S$. By Corollary 3.12, $\Omega - S$ is in a full component associated with S and therefore of S_i. Let b be a vertex of $\Omega - S$.

Then a and b are in different full components associated with S_i, so S_i is a minimal a,b-separator by Lemma 2.6.

We now have to prove that for any minimal separator $S \subseteq \Omega$, there is some $i, 1 \leq i \leq p$, such that $S = S_i$. We have that $\Omega - S \neq \emptyset$ and $\Omega - S$ is in some full component associated with S. Let C be another full component associated with S. Then C is a connected component of $G - \Omega$, let us say C_i. It follows that $S \subseteq S_i$. Suppose there exists a vertex $x \in S_i - S$. Then x has a neighbor in C_i, so x must be in the connected component C of $G - S$, contradicting $C = C_i$. It remains that $S = S_i$. We conclude that the separators of G included in Ω are exactly the sets S_i. □

Remark 1. Actually, each minimal separator S_i separates any vertex of C_i from any vertex of $G - (S_i, C_i)$.

We also give a "sufficient condition" to characterize the potential maximal cliques, which is somehow the dual of Lemma 3.14.

THEOREM 3.15. *Let $K \subseteq V$ be a set of vertices. We denote by $\mathcal{S}$ the set of all $S_i(K)$. K is a potential maximal clique if and only if*

1. *$G - K$ has no full components associated to K;*
2. *$G_{\mathcal{S}}[K]$ is a clique.*

Proof. We prove the "only if" part. Suppose that K is a potential maximal clique of G. By Lemma 3.14, $\mathcal{S} = \Delta_G(K)$. By Theorems 3.10 and 3.11, K is a clique in the graph $G_{\mathcal{S}}$. It remains to show that $G - K$ has no full components associated with K. Let C_i be any connected component of $G - K$. Then S_i is the neighborhood of C_i in K. Since K is a potential maximal clique and S_i is a separator contained in K, we have that S_i is strictly contained in K by Corollary 3.12. Therefore, C_i is not a full component associated with K.

We now prove the "if" part. Let us show at first that for any $i, 1 \leq i \leq p$, S_i is a minimal separator. S_i is clearly a separator and C_i is a full component associated with S_i. Let x be a vertex of $K - S_i$. We show that x belongs to a full component associated with S_i and different from C_i. We denote by C_x the connected component of $G - S_i$ containing x. For any $y \in S_i$, y must have a neighbor in C_x. This is true if x and y are adjacent in G. If x and y are not adjacent, by the second condition of the theorem, x and y belong to a same S_j. C_j being a full component associated with S_j, there is a path in G connecting x to y entirely contained in C_j except from x and y. We deduce that $C_j \subseteq C_x$. It follows that y has a neighbor in C_x since it has a neighbor in C_j. S_i is a minimal separator of G according to Lemma 2.6.

Now, given two distinct separators S_i and S_j, we have to show that they are parallel. We prove that $K - S_i$ is in a connected component of $G - S_i$. Let $x, y \in K - S_i$. If x and y are adjacent they are clearly in the same component of $G - S_i$. Otherwise, since $G_{\mathcal{S}}[K]$ is a clique, they are in a same S_k, so they are connected via C_k. Therefore S_j intersects only the component of $G - S_i$ containing $K - S_i$ and consequently $S_i \| S_j$. Therefore $\mathcal{S}$ is a set of pairwise parallel minimal separators.

We have to show that any separator of G included in K is an element of $\mathcal{S}$. Consider a minimal triangulation H of G such that all the elements of $\mathcal{S}$ are minimal separators of H. We know that K is a clique in H. Now let $U \subseteq K$ be any minimal separator of G. Notice that U must be strictly included in K, otherwise $G - K$ would have two full components associated with K in G, contradicting our choice of K. Clearly U is a clique in H, so by Lemma 3.5, it is a minimal separator of H. Since K is a clique in H, it must be included in some full block associated with U. Let (U, C) be another full block associated with U in H and consequently in G. We have that C

is a connected component of $G-U$ and U separates C and $K-U$. We deduce that C is also a connected component of $G-K$, let us say C_i. By definition of S_i, we have $U \subseteq S_i$. Suppose there exists a vertex $x \in S_i - U$; since x has a neighbor in C, the connected component C of $G-U$ would contain x contradicting $C = C_i$. Therefore we have $U = S_i$ and $U \in \mathcal{S}$.

We want to prove that K and $\mathcal{S} = \Delta_G(K)$ satisfy the conditions of Theorems 3.10 or 3.11. Remark that for any $y \in V - K$, y is in some connected component C_i of $G-K$, and the separator $S_i \in \mathcal{S}$ separates y from $K - S_i$. Now suppose that $\mathcal{S}$ has an element S, maximum by inclusion. Let (S, C) be the block associated with S containing K. By the previous remark, for any $y \in V - K$, S separates y and $K - S$, so $y \notin (S, C)$. It follows that $(S, C) = K$, so K satisfies all the conditions of Theorem 3.10. Now if $\mathcal{S}$ does not have an element maximum by inclusion, K is clearly contained in the piece between the separators of $\mathcal{S}$ in G. By the previous remark, $P_G(\mathcal{S})$ does not contain any $y \in V - K$, so $K = P_G(\mathcal{S})$ and therefore we are under the conditions of Theorem 3.11. It follows that K is a potential maximal clique of G. □

COROLLARY 3.16. *There is an algorithm that, given a graph $G = (V, E)$ and a set of vertices $K \subseteq V$, verifies if K is a potential maximal clique of G. The time complexity of the algorithm is $\mathcal{O}(n^3)$, where n is the number of vertices of G.*

Proof. The algorithm computes the connected components C_i of $G-K$ and their neighborhoods S_i. It then checks the two conditions of Theorem 3.15. Remark that computing the sets C_i, S_i and verifying that $G-K$ has no full components associated with K can be done in linear time. We can complete each set S_i in $\mathcal{O}(n^2)$ steps, and therefore computing the graph $G_{\mathcal{S}}[K]$ takes $\mathcal{O}(n^3)$ time. □

4. Triangulating blocks. In this section we prove that the potential maximal cliques of a graph are sufficient to compute its treewidth and its minimum fill-in.

Let $B = (S, C)$ be a block of the graph G. The graph $R(S, C) = G_S[S \cup C]$ is called the *realization* of the block B. The following lemma, proved in [23], gives a relation between minimal triangulation of a graph and minimal triangulations of some block realizations.

LEMMA 4.1. *Let $S \in \Delta_G$ and let $C_1, C_2, \ldots, C_p$ be the connected components of $G - S$. Suppose that H_i is a minimal triangulation of $R(S, C_i)$ for any $i, 1 \leq i \leq p$. Then the graph $H = (V, E(H))$ with $E(H) = \bigcup_{i=1}^{p} E(H_i)$ is a minimal triangulation of G.*

Conversely, let H be a minimal triangulation of G with $S \in \Delta_H$. Then $H[S \cup C]$ is a minimal triangulation of the realization $R(S, C)$ for each component C of $G - S$.

This gives an equation for computing the treewidth and the minimum fill-in of a graph (see [23] for a proof).

COROLLARY 4.2. *Let G be a noncomplete graph. Then*

$$\mathrm{tw}(G) = \min_{S \in \Delta_G} \max_{C \in \mathcal{C}(S)} \mathrm{tw}(R(S, C)),$$

$$\mathrm{mfi}(G) = \min_{S \in \Delta_G} (\mathrm{fill}(S) + \sum_{C \in \mathcal{C}(S)} \mathrm{mfi}(R(S, C))),$$

where $\mathrm{fill}(S)$ *is the number of nonedges of S.*

We give a version of Lemma 4.1 using potential maximal cliques instead of minimal separators. If Ω is a potential maximal clique of a graph G, we denote as usual

by $C_i, 1 \leq i \leq p$, the connected components of $G - \Omega$ and by $S_i, 1 \leq i \leq p$, the neighborhood of each C_i. We say that the blocks (S_i, C_i) are the *blocks associated with* Ω in G.

THEOREM 4.3. *Let H be a minimal triangulation of G and let Ω be a maximal clique of H. Then for each block (S_i, C_i) associated with Ω in G, the graph $H_i = H[S_i \cup C_i]$ is a minimal triangulation of the realization $R(S_i, C_i)$.*

Conversely, let Ω be a potential maximal clique of G. For each block (S_i, C_i) associated with Ω in G, let H_i be a minimal triangulation of $R(S_i, C_i)$. Then $H = (V, E(H))$ with $E(H) = \bigcup_{i=1}^{p} E(H_i) \cup \{\{x, y\} | x, y \in \Omega\}$ is a minimal triangulation of G.

Proof. For proving the first part, notice that by Theorem 3.15, (S_i, C_i) are blocks of G and of H. Then by Lemma 4.1, $H[S_i \cup C_i]$ are minimal triangulations of $R(S_i, C_i)$.

We now prove the second part. Let us prove that H is a triangulated graph. Suppose that H has a chordless cycle of length at least four. Since Ω is a clique, the cycle is not contained in Ω, so it has a vertex a in some component C_i. Since $H[S_i \cup C_i]$ is chordal, the cycle is not contained in (S_i, C_i). Then at least two nonconsecutive points of the cycle, say, b and c, are in S_i. Therefore the cycle has a chord, namely, the edge $\{b, c\}$.

Now let us prove that H is a minimal triangulation of G. Let $e = \{x, y\}$ be an edge of $H - G$. Suppose at first that x and y are not both contained in Ω. Then x and y are in some H_i. Since e is not in Ω, e is not in S_i. It remains that e is an edge of $H_i - R(S_i, C_i)$. By Lemma 2.1, $H_i - e$ is not chordal, so $H - e$ is not chordal. Now suppose that $x, y \in \Omega$. Then x, y must be in some S_i, since $H[\Omega] = G_{\Delta_G(\Omega)}[\Omega]$ and $e \in H - G$. Consider two full components of $H - S_i$ associated with S_i and a shortest path from x to y in each of them. The two paths form a cycle of length at least four, which is a chordless cycle of $H - e$.

By Lemma 2.1, H is a minimal triangulation of G. □

We want to give a characterization of the minimal triangulations of a realization $R(S, C)$ using the potential maximal cliques Ω with $S \subset \Omega$ and $\Omega \subseteq (S, C)$ and the minimal triangulations of the realizations of some blocks (S_i, C_i), strictly included in (S, C). This will express the treewidth and the minimum fill-in of a realization from the treewidth (respectively, the minimum fill-in) of realizations of smaller blocks, and we will compute the two parameters by dynamic programming on blocks.

The minimal triangulations of the realizations of nonfull blocks are easily reducible to the case of full blocks. For the following lemma, see, for example, [8].

LEMMA 4.4. *Let (S, C) be a nonfull block of G and let S^* be the set of vertices of S having some neighbor in C. Then (S^*, C) is a full block of G and a super graph H of $R(S, C)$ is a minimal triangulation of $R(S, C)$ if and only if $H[S^* \cup C]$ is a minimal triangulation of $R(S^*, C)$.*

We conclude in the following corollary.

COROLLARY 4.5. *Let (S, C) be a nonfull block of G and let S^* be the vertices of S adjacent in G to at least one vertex of C. Then*

$$\mathrm{tw}(R(S, C)) = \max(|S| - 1, \mathrm{tw}(R(S^*, C))),$$

$$\mathrm{mfi}(R(S, C)) = \mathrm{mfi}(R(S^*, C)).$$

It remains to express the treewidth and the minimum fill-in of realizations of full blocks from realizations of smaller blocks.

LEMMA 4.6. *Let $R(S,C)$ be the realization of some full block (S,C) and let $H(S,C)$ be a minimal triangulation of $R(S,C)$. Then there is a maximal clique Ω of $H(S,C)$ such that $S \subset \Omega$ and Ω is a potential maximal clique of G.*

Proof. S clearly is a clique in $H(S,C)$. Consider a maximal clique Ω of $H(S,C)$ containing S. Let us show that S is strictly contained in Ω. C is a full component associated with S in G, so also in $H(S,C)$. If $S = \Omega$, then C is a full component associated with Ω in $H(S,C)$. However, Ω is also a potential maximal clique of the chordal graph $H(S,C)$, so by Theorem 3.15, $H(S,C) - \Omega$ cannot have full components associated with Ω, which leads to a contradiction. It remains to show that Ω is a potential maximal clique of G. By Lemma 4.1, there is a minimal triangulation H of G such that $H[S \cup C] = H(S,C)$. The components associated with S in H are the same as in G. Since S separates C from $V - (C \cup S)$, Ω is a maximal clique of H, so by definition it is a potential maximal clique of G. □

Notice that we have proved that any maximal clique of $H(S,C)$ is a potential maximal clique of G and, moreover, it is a maximal clique of any minimal triangulation H with $H[S \cup C] = H(S,C)$.

By Theorem 4.3 and Lemma 4.6 we have proved the following theorem.

THEOREM 4.7. *Let (S,C) be a full block of a graph G. Then $H(S,C)$ is a minimal triangulation of $R(S,C)$ if and only if there is a potential maximal clique $\Omega \subseteq (S,C)$ of G such that $S \subset \Omega$ and $H(S,C) = (S \cup C, E(H))$ with $E(H) = \bigcup_{i=1}^{p} E(H_i) \cup \{\{x,y\} | x,y \in \Omega\}$, where (S_i, C_i) are the blocks associated with Ω in $H(S,C)$ and H_i are minimal triangulations of $R(S_i, C_i)$.*

COROLLARY 4.8. *Let (S,C) be a full block of G. Then*

$$\mathrm{tw}(R(S,C)) = \min_{S \subset \Omega \subseteq (S,C)} \max(|\Omega| - 1, \mathrm{tw}(R(S_i, C_i))),$$

$$\mathrm{mfi}(R(S,C)) = \min_{S \subset \Omega \subseteq (S,C)} \left(\mathrm{fill}(\Omega) - \mathrm{fill}(S) + \sum \mathrm{mfi}(R(S_i, C_i)) \right),$$

where (S_i, C_i) are the blocks associated with Ω in $R(S,C)$.

We give in Table 4.1 a sketch of the algorithm that, given a graph and the list of all its potential maximal cliques, computes, by standard dynamic programming techniques, the treewidth and the minimum fill-in of the graph. The first part of the algorithm computes the treewidth and the minimum fill-in of the realization of each block (S,C) of G. For computing the treewidth and the minimum fill-in of a realization $R(S,C)$, all we need are the potential maximal cliques Ω such that $S \subset \Omega \subseteq (S,C)$ and the treewidth and the minimum fill-in of the realizations of some blocks strictly contained in (S,C) (see Corollaries 4.5 and 4.8). The treewidth and the minimum fill-in of the input graph is computed, as in Corollary 4.2, using the treewidth and the minimum fill-in of the realizations of its blocks.

THEOREM 4.9. *Given a graph $G = (V,E)$ and the set Π_G of all its potential maximal cliques, we can compute the treewidth and the minimum fill-in of G in $\mathcal{O}(n^2 |\Delta_G| \times |\Pi_G|)$ time.*

Proof. The whole algorithm can be implemented in $\mathcal{O}(bpn + re)$ time, where n and e are the number of vertices, respectively, of edges of G and r, b, and p are the number of minimal separators (respectively, blocks and potential maximal cliques) of G. We make some observations on the relations between r, p, and b. Each minimal separator S induces in $G - S$ at most n connected components, so the number b of blocks is at most rn. The minimal separators contained in a potential maximal

TABLE 4.1
Algorithm computing the treewidth and the minimum fill-in of a graph.

```
Input: G and all its potential maximal cliques
Output: tw(G) and mfi(G)
begin
    compute all the blocks (S, C) and sort them by the number of vertices
    for each block (S, C) taken in increasing order
        if (S, C) is not full then
            compute the neighborhood S* of C
            tw(R(S, C)) := max(|S| − 1, tw(R(S*, C)))
            mfi(R(S, C)) := mfi(R(S*, C))
        else {the block (S, C) is full}
            tw(R(S, C)) := ∞
            mfi(R(S, C)) := ∞
            for each p.m.c. Ω with S ⊂ Ω ⊆ (S, C)
                compute the blocks (S_i, C_i) associated with Ω in R(S, C)
                tw(R(S, C)) := min(tw(R(S, C)),
                                   max_i(|Ω| − 1, tw(R(S_i, C_i))))
                mfi(R(S, C)) := min(mfi(R(S, C)),
                                   fill(Ω) − fill(S) + Σ_i(mfi(R(S_i, C_i))))
        end_if
    end_for
    tw(G) := min_{S∈Δ_G} max_{C∈C(S)} tw(R(S, C))
    mfi(G) := min_{S∈Δ_G} (fill(S) + Σ_{C∈C(S)} mfi(R(S, C)))
end
```

clique Ω correspond to the neighborhoods of the connected components of $G - \Omega$, so Ω contains at most n minimal separators. Thus, $r \leq pn$. The algorithm is clearly polynomial in the size of the input, i.e., in n and p. □

5. Application to some classes of graphs. Several classes of graphs have "few" minimal separators, in the sense that the number of minimal separators of such graphs is polynomially bounded in the size of the graph. Moreover, an algorithm given in [21] computes all the minimal separators of these graphs.

For some of these classes of graphs we also have algorithms computing the treewidth and the minimum fill-in in polynomial time, using the minimal separators (cf. [20, 9, 3, 32, 24, 15, 26]). Different proofs have been given for each of these algorithms. We have remarked that, for computing the treewidth or the minimum fill-in of a graph, all these algorithms compute, in an implicit manner, all the potential maximal cliques of the input graph. Therefore, our approach unifies the cited algorithms (see also [4]).

We will also show how to compute in polynomial time the potential maximal cliques for a new class of graphs, namely, the weakly triangulated graphs (see also [5]).

5.1. AT-free graphs. We say that three vertices (x, y, z) of a graph form an *asteroidal triple* of a graph G if they are pairwise nonadjacent and between every two of them there exists a path avoiding the neighborhood of the third. A graph is *AT-free*

if it has no asteroidal triple. The notion of the asteroidal triple was introduced by Lekkerkerker and Boland [25] in relation to interval graphs.

Notice that the treewidth and the minimum fill-in problems are NP-complete even restricted to the class of cobipartite graphs [1, 34], which is contained in the class of AT-free graphs. It was shown in [23, 26] that the treewidth and the minimum fill-in are tractable in polynomial time for AT-free graphs with a polynomial number of minimal separators. Among the AT-free graphs with few minimal separators, we find the *cocomparability graphs* of bounded dimension and in particular the *permutation graphs* or the *d-trapezoid graphs*, which have a polynomial number of separators for any fixed d.

The following proposition gives an easy way to compute the potential maximal cliques of an AT-free graph G in a time which is polynomial in the size of G and in the number of its minimal separators.

PROPOSITION 5.1. *Let Ω be a potential maximal clique of an AT-free graph G. Then $\Delta_G(\Omega)$ has at most two elements maximal by inclusion.*

Proof. Recall that we denote by $C_1, C_2, \dots C_p$ the connected components of $G - \Omega$ and by S_i the neighborhood of C_i in Ω. By Theorem 3.15, the elements of $\Delta_G(\Omega)$ are exactly the sets of vertices $S_1, \dots, S_p$. Suppose that $\Delta_G(\Omega)$ has three elements maximal by inclusion, say, S_1, S_2, S_3. Let x_1, x_2, and x_3 be three vertices of C_1, C_2, and C_3, respectively. We show that (x_1, x_2, x_3) is an asteroidal triple. Let y be a vertex of $S_2 - S_1$. There is a path from y to x_2 that avoids S_1, so y and x_2 are in the same connected component of $G - S_1$. Since Ω is contained in a block $(S_1, C(S_1))$ associated with S_1, we deduce that y and x_2 are in the connected component $C(S_1)$ of $G - S_1$. For the same reasons, x_3 is in the same connected component $C(S_1)$ of $G - S_1$. Therefore, there is a path from x_2 to x_3 in $C(S_1)$, which clearly avoids the neighborhood of x_1. By symmetry, x_1, x_2, and x_3 form an asteroidal triple. □

COROLLARY 5.2. *An AT-free graph G has $\mathcal{O}(|\Delta_G|^2 + n|\Delta_G|)$ potential maximal cliques, computable in a time polynomial in the number n of vertices of G and the number $|\Delta_G|$ of its minimal separators.*

Proof. In an AT-free graph, we have two types of potential maximal cliques: the potential maximal cliques Ω such that $\Delta_G(\Omega)$ has two distinct elements maximal by inclusion, and the potential maximal cliques such that $\Delta_G(\Omega)$ has one element maximum by inclusion.

Let us prove that if Ω is a potential maximal clique of G and S_1, S_2 are two distinct elements maximal by inclusion in $\Delta_G(\Omega)$, then $\Omega = P(S_1, S_2)$. For any $T \in \Delta_G(\Omega)$, let $(T, C(T))$ be the unique block of T containing Ω. If $T \subseteq S_1$, then $(T, C(T))$ contains the block $(S_1, C(S_1))$. It follows that $\cap_{T \in \Delta_G(\Omega)}(T, C(T) = (S_1, C(S_1)) \cap (S_2, C(S_2))$, so $P(\Delta_G(\Omega)) = P(S_1, S_2)$. Thus, we have $\mathcal{O}(|\Delta_G|^2)$ potential maximal cliques Ω with two maximal elements in $\Delta_G(\Omega)$. Clearly, all these potential maximal cliques can be computed in polynomial time.

If Ω is a potential maximal clique and S is the unique element maximum by inclusion in $\Delta_G(\Omega)$, then by Theorem 3.10 we have $\Omega = (S, C)$, where (S, C) is a full block associated with S. Consequently, we have at most $n|\Delta_G|$ potential maximal cliques of this type, computable in polynomial time. □

Remark 2. One can prove that an AT-free graph has at most $\mathcal{O}(n^2|\Delta_G|)$ potential maximal cliques. This result follows directly from [26].

The notion of asteroidal triple can be extended to more than three vertices. We say that a set of vertices A is an *asteroidal set* if for any $a \in A$, $A - \{a\}$ is contained in a same connected component of $G - N(a)$, where $N(a)$ is the neighborhood of a

in G. In particular, an asteroidal triple is an asteroidal set of cardinality 3. The *asteroidal number* of a graph is $\mathrm{na}(G) = \max\{|A|,\ A \text{ is an asteroidal set of } G\}$. In [7] it is proved that the treewidth and the minimum fill-in are polynomially tractable for graphs with a bounded asteroidal number and with few minimal separators. As for the AT-free graphs, a potential maximal clique of G has at most $\mathrm{na}(G)$ maximal elements, so a graph has at most $\mathcal{O}(|\Delta_G|^{\mathrm{na}(G)} + n|\Delta_G|)$ potential maximal cliques. We deduce as in Corollary 5.2 that if $\mathrm{na}(G)$ is bounded by a constant c and if G has few minimal separators, then all the potential maximal cliques of G can be listed in polynomial time.

5.2. Circle and circular arc graphs. *Circle* and *circular arc graphs* are obtained from an intersection model. A graph $G = (V, E)$ is a *circle graph* if and only if we can associate each vertex of the graph with a chord of a circle such that two vertices are adjacent in the graph if and only if the corresponding chords cross. The circle and its chords are the *circle model* $D(G)$ of the graph. In the same manner, a graph G is a *circular arc graph* if each vertex can be associated with an arc of a circle and two vertices are adjacent if and only if the corresponding arcs overlap. The circle and its circular arcs are said to be the *circular arc model* $D(G)$ of the graph.

Without loss of generality, we can assume that no two chords of the circle model (respectively, no two arcs of the circular arc model) share an endpoint. For both circle and circular arc graphs, there are recognition algorithms working in $\mathcal{O}(n^2)$ time, which also compute the circle (respectively, the circular arc model of a graph) (see [30, 11]). Therefore, we will assume that an intersection model (i.e., a circle (respectively, a circular-arc model)) of the input graph is always given.

The treewidth and the minimum fill-in problems have been solved for the circle and the circular arc graphs in $\mathcal{O}(n^3)$ time [16, 32, 24]. We show here that all the potential maximal cliques of a circle or a circular arc graph can be listed in polynomial time. We will use the results of [24] in order to prove this assertion. Actually, the cited algorithms compute, in an implicit manner, all the potential maximal cliques of the input graph.

We will present here in detail only the circle graphs. The results can be extended to circular arc graphs using the same techniques.

For these "geometrical" classes of graphs, the minimal separators can be modeled by *scanlines*. In the circle model of a circle graph, we add a *scanpoint* between every two consecutive endpoints of the chords. The scanlines are the straight line segments between two scanpoints.

Let G be a circle graph and $D(G)$ be a circle model of G. For each scanline s, we denote by $S(s)$ the set of all vertices of G corresponding to the chords of $D(G)$ that intersect s.

Kloks [16] proved the following proposition.

PROPOSITION 5.3. *Let G be a circle graph and let $D(G)$ be a circle model of G. For any minimal separator S of G, there is a scanline s in $D(G)$ such that $S = S(s)$.*

Since the circle model has n chords, we have $2n$ scanpoints and consequently $n(2n-1)$ scanlines. Therefore, a circle graph has $\mathcal{O}(n^2)$ minimal separators.

Kloks, Kratsch, and Wong [24] gave a characterization of minimal triangulations of a circle graph G using scanlines of the circle model. The scanpoints of $D(G)$ form a convex polygon P. A *planar triangulation* of P is a set T of noncrossing diagonals of P dividing the interior of P in triangles. Notice that if P has n vertices, then T has $n-3$ diagonals dividing the interior of P in $n-2$ triangles.

If T is a planar triangulation of P, the graph $H(T)$ is defined as the graph with

the same vertex set as G, and two vertices x and y are adjacent in $H(T)$ if and only if the chords corresponding to x and y in $D(G)$ intersect a same triangle of T. Clearly, $H(T)$ is a supergraph of G. Moreover, it is proved in [24] that $H(T)$ is a triangulation of G. However, we will use only the following theorem of [24].

THEOREM 5.4. *Let G be a circle graph, $D(G)$ a circle model of G, and P its polygon of scanpoints. Then for any minimal triangulation H of G, there is a planar triangulation T of P such that $H = H(T)$.*

All we have to do is to notice how maximal cliques are formed in $H(T)$. Let Q be a triangle of a planar triangulation T of P. We denote by $S(Q)$ the set of all vertices of G for which the corresponding chords intersect Q.

PROPOSITION 5.5. *Let G be a circle graph, $D(G)$ a circle model of G, and T a planar triangulation of its polygon P of scanpoints. Then for any maximal clique Ω of $H(T)$, there is a triangle Q of T such that $\Omega = S(Q)$.*

Proof. Clearly, for any triangle Q of T, the set of vertices $S(Q)$ induces a clique in $H(T)$.

We want to prove that all the chords corresponding to vertices of Ω intersect a same triangle Q. Let s be any diagonal of T and let x, y be two adjacent vertices of $H(T)$. The scanline s induces two regions $\mathcal{R}_1$ and $\mathcal{R}_2$ in the polygon P. Since x and y are adjacent in $H(T)$, the chords of $D(G)$ corresponding to x (respectively, y) cannot be in different regions induced by s. Consequently, all the chords corresponding to vertices of Ω intersect a same region induced by s, say, $\mathcal{R}_1$. If $\mathcal{R}_1$ is a triangle, we have found our triangle Q. Otherwise, let s' be a diagonal of T contained in $\mathcal{R}_1$. Then s' divides $\mathcal{R}_1$ in two smaller regions $\mathcal{R}'_1$ and $\mathcal{R}'_2$. For the same reasons as previously, the chords corresponding to vertices of Ω intersect a same subregion, say, $\mathcal{R}'_1$. We can iterate the process until finding a triangle Q such that $\Omega \subseteq S(Q)$.

We have that $\Omega \subseteq S(Q)$, $S(Q)$ is a clique in $H(T)$, and Q is a maximal clique of $H(T)$. It follows that $\Omega = S(Q)$. □

COROLLARY 5.6. *A circle graph has $\mathcal{O}(n^3)$ potential maximal cliques computable in polynomial time.*

Proof. Let Ω be a potential maximal clique of G and H a minimal triangulation of G such that Ω is a maximal clique of H. By Theorem 5.4, there is a planar triangulation T of the polygon of scanpoints such that $H = H(T)$. By Proposition 5.5, there is a triangle Q of scanlines such that $\Omega = S(Q)$.

Consequently, for any potential maximal clique Ω there is a triangle of scanpoints Q such that $\Omega = S(Q)$. Since we have at most $\mathcal{O}(n^3)$ triangle of scanpoints, G has at most $\mathcal{O}(n^3)$ potential maximal cliques.

For listing all the potential maximal cliques of G, it is sufficient to list all the triangles of scanpoints Q, to compute $S(Q)$, and to check if $S(Q)$ is a potential maximal clique of G. This enumeration can be done in polynomial time. □

Therefore for circle graphs, all the potential maximal cliques can be listed in polynomial time. Notice that the algorithms of [16, 24] that compute the treewidth and the minimum fill-in of circle graphs are looking for triangulations of type $H(T)$ of G. In particular, all the minimal triangulations of G are of this type. Moreover, these algorithms compute all the sets $S(Q)$ for all the triangles of scanlines. Therefore they implicitly use all the potential maximal cliques of the input graph. The fact that they are using triangulations of type $H(T)$ of G instead of using only minimal triangulations leads to an efficient algorithm in time $\mathcal{O}(n^3)$.

For characterizing the potential maximal cliques of a circle graph, we have used the global characterization of its minimal triangulations given by Theorem 5.4. One

proves directly that, under certain conditions, two minimal separators of a circle graph are parallel if and only if the corresponding scanlines do not cross (they may have a common endpoint). Using Theorem 3.15 and the same arguments as for the proof of Proposition 5.5, we can prove that, if Ω is a potential maximal clique of G, then the scanlines corresponding to the minimal separators of $\Delta_G(\Omega)$ determine a region $\mathcal{R}$ such that $\Omega = S(\mathcal{R})$ (the chords corresponding to vertices of Ω are exactly the chords intersecting $\mathcal{R}$). One can deduce directly the Proposition 5.5 (see [33]).

For the class of circular arc graphs, the results are very similar. Let G be a circular arc graph and $D(G)$ be a circular arc model of G. For a point p of the circle, we denote by $S(p)$ the vertices of G for which the corresponding arcs contain p. We place a point p between every two consecutive endpoints u and v of the arcs of $D(G)$. We say that p is a scanpoint if $|S(p)| < \min(|S(u)|, |S(v)|)$. Once again, a straight line s between two scanpoints is called a scanline. If p_1 and p_2 are the scanpoints determining the scanline s, we define $S(s) = S(p_1) \cup S(p_2)$, i.e., the vertices of G such that the corresponding arcs contain a point of the scanline. If Q is a triangle of scanpoints of vertices p_1, p_2, p_3, then we put $S(Q) = S(p_1) \cup S(p_2) \cup S(p_3)$. We can also define a triangulation T of the polygon P of scanpoints and the supergraph graph $H(T)$ of G in which two vertices x and y are adjacent if they are adjacent in G or if there is a scanline $s \in T$ with $x, y \in S(s)$.

Kloks, Kratsch, and Wong [24] proved the following results.

THEOREM 5.7. *Let G be a circular arc graph, $D(G)$ a circular arc model of G, and P its polygon of scanpoints. Then for any minimal triangulation H of G there is a planar triangulation T of P such that $H = H(T)$.*

We can give the same characterization of the potential maximal cliques as in the case of circle graph. The proof is almost the same as for Proposition 5.5, so we omit it.

PROPOSITION 5.8. *Let G be a circular arc graph, $D(G)$ a circular arc model of G, and T a planar triangulation of its polygon of scanpoints P. Then for any maximal clique Ω of $H(T)$, there is a triangle of scanpoints Q of T such that $\Omega = S(Q)$.*

Clearly, circular arc graphs have a polynomial number of potential maximal cliques and we can list these potential maximal cliques in polynomial time.

COROLLARY 5.9. *A circular arc graph has $\mathcal{O}(n^3)$ potential maximal cliques computable in polynomial time.*

Once again, the algorithms of [32, 24] compute all the sets of type $S(Q)$, so in particular they use the potential maximal cliques of the input graph.

5.3. Weakly triangulated graphs. We now consider two nonadjacent vertices x, y of an arbitrary graph G. Let G' be the graph obtained from G by adding the edge $\{x, y\}$. We will show in this section that the potential maximal cliques of G can be computed from the minimal separators of G and the potential maximal cliques of G'. We will use this technique to compute all the potential maximal cliques of any weakly triangulated graph.

Let Ω once again be a potential maximal clique of G. Let $C_1, \ldots, C_p$ be the connected components of $G - \Omega$ and let S_i be the set of vertices of Ω having at least a neighbor in C_i. We want to describe the behavior of Ω and $\mathcal{S}$ in the graph G'. We deduce the following lemma directly from Theorem 3.15.

LEMMA 5.10.

1. *If $x, y \in \Omega$ or there is an $i, 1 \leq i \leq p$, such that $x, y \in C_i$ or $x \in S_i$ and $y \in C_i$, then Ω is a potential maximal clique of G' and the elements of $\Delta_{G'}(\Omega)$ are $S_1, S_2, \ldots, S_p$.*

2. *If $y \in C_1$, $x \in \Omega - S_1$, and $\Omega \neq S_1 \cup \{x\}$, then Ω is a potential maximal clique of G' and the elements of $\Delta_{G'}(\Omega)$ are $S_1 \cup \{x\}, S_2, \ldots, S_p$.*
3. *If $x \in C_1$, $y \in C_2$, and $\Omega \neq S_1 \cup S_2$, then Ω is a potential maximal clique of G' and the elements of $\Delta_{G'}(\Omega)$ are $S_1 \cup S_2, S_3, \ldots, S_p$.*

Proof. By Theorem 3.15, $G - \Omega$ has no full components associated with Ω and $G_{\Delta_G(\Omega)}[\Omega]$ is a clique. Notice that, by Lemma 3.14, the elements of $\Delta_G(\Omega) = \mathcal{S}(\Omega)$ are exactly the sets $S_1, \ldots, S_p$.

If we are in the first case of the lemma, then the connected components of $G' - \Omega$ are exactly the same in $G - \Omega$, and their neighborhoods in G' are also the same. If we denote by $\mathcal{S}'(\Omega)$ the neighborhoods of the connected components of $G' - \Omega$, then $\mathcal{S}'(\Omega)$ consists of $S_1, S_2, \ldots, S_p$. Clearly, $G' - \Omega$ has no full components associated with Ω and $G_{\mathcal{S}'(\Omega)}[\Omega]$ is a clique. By Theorem 3.15, Ω is a potential maximal clique of G'.

If we are in the second case, the connected components of $G' - \Omega$ are $C_1, C_2 \ldots, C_p$ and their neighborhoods in G' are $S_1 \cup \{x\}, S_2, \ldots, S_p$. If $\Omega \neq S_1 \cup \{x\}$, then $G' - \Omega$ has no full components associated with Ω, and once again Ω is a potential maximal clique of G', by Theorem 3.15.

In the third case, the connected components of $G' - \Omega$ are $C_1 \cup C_2, C_3, \ldots, C_p$, and their neighborhoods are $S_1 \cup S_2, S_3, \ldots, S_p$, respectively. The fact that $\Omega \neq S_1 \cup S_2$ ensures that $G' - \Omega$ has no full components associated with Ω. Clearly $G_{\mathcal{S}'(\Omega)}[\Omega]$ is a clique, so by Theorem 3.15, Ω is a potential maximal clique of G'. □

The following theorem follows directly.

THEOREM 5.11. *Let Ω be a potential maximal clique of G. Let x, y be two nonadjacent vertices of G and let $G' = G \cup \{x, y\}$. Two cases are possible.*

1. *Ω can be written as $S_1 \cup \{x\}$, $S_1 \cup \{y\}$ or $S_1 \cup S_2$, where S_1, S_2 are minimal x, y-separators of G.*
2. *Ω is a potential maximal clique of G'.*

The weakly triangulated graphs were introduced in [13]. A graph G is called *weakly triangulated* if neither G nor its complement $\overline{G}$ have an induced cycle with strictly more than four vertices. This class contains the chordal graphs, the chordal bipartite graphs, and the distance hereditary graphs.

We denote by $N(x)$ the neighbors of the vertex x. We say that two vertices x, y of a graph G form a *two-pair* if their common neighbors $N(x) \cap N(y)$ form an x, y-separator. The following theorem was proved in [14].

THEOREM 5.12. *If G is a weakly triangulated graph, then every induced subgraph of G that is not a clique contains a two-pair.*

Spinrad and Sritharan give in [31] an algorithm recognizing the weakly triangulated graphs based on the following theorem.

THEOREM 5.13. *Let $G = (V, E)$ be a graph and let $\{x, y\}$ be a two-pair of G. Let $G' = (V, E')$ be the graph obtained from G by adding the edge $\{x, y\}$. Then G is weakly triangulated if and only if G' is weakly triangulated.*

Notice that a clique is a weakly triangulated graph. The recognition algorithm considers an input graph G and, while G has a two-pair $\{x, y\}$, it adds the edge between x and y to G. At the end of the loop, either G becomes a clique, in which case the initial graph is weakly triangulated by Theorem 5.13, or G is not a clique and it has no two-pair, in which case the input graph cannot be weakly triangulated by Theorems 5.12 and 5.13.

We denote by $\overline{e}$ the number of edges of $\overline{G}$. Now let $G = (V, E)$ be a weakly triangulated graph and let $f_1 = \{x_1, y_1\}, \ldots, f_{\overline{e}} = \{x_{\overline{e}}, y_{\overline{e}}\}$ be the edges added to

G by the recognition algorithm in this order. In particular, $(V, E \cup \{f_1, \ldots, f_{\overline{e}}\})$ is a clique. We denote by G_i the graph $(V, E \cup \{f_1, f_2, \ldots, f_i\})$ with $0 \leq i \leq \overline{e}$ (so $G_0 = G$ and $G_{\overline{e}}$ is a clique). We will describe the minimal separators (respectively, the potential maximal cliques of G_i) using the minimal separators (respectively, the potential maximal cliques of G_{i+1}) for any $i < \overline{e}$.

It is known that a weakly triangulated graph has at most $\overline{e}$ minimal separators (Kloks [17]). We give here the proof of this result because we will reuse the same technique for counting and listing the potential maximal cliques of a weakly triangulated graph.

THEOREM 5.14. *Let G be a noncomplete weakly triangulated graph and let $\{x, y\}$ be a two-pair of G. Let S_{xy} be the set $N(x) \cap N(y)$. Consider the graph G' obtained from G by adding the edge $\{x, y\}$. Then $\Delta_G \subseteq \Delta_{G'} \cup \{S_{xy}\}$.*

Proof. Notice that S_{xy} is a minimal x, y-separator of G by definition of a two-pair.

Let S be any minimal separator of G.

Suppose at first that S separates x and y, and let C_x, C_y be the connected components of $G - S$ containing x (respectively, y). If both C_x and C_y are full components associated with S, then S is a minimal x, y-separator by Lemma 2.6. Notice that $N(x) \subset C_x \cup S$ and $N(y) \subset C_y \cup S$. We deduce that $N(x) \cap N(y) \subseteq S$, and since $N(x) \cap N(y) = S_{xy}$ is a x, y-separator in G by definition of a two-pair, it follows that $S = S_{xy}$. Now suppose that C_x and C_y are not both full components of S in G. The connected components of $G' - S$ are the same as in $G - S$, except for C_x and C_y which form a unique component $C_x \cup C_y$ in $G' - S$. If $G - S$ has two full components D and E associated with S, both different from C_x and C_y, then D and E are full components associated with S in G', so S is a minimal separator of G' by Lemma 2.6. Otherwise, $G - S$ has a unique full component D associated with S and different from C_x and C_y, so at least one of C_x, C_y must be a full component associated with S in G. Therefore, D and $C_x \cup C_y$ are full components associated with S in G', so S is a minimal separator of G' by Lemma 2.6.

If S does not separate x and y, the connected components of $G' - S$ are the same as in $G - S$, so $G' - S$ has two full components associated with S. Consequently, S is a minimal separator of G'. □

Therefore for any $i < \overline{e}$, the graph G_i has at most one more minimal separator than G_{i+1}. We deduce the following corollary.

COROLLARY 5.15. *A weakly triangulated graph G has at most $\overline{e}$ minimal separators, where $\overline{e}$ is the number of edges of $\overline{G}$.*

We can conclude directly from Theorem 5.11 and Lemma 5.15 that all the potential maximal cliques of a weakly triangulated graph can be computed in polynomial time. However, we can refine the results of the third case of Lemma 5.10.

LEMMA 5.16. *Let G be a graph, let x, y be a two-pair of G, and let Ω be a potential maximal clique of G such that x and y are in different connected components of $G - \Omega$. Then Ω is a potential maximal clique of $G' = G \cup \{x, y\}$.*

Proof. We use the fact that if x, y is a two-pair of a graph G, then $S_{xy} = N(x) \cap N(y)$ is the only x, y-minimal separator of G. Let C_1 and C_2 be the connected components of $G - \Omega$ containing x (respectively, y). As before, let S_1 and S_2 be the sets of vertices of Ω having a neighbor in C_1 (respectively, C_2). We want to prove that both S_1 and S_2 contain S_{xy} and at least one of them is equal to S_{xy}. By Remark 1, S_1 separates x and y, so it must contain $N(x) \cap N(y) = S_{xy}$. The same holds for S_2. Suppose that there is some vertex $a \in S_1 - S_{xy}$ and some vertex $b \in S_2 - S_{xy}$. Since a has a neighbor in C_1, a is in the connected component $G - S_{xy}$ containing x. b is

also in the connected component of $G - S_{xy}$ containing y. Then Ω intersects different connected components of $G - S_{xy}$, which is, by Corollary 3.12, a contradiction to the fact that $S_{xy} \subset \Omega$. It remains that $S_1 = S_{xy}$ or $S_2 = S_{xy}$. Therefore, $S_1 \cup S_2$ is equal to S_1 or S_2, and consequently $\Omega \neq S_1 \cup S_2$ by Corollary 3.12. By the third part of Lemma 5.10, Ω will be a potential maximal clique of G'. □

We deduce from Theorem 5.11 and Lemma 5.16 the following corollary.

COROLLARY 5.17. *Let Ω be a potential maximal clique of G. Let x, y be a two-pair of G and let $G' = G \cup \{xy\}$. Let $S_{xy} = N(x) \cap N(y)$. Two cases are possible.*

1. *Ω can be written as $S_{xy} \cup \{x\}$ or $S_{xy} \cup \{y\}$.*
2. *Ω is a potential maximal clique of G'.*

COROLLARY 5.18. *A weakly triangulated graph G has at most $2\overline{e} + 1$ potential maximal cliques.*

Proof. We consider the sequence of graphs $G_0 = G, G_1, \ldots, G_{\overline{e}}$ previously defined. Since G_{i+1} is obtained from G_i by adding an edge between a two-pair, by Corollary 5.17 the graph G_i has at most two more potential maximal cliques than G_{i+1}. Clearly $G_{\overline{e}}$, which is a clique, has a unique potential maximal clique. □

COROLLARY 5.19. *The treewidth and the minimum fill-in of weakly triangulated graphs can be computed in polynomial time.*

The complexity of computing all the potential maximal cliques is $\mathcal{O}(n^3\overline{e})$. It is sufficient to use the weakly triangulated graphs recognition algorithm of [31] to generate the two-pairs $\{x_i, y_i\}$ and the sets $S_{x_i y_i}$. This takes $\mathcal{O}(n^2\overline{e})$ time. One can check in $\mathcal{O}(n^3)$ time if a set is a potential maximal clique by Corollary 3.16. Therefore the list of all potential maximal cliques is computable in $\mathcal{O}(n^3\overline{e})$ time.

For weakly triangulated graphs, the number of minimal separators is $r = \mathcal{O}(\overline{e})$, the number of blocks is $b = \mathcal{O}(n\overline{e})$, and the number of potential maximal cliques is $p = \mathcal{O}(\overline{e})$. According to the complexity of the algorithm of Corollary 3.16, the treewidth and the minimum fill-in are computable in $\mathcal{O}(n^2\overline{e}^2 + n^3\overline{e})$ time.

6. Conclusion. The main result of this paper is that, given a graph G and the list Π_G of all its potential maximal cliques, the treewidth and the minimum fill-in of G can be determined in polynomial time in the size of G and the number $|\Pi_G|$ of its potential maximal cliques.

We did not say whether there is a connection between the number of potential maximal cliques and the number of minimal separators in a graph. We have recently proved (see [6]) that, for any graph, $|\Pi_G| = \mathcal{O}(n|\Delta_G|^2)$ and the potential maximal cliques can be computed in polynomial time in the size of the graph and the number of its minimal separators. Thus, the treewidth and the minimum fill-in are tractable in polynomial time in the size of the graph and the number of its minimal separators.

REFERENCES

[1] S. ARNBORG, D. G. CORNEIL, AND A. PROSKUROWSKI, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.

[2] J. R. S. BLAIR AND B. PEYTON, *An introduction to chordal graphs and clique trees*, in Graph Theory and Sparse Matrix Computations, A. George, J. R. Gilbert, and J. H. U. Liu, eds., Springer-Verlag, New York, 1993, pp. 1–29.

[3] H. L. BODLAENDER, T. KLOKS, AND D. KRATSCH, *Treewidth and pathwidth of permutation graphs*, in Proceedings of the 20th International Colloquium on Automata, Languages and Programming (ICALP'93), Lecture Notes in Comput. Sci. 700, Springer-Verlag, Berlin, 1993, pp. 114–125.

[4] V. Bouchitté and I. Todinca, *Minimal triangulations for graphs with "few" minimal separators*, in Proceedings of the Sixth Annual European Symposium on Algorithms (ESA'98), Lecture Notes in Comput. Sci. 1461, Springer-Verlag, Berlin, 1998, pp. 344–355.

[5] V. Bouchitté and I. Todinca, *Treewidth and minimum fill-in of weakly triangulated graphs*, in Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99), Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 197–206.

[6] V. Bouchitté and I. Todinca, *Listing all potential maximal cliques of a graph*, in Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000), Lecture Notes in Comput. Sci. 1770, Springer-Verlag, Berlin, 2000, pp. 503–515.

[7] H. Broersma, T. Kloks, D. Kratsch, and H. Müller, *A generalization of AT-free graphs and a generic algorithm for solving treewidth, minimum fill-in and vertex ranking*, in Workshop on Graph-theoretic Concepts in Computer Science (WG'98), Lecture Notes in Comput. Sci. 1517, Springer-Verlag, Berlin, 1998, pp. 88–99.

[8] H. J. Broersma, E. Dahlhaus, and T. Kloks, *Algorithms for the treewidth and minimum fill-in of HHD-free graphs*, in Workshop on Graph-Theoretic Concepts in Computer Science (WG'97), Lecture Notes in Comput. Sci. 1335, Springer-Verlag, Berlin, 1997, pp. 109–117.

[9] M. S. Chang, *Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs*, in Proceedings of the Seventh Annual International Symposium on Algorithms and Computation (ISAAC'96), Lecture Notes in Comput. Sci. 1178, Springer-Verlag, Berlin, 1996, pp. 146–155.

[10] G. A. Dirac, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71–76.

[11] E. M. Eschen and J. P. Spinrad, *An $O(n^2)$ algorithm for circular-arc graph recognition*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'93), Austin, TX, 1993, pp. 128–137.

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

[13] R. Hayward, *Weakly triangulated graphs*, J. Combin. Theory Ser. B, 39 (1985), pp. 200–208.

[14] R. Hayward, C. Hoàng, and F. Maffray, *Optimizing weakly triangulated graphs*, Graphs Combin., 5 (1989), pp. 339–349.

[15] T. Kloks, *Treewidth of circle graphs*, in Proceedings of the Fourth Annual International Symposium on Algorithms and Computation (ISAAC'93), Lecture Notes in Comput. Sci. 762, Springer-Verlag, Berlin, 1993, pp. 108–117.

[16] T. Kloks, *Treewidth of circle graphs*, Internat. J. Found. Comput. Sci., 7 (1996), pp. 111–120.

[17] T. Kloks, *private communication*, University of Twente, 1998.

[18] T. Kloks, H. L. Bodlaender, H. Müller, and D. Kratsch, *Computing treewidth and minimum fill-in: All you need are the minimal separators*, in Proceedings of the First Annual European Symposium on Algorithms (ESA'93), Lecture Notes in Comput. Sci. 726, Springer-Verlag, Berlin, 1993, pp. 260–271.

[19] T. Kloks, H. L. Bodlaender, H. Müller, and D. Kratsch, *Erratum to the ESA'93 proceedings*, in Proceedings of the Second Annual European Symposium on Algorithms (ESA'94), Lecture Notes in Comput. Sci. 855, Springer-Verlag, Berlin, 1994, p. 508.

[20] T. Kloks and D. Kratsch, *Treewidth of chordal bipartite graphs*, J. Algorithms, 19 (1995), pp. 266–281.

[21] T. Kloks and D. Kratsch, *Listing all minimal separators of a graph*, SIAM J. Comput., 27 (1998), pp. 605–613.

[22] T. Kloks, D. Kratsch, and H. Müller, *Approximating the bandwidth for asteroidal triple-free graphs*, in Proceedings of the Third Annual European Symposium on Algorithms (ESA'95), Lecture Notes in Comput. Sci. 979, Springer-Verlag, Berlin, 1995, pp. 434–447.

[23] T. Kloks, D. Kratsch, and J. Spinrad, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theoret. Comput. Sci., 175 (1997), pp. 309–335.

[24] T. Kloks, D. Kratsch, and C. K. Wong, *Minimum fill-in of circle and circular-arc graphs*, J. Algorithms, 28 (1998), pp. 272–289.

[25] C. G. Lekkerkerker and J. Ch. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.

[26] A. Parra, *Structural and Algorithmic Aspects of Chordal Graph Embeddings*, Ph.D. thesis, Technische Universität Berlin, Berlin, Germany, 1996.

[27] A. Parra and P. Scheffler, *Characterizations and algorithmic applications of chordal graph embeddings*, Discrete Appl. Math., 79 (1997), pp. 171–188.

[28] N. Robertson and P. Seymour, *Graphs minors.* II. *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[29] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on*

graphs, SIAM J. Comput., 5 (1976), pp. 266–283.
[30] J. Spinrad, *Recognition of circle graphs*, J. Algorithms, 16 (1994), pp. 264–282.
[31] J. Spinrad and R. Sritharan, *Algorithms for weakly triangulated graphs*, Discrete Appl. Math., 59 (1995), pp. 181–191.
[32] R. Sundaram, K. Sher Singh, and C. Pandu Rangan, *Treewidth of circular-arc graphs*, SIAM J. Discrete Math., 7 (1994), pp. 647–655.
[33] I. Todinca, *Aspects Algorithmiques des Triangulations Minimales des Graphes*, Ph.D. thesis, École Normale Supérieure de Lyon, Lyon, France, 1999.
[34] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 77–79.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 233–258

THE ACCOMMODATING FUNCTION: A GENERALIZATION OF THE COMPETITIVE RATIO*

JOAN BOYAR†, KIM S. LARSEN†, AND MORTEN N. NIELSEN†

Abstract. A new measure, the *accommodating function*, for the quality of on-line algorithms is presented. The accommodating function, which is a generalization of both the competitive ratio and the competitive ratio on accommodating sequences, measures the quality of an on-line algorithm as a function of the resources that would be sufficient for an optimal off-line algorithm to fully grant all requests. More precisely, if we have some amount of resources n, the function value at α is the usual ratio (still on some fixed amount of resources n), except that input sequences are restricted to those where the optimal off-line algorithm will not obtain a better result by having more than the amount αn of resources.

The accommodating functions for three specific on-line problems are investigated: a variant of bin packing in which the goal is to maximize the number of items put in n bins, the seat reservation problem, and the problem of optimizing total flow time when preemption is allowed.

We also show that when trying to distinguish between two algorithms, the decision as to which one performs better cannot necessarily be made from the competitive ratio or the competitive ratio on accommodating sequences alone. For the variant of bin-packing considered, we show that Worst-Fit has a strictly better competitive ratio than First-Fit, while First-Fit has a strictly better competitive ratio on accommodating sequences than Worst-Fit.

Key words. on-line algorithms, performance measures, competitive analysis, restricted adversaries, bin packing, seat reservations, flow time

AMS subject classifications. 68Q25, 05B40, 90B35

PII. S0097539799361786

1. Introduction.

The competitive ratio [17, 29, 23], as a measure for the quality of on-line algorithms, has been criticized for giving bounds that are unrealistically pessimistic [5, 7, 18, 22, 25] and for not being able to distinguish between algorithms with very different behavior in practical applications [7, 18, 25, 28]. Though this criticism also applies to standard worst-case analysis, it is often more disturbing in the on-line scenario [18].

The basic problem is that the adversary is too powerful compared with the on-line algorithm. For instance, it would often be more interesting to compare an on-line algorithm to other on-line alternatives than to an all-powerful off-line algorithm. A number of papers have addressed this problem [16] by making the on-line algorithm more powerful, by providing the on-line algorithm with more information, or by restricting the set of legal input sequences.

With regard to providing the on-line algorithm with more information, most progress has been made on paging problems. It has been observed [7] that programs exhibit "locality of reference." By supplying an "access graph" as part of the input to the algorithms, this behavior can be modeled. In [7, 19], a number of classes of

*Received by the editors September 30, 1999; accepted for publication (in revised form) March 27, 2001; published electronically July 25, 2001. A few of the results in this paper appeared in [10]. This work was supported in part by the ESPRIT Long Term Research Programme of the EU under project 20244 (ALCOM-IT) and by grants from the Danish Natural Sciences Research Council (SNF).

http://www.siam.org/journals/sicomp/31-1/36178.html

†Department of Mathematics and Computer Science, University of Southern Denmark, Main Campus: Odense University, Campusvej 55, DK-5230 Odense M, Denmark (joan@imada.sdu.dk, kslarsen@imada.sdu.dk, nyhave@imada.sdu.dk).

access graphs have been studied. In [11], it is shown that LRU is never worse than FIFO on any access graph.

The "loose competitive ratio" from [31] represents another attempt at improving the ratio. When determining the loose competitive ratio c, the following steps are taken. First, from an infinite set of input sequences, a set of sequences, asymptotically smaller than the whole set, may be disregarded. The remaining sequences should then either be c-competitive or have small cost. The assumption is that sequences of small cost are relatively unimportant.

With regard to making the on-line algorithm more powerful, this has been achieved through so-called "extra-resource analysis" of scheduling problems. In [22] and [28], processor speed is a resource which to some degree compensates for the on-line algorithm's lack of knowledge of the future compared with the (optimal) off-line algorithm. In [28], reduced job arrival rate is also considered as an extra resource the on-line algorithms could be allowed. Another possibility for an extra resource in scheduling problems is extra machines. Graham in [17] compares two arbitrary algorithms, allowing different numbers of processors for the two algorithms. This work was done before the competitive ratio was formally defined, but if one of these two algorithms is the optimal off-line algorithm, this result can be viewed as allowing the on-line algorithm extra machines. In [30, 31], the competitive ratio is improved by allowing some limited look-ahead.

In [25], unrealistic sequences can be removed by specifying a collection of possible distributions. The off-line adversary will choose the distribution which maximizes the ratio of the expected performance of the on-line algorithm to the expected performance of the adversary.

In this paper, we obtain new and stronger results by restricting the adversary. This can be done various ways; we move in the direction of restricting the set of input sequences the adversary can supply. However, instead of a "fixed" restriction, we consider a function of the restriction, the *accommodating function*. Informally, in on-line problems, where requests are made for parts of some resource, we measure the quality of an on-line algorithm as a function of the resources that would be sufficient for an optimal off-line algorithm. More precisely, if we have some amount of resources n, the function value at α is the usual ratio (still on some fixed amount of resources n), except that input sequences are restricted to those where the optimal off-line algorithm will not obtain a better result by having more than the amount αn of resources.

In the limit, as α tends towards infinity, there is no restriction on the input sequence, so this is the competitive ratio. However, when α is very large, the allowed sequences cannot necessarily be handled very well by the optimal off-line algorithm and can, depending on the application, be quite unrealistic. To avoid comparing an on-line algorithm to the optimal off-line algorithm on problematic sequences of this type, we consider smaller values of α and restrict the adversary so that it can supply only sequences which the optimal off-line algorithm could handle optimally with only αn resources. In the case where increasing the amount of resources available will not improve the optimal off-line algorithm's result, the sequences are called *accommodating sequences* [9].[1] Thus, when $\alpha = 1$, the function value is the competitive ratio on accommodating sequences. Consequently, the accommodating function is a true gen-

[1] In that paper and a preliminary version of the current paper [10], this competitive ratio on accommodating sequences was called the accommodating ratio. The change is made here for consistency with common practice in the field.

eralization of the competitive ratio as well as the competitive ratio on accommodating sequences.

In addition to giving rise to new interesting algorithmic and analytical problems, which we have only begun investigating, this function, compared to just one ratio, contains more information about the on-line algorithms. For some problems, this information gives a more realistic impression of the algorithm than the competitive ratio does. Additionally, this information can be exploited in new ways. The shape of the function, for instance, can be used to warn against critical scenarios, where the performance of the on-line algorithm compared to the off-line can suddenly drop rapidly when more sequences are allowed.

In the next section, we formally define the accommodating function. In the following sections, the accommodating functions for three specific on-line problems are investigated: a variant of bin-packing in which the goal is to maximize the number of items put in n bins, the seat reservation problem, and the problem of optimizing total flow time when preemption is allowed.

In section 3, where we consider the variant of bin-packing, we consider two specific algorithms, First-Fit and Worst-Fit. We show that although First-Fit performs worse than Worst-Fit with respect to the competitive ratio, it performs better with respect to the competitive ratio on accommodating sequences. Thus, the choice as to which algorithm to use depends on which ratio is more relevant in a specific situation. This would depend on the actual distribution of request sequences and on the accommodating functions for the algorithms.

2. The accommodating function. Consider an on-line problem with a fixed amount of resources n. For a *maximization problem*, $\mathbb{A}(I)$ is the *value* of running the on-line algorithm $\mathbb{A}$ on I, and $\mathrm{OPT}(I)$ is the *maximum value* that can be achieved on I by an optimal off-line algorithm, OPT.

For a *minimization* problem, $\mathbb{A}(I)$ is a *cost* and $\mathrm{OPT}(I)$ is the *minimum cost* which can be achieved.

$\mathbb{A}$ and OPT use the same amount of resources, n. For a problem with some limited resource, $\mathrm{OPT}_m(\mathbb{A}_m)$ denotes the value/cost of an optimal off-line algorithm (the on-line algorithm) when the amount m of the limited resource is available.

DEFINITION 2.1. *Let P be an on-line problem with a fixed amount n of resources. For any $\alpha > 0$, an input sequence I to the problem P is said to be an* α-sequence *if $OPT_{\alpha n}(I) = OPT_{n'}(I)$ for all $n' \geq \alpha n$. 1-sequences are also called* accommodating sequences.

If an input sequence is an α-sequence, then an optimal off-line algorithm does not benefit from having more than the amount αn of resources. For maximization problems, this will often mean that the optimal off-line algorithm could have fully granted all requests with the amount αn of resources. If an input sequence is an accommodating sequence, then an optimal off-line algorithm does not benefit from having more resources than the amount already available.

For a maximization problem, the algorithm $\mathbb{A}$ is *c-competitive on α-sequences* if $c \leq 1$, and for every n and every α-sequence I, $\mathbb{A}_n(I) \geq c \cdot \mathrm{OPT}_n(I) - b$, where b is a fixed constant for the given problem, and, thus, independent of I.

Let $\mathcal{C}^{\alpha}_{\mathbb{A}} = \{c \mid \mathbb{A} \text{ is } c\text{-competitive on } \alpha\text{-sequences}\}$. The *accommodating function* $\mathcal{A}$ is defined as $\mathcal{A}_{\mathbb{A}}(\alpha) = \sup \mathcal{C}^{\alpha}_{\mathbb{A}}$.

For a minimization problem, $\mathbb{A}$ is c-competitive on α-sequences if $c \geq 1$ and for every n and every α-sequence I, $\mathbb{A}_n(I) \leq c \cdot \mathrm{OPT}_n(I) + b$, and the accommodating function is defined as $\mathcal{A}_{\mathbb{A}}(\alpha) = \inf \mathcal{C}^{\alpha}_{\mathbb{A}}$.

With this definition, the competitive ratio on accommodating sequences from [9] is $\mathcal{A}(1)$ and the competitive ratio is $\lim_{\alpha\to\infty}\mathcal{A}(\alpha)$. In this paper, we consider only $\alpha \geq 1$. In Figure 2.1, these relationships are depicted using a hypothetical example for a maximization problem.

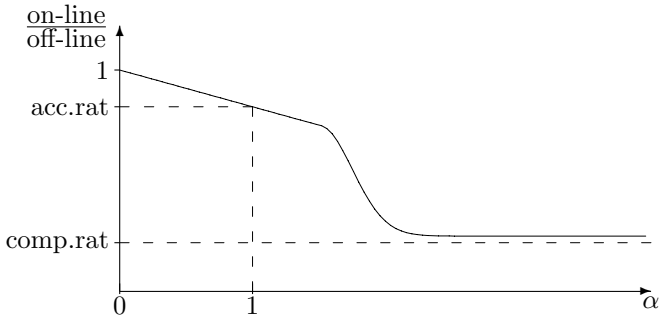


FIG. 2.1. *A typical accommodating function for a maximization problem.*

The extra information contained in the accommodating function compared with the competitive ratio can be used in different ways. If the user knows that estimates of required resources cannot be off by more than a factor three, for instance, then $\mathcal{A}(3)$ is a bound for the problem and thereby a better guarantee than the bound given by the competitive ratio. The shape of the function is also of interest. Intervals where the function is very steep are critical, since there earnings, compared to the optimal earnings, drop rapidly. Thus, the user misses out on business. Therefore, the function can warn against algorithms with unfortunate behavior, and, if such an algorithm must be used, the function can be used to locate resource critical areas.

3. Fair bin packing. Consider the following bin packing problem: Let n be the number of bins, all of size k. Given a sequence of integer-sized items of size at most k, the objective is to maximize the total number of items in these bins. This problem has been studied in the off-line setting, starting in [14], and its applicability to processor and storage allocation is discussed in [13]. (For surveys on Bin Packing, see [12, 15].) The problem we are considering is on-line, so the requests occur in a definite order. We require the packing to be *fair*; that is, an item can be rejected only if it cannot fit in any bin at the time when it is given. We refer to the problem as *Fair Bin Packing*.[2] Notice that the fairness criterion is a part of the problem specification. Thus, even though the optimal off-line algorithm knows the whole sequence of requests in advance, it must process the requests in the same order as the on-line algorithm and do so fairly.

In this problem, for a given α, we consider only sequences which could be packed in αn bins by an optimal off-line algorithm.

3.1. Summary of results. The following six theorems summarize our results for deterministic algorithms for Fair Bin Packing. Note that since this is a maximization problem, lower bounds are obtained by proving a bound on the worst-case behavior of algorithms, and upper bounds are obtained by giving adversary arguments.

[2] In [10], where a preliminary version of some of these results was presented, the same problem was referred to as Unit Price Bin Packing.

THEOREM 3.1. *For any Fair Bin Packing algorithm, the accommodating function is at most*

$$\mathcal{A}(\alpha) \leq \begin{cases} \frac{6}{7} & : \quad \alpha = 1, \\ \frac{2}{2+(\alpha-1)(k-2)} & : \quad 1 < \alpha \leq \frac{5}{4}, \\ \frac{8}{6+k} & : \quad \alpha > \frac{5}{4} \end{cases}$$

for $k \geq 3$.

THEOREM 3.2. *For any Fair Bin Packing algorithm, the accommodating function is at least*

$$\mathcal{A}(\alpha) \geq \begin{cases} \frac{1}{2} & : \quad \alpha = 1, \\ \frac{\alpha}{\max\{1+(\alpha-1)k, 2+(\alpha-1)\frac{k}{2}\}} & : \quad 1 < \alpha < 2, \\ \frac{2-\frac{1}{k}}{k} & : \quad \alpha \geq 2 \end{cases}$$

for $k \geq 3$.

The bounds presented in the previous two theorems are depicted in Figure 3.1.

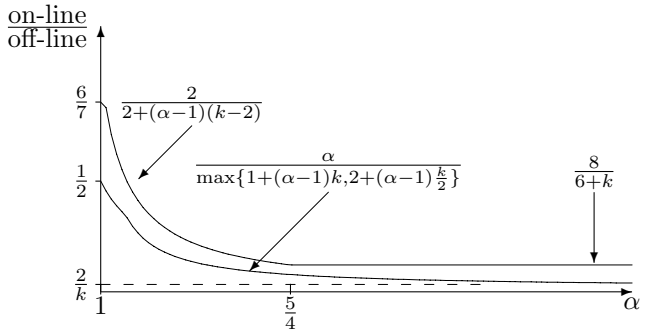


FIG. 3.1. *General upper and lower bounds on the accommodating function for Fair Bin Packing, drawn for* $k = 60$.

The specific algorithms we consider are First-Fit and Worst-Fit. First-Fit places an item in the lowest numbered bin in which it fits, while Worst-Fit places an item in one of the bins which are least full.

THEOREM 3.3. *An upper bound on the accommodating function for First-Fit is*

$$\mathcal{A}_{FF}(\alpha) \leq \begin{cases} \frac{7}{11} & : \quad \alpha = 1, \\ \frac{\alpha}{1+(\alpha-1)(k-1)} & : \quad 1 < \alpha < 2, \\ \frac{2-\frac{1}{k}}{k} & : \quad \alpha \geq 2. \end{cases}$$

The general lower bounds from Theorem 3.2 apply to all algorithms, including First-Fit. A better lower bound on the competitive ratio on accommodating sequences can be shown.

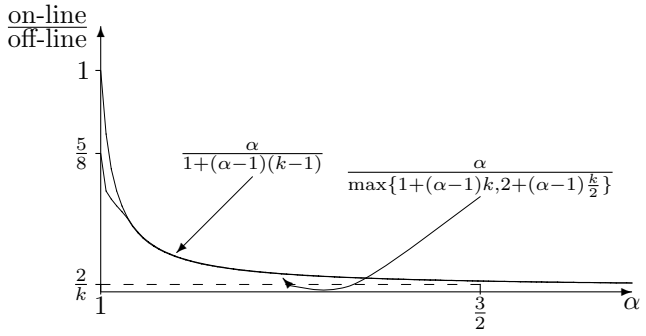


FIG. 3.2. *Upper and lower bounds on the accommodating function for First-Fit, drawn for* $k = 60$.

THEOREM 3.4. *For First-Fit, the competitive ratio on accommodating sequences is at least*

$$\mathcal{A}_{FF}(1) \geq \frac{5}{8}.$$

The bounds on First-Fit presented in the previous three theorems are depicted in Figure 3.2.

THEOREM 3.5. *An upper bound on the accommodating function for Worst-Fit is*

$$\mathcal{A}_{WF}(\alpha) \leq \begin{cases} \frac{1}{2-\frac{1}{k}} & : \quad \alpha = 1, \\ \frac{3-\frac{1}{n}}{2-\frac{1}{n}+(\alpha-1)k} & : \quad 1 < \alpha < 1 + \frac{1}{n}\left\lceil \frac{(n-1)(k-n)+(n-1)}{k} \right\rceil, \\ \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k} & : \quad \alpha \geq 1 + \frac{1}{n}\left\lceil \frac{(n-1)(k-n)+(n-1)}{k} \right\rceil,\ k \geq \beta n,\ \beta \geq 1. \end{cases}$$

The general lower bounds from Theorem 3.2 apply to all algorithms, including Worst-Fit. A better lower bound on the competitive ratio can be shown.

THEOREM 3.6. *The competitive ratio of Worst-Fit is at least* $\frac{3}{2+k}$.

3.2. Upper bounds for all deterministic algorithms. Here we prove bounds which apply to all deterministic algorithms for Fair Bin Packing, beginning with an upper bound on the accommodating function. All of the results in this subsection also hold if one relaxes the restriction that all items must be integer-sized to a restriction that the bins have unit size and the smallest item has size $1/k$.

THEOREM 3.7. *For any Fair Bin Packing algorithm, if* $\alpha \leq \frac{5}{4}$ *and* $k \geq 3$, *then* $\mathcal{A}(\alpha) \leq \frac{2}{2+(\alpha-1)(k-2)}$.

Proof. Consider an arbitrary fair on-line algorithm $\mathbb{A}$. An adversary can give $\mathbb{A}$ the following request sequence, divided into three phases. Phase 1 consists of n small items of unit size. Phase 2 consists of items, one for each bin which $\mathbb{A}$ did not fill completely with size equal to the empty space in that bin, sorted in decreasing order. After these are given, $\mathbb{A}$ has filled all bins completely and so must reject the items in phase 3, which consists of $(\alpha n - n)k$ items of unit size. Let q denote the number of empty bins in $\mathbb{A}$'s configuration after the first phase.

In the case where $q < \frac{n}{4}$, we know that $\mathbb{A}$ has at least $n - 2q \geq 2(\alpha n - n)$ bins with exactly one item after phase 1. OPT can arrange the items from phase 1 such that half of the bins contain two items and half contain no items. In the second request phase, there are at least $2(\alpha n - n)$ items of size $k - 1$. OPT rejects at least $\alpha n - n$,

leaving room for at least $(\alpha n - n)(k-1)$ of the unit size items from phase 3. This gives a total gain of at least $(k-2)(\alpha n - n)$, and the performance ratio is at most $\frac{2n}{2n+(\alpha n-n)(k-2)} = \frac{2}{2+(\alpha-1)(k-2)}$.

In the case where $q \geq \frac{n}{4}$, we know that $\mathbb{A}$ has at least $\alpha n - n$ empty bins after phase 1. OPT places each of the items from phase 1 in a different bin. This gives a performance ratio of at most $\frac{2n}{2n+(\alpha n-n)(k-1)} = \frac{2}{2+(\alpha-1)(k-1)}$, since OPT rejects $\alpha n - n$ items of size k. □

Using the value $\alpha = \frac{5}{4}$ in the above theorem gives an upper bound on the competitive ratio.

COROLLARY 3.8. *If $k \geq 3$, no Fair Bin Packing algorithm is more than $\frac{8}{6+k}$-competitive.*

The next theorem improves the bound on the accommodating function for $\alpha = 1$.

THEOREM 3.9. *For $k \geq 7$, any Fair Bin Packing algorithm has a competitive ratio on accommodating sequences of at most $\frac{6}{7}$.*

Proof. Consider an arbitrary fair on-line algorithm $\mathbb{A}$ and assume n is even. An adversary first gives n items of size $\lceil \frac{k}{2} \rceil - 1$. Since $k \geq 7$, $\mathbb{A}$ has packed at most two items per bin. Let q denote the number of empty bins in $\mathbb{A}$'s packing. In the case where $q < \frac{2n}{7}$, the off-line algorithm can pack these n requests in the first $\frac{n}{2}$ bins. Now the adversary can give $\frac{n}{2}$ long requests of size k. The performance ratio is then $\frac{n+q}{n+\frac{n}{2}} = \frac{2n+2q}{3n} < \frac{6}{7}$.

In the case where $q \geq \frac{2n}{7}$, observe that $\mathbb{A}$ has q bins with exactly two items. Let the off-line algorithm place one item in each bin. In this case, the adversary can now give n requests of size $\lfloor \frac{k}{2} \rfloor + 1$. The performance ratio is then $\frac{2n-q}{2n} \leq \frac{6}{7}$. □

3.3. Lower bounds for all deterministic algorithms. Now we prove lower bounds which apply to all deterministic algorithms for Fair Bin Packing. The first lower bound is on the competitive ratio. Fix an on-line algorithm $\mathbb{A}$ and consider the configuration of $\mathbb{A}$ after a given sequence I has been packed. Let e denote the maximal empty space in any bin. Since all items are integer-sized, if the bins are not all full, $e \geq 1$.

LEMMA 3.10. *If $e \geq 1$, the performance ratio of $\mathbb{A}$ is at least $\frac{3}{2+k}$, for $k \geq 3$.*

Proof. Let V denote the volume of the first n requests, all of which must be accepted by $\mathbb{A}$. Note that $n \leq V \leq nk$. All items of size at most e have been accepted by $\mathbb{A}$ due to the fairness criterion. Let M count the number of such small items which arrive after the first n requests.

In the case where $e = 1$, M counts the number of unit size items. As argued above, $\mathbb{A}$ must accept the first n items, plus the M unit size items, and together they have volume $V + M$. Since $\mathbb{A}$ fills every bin with volume at least $k - 1$, these items account for all of what $\mathbb{A}$ places in at most $\lfloor \frac{V+M}{k-1} \rfloor$ bins. Thus, it must accept at least $n - \lfloor \frac{V+M}{k-1} \rfloor$ additional items. The optimal off-line algorithm OPT must also accept the first n items and the M unit size items. For other items, it has $nk - V - M$ space remaining. All other items in the request sequence have size at least 2, so OPT accepts at most $\frac{nk-V-M}{2}$ additional items. Hence, the performance ratio is at least

$$\frac{n + M + n - \lfloor \frac{V+M}{k-1} \rfloor}{n + M + \frac{nk-V-M}{2}} \geq \frac{n + M + n - \frac{V+M}{k-1}}{n + M + \frac{nk-V-M}{2}},$$

defined for $n \leq V \leq nk$ and $0 \leq M \leq nk - V$. By finding the minimum with respect

to M, it can be shown that the following is a lower bound for the performance ratio:

$$\frac{2n - \frac{V}{k-1}}{n + \frac{nk-V}{2}} \geq \frac{2n - \frac{n}{k-1}}{n + \frac{nk-n}{2}} = \frac{4k-6}{k^2-1}.$$

The last inequality is obtained by finding the minimum with respect to V.

In the case where $e \geq 2$, the performance ratio is at least

$$\begin{aligned}\frac{n + M + (n - \lfloor \frac{V+M}{k-e} \rfloor)}{n + M + \frac{nk-V-M}{e+1}} &\geq \frac{n+M}{n + M + \frac{nk-n-M}{e+1}} \\ &= \frac{(e+1)(M+n)}{en + eM + nk} \\ &\geq \frac{(e+1)n}{en+nk} \geq \frac{3}{2+k}.\end{aligned}$$

For $k \geq 3$, the case $e \geq 2$ gives the smallest value. □

THEOREM 3.11. *The competitive ratio for any Fair Bin Packing algorithm is at least* $\frac{2-\frac{1}{k}}{k}$ *when* $k \geq 3$.

Proof. Consider an algorithm $\mathbb{A}$ for Fair Bin Packing. When $k \geq 3$, the ratio $\frac{3}{2+k}$ from Lemma 3.10 is larger than $\frac{2-\frac{1}{k}}{k}$. Thus, it suffices to consider the case where $e = 0$. Again, let V denote the volume of the first n requests, all of which must be accepted by $\mathbb{A}$, since it is fair. Since $e = 0$, every bin has been filled up by the on-line algorithm, but the first n requests filled at most $\lfloor \frac{V}{k} \rfloor$ bins. Thus, $\mathbb{A}$ accepted at least $n + (n - \lfloor \frac{V}{k} \rfloor)$ items. Since all items have size at least 1, OPT accepts at most $n + (nk - V)$ items, giving a performance ratio of at least $\frac{2n - \lfloor \frac{V}{k} \rfloor}{n+nk-V} \geq \frac{2n-\frac{V}{k}}{n+nk-V} \geq \frac{2n-\frac{n}{k}}{nk} = \frac{2-\frac{1}{k}}{k}$. □

Next we give a lower bound on the competitive ratio on accommodating sequences, $\mathcal{A}(1)$. Note that the result does not depend on the items being integer-sized.

THEOREM 3.12. *Let I be an input sequence which can be accommodated within n bins. The performance ratio is greater than $\frac{1}{2}$ for any Fair Bin Packing algorithm.*

Proof. Let A denote the set of items accepted by the on-line algorithm, and let R denote the set of items rejected. The set R could be empty, but then the on-line algorithm would have accommodated every request, giving a performance ratio of 1. If R is nonempty, at least n items are accepted. Since the performance ratio is $\frac{|A|}{|A|+|R|}$, it is enough to show that $|R| < n$. Let e denote the maximal empty space in any bin. Every item in R has size greater than e. If $|R| \geq n$, then the volume of R is greater than ne. However, this contradicts the fact that the total empty space is at most ne, and the entire sequence can be packed in n bins. □

The final general lower bound is on the accommodating function.

THEOREM 3.13. *For any Fair Bin Packing algorithm, the accommodating function can be bounded by* $\mathcal{A}(\alpha) \geq \frac{\alpha}{\max\{1+(\alpha-1)k, 2+(\alpha-1)\frac{k}{2}\}}$ *for* $1 < \alpha < 2$ *and* $k \geq 4$.

Proof. Consider a worst-case sequence I for an on-line algorithm $\mathbb{A}$. We may assume that I contains no item which is rejected by both $\mathbb{A}$ and OPT, since such an item has no influence on the ratio.

Again, let A denote the set of items accepted by the on-line algorithm $\mathbb{A}$, and let R denote the set of items rejected. Let $\rho(I)$ denote the least number of bins in which the sequence can be packed and define $l = \rho(I) - n$. If $l = 0$, we can use the result from Theorem 3.12. Assume that $l \geq 1$ and note that l is a lower bound on the

number of items rejected by OPT. The first n items are accepted by both algorithms. Since, by assumption, the on-line algorithm accepts all those items OPT rejects, it must accept at least $n + l$ items. Thus, $|A| \geq n + l$. Again, let e denote the size of the maximal empty space in any bin. The proof is divided into two cases depending on e.

If $e = 0$, all n bins have been filled by the on-line algorithm, so the number of rejected items is at most lk.

In the case where $e \geq 1$, an upper bound on the number of items rejected is $|R| \leq \frac{1}{e+1}(en + lk)$ using the same arguments as in the previous theorem. We can now bound the accommodating function: $\mathcal{A}(\alpha) \geq \frac{|A|}{|A|+|R|-l} \geq \frac{n+l}{n+l+\max\{lk, \frac{en}{e+1}+l\frac{k}{e+1}\}-l}$ $\geq \frac{n+(\alpha n-n)}{n+\max\{(\alpha n-n)k, n+(\alpha n-n)\frac{k}{2}\}} = \frac{\alpha}{1+\max\{(\alpha-1)k, 1+(\alpha-1)\frac{k}{2}\}}$, for $1 < \alpha < 2$. □

3.4. Separation of First-Fit and Worst-Fit using the accommodating function. In this section, we prove that the accommodating function provides extra information by showing that the best choice between two different algorithms for the same natural problem (Fair Bin Packing) cannot be made based on either the competitive ratio or the competitive ratio on accommodating sequences alone.

The two specific algorithms that we consider are First-Fit and Worst-Fit. First-Fit places an item in the lowest numbered bin in which it fits, while Worst-Fit places an item in a bin which is least full. We show that Worst-Fit has a better competitive ratio ($r_{\mathrm{WF}} \geq \frac{3}{2+k}$) than First-Fit ($r_{\mathrm{FF}} \leq \frac{2-\frac{1}{k}}{k}$), while First-Fit has a better competitive ratio on accommodating sequences ($\mathcal{A}_{\mathrm{FF}}(1) \geq \frac{5}{8}$) than Worst-Fit ($\mathcal{A}_{\mathrm{WF}}(1) \leq \frac{1}{2-\frac{1}{k}}$). All of the results proven in this section for First-Fit also hold for Best-Fit. Best-Fit is the algorithm which places an item in the most full among the bins where it fits. If the "most full" is not unique, then Best-Fit chooses the first among those "most full" bins.

Worst-Fit's competitive ratio on accommodating sequences. First, we prove an upper bound on Worst-Fit's competitive ratio on accommodating sequences.

THEOREM 3.14. *Worst-Fit has a competitive ratio on accommodating sequences of at most* $\frac{1}{2-\frac{1}{k}}$ *for all* k.

Proof. Assume that n is divisible by k. An adversary can give the following request sequence:

1. n items of unit size;
2. $n - \frac{n}{k}$ items of size k.

Worst-Fit places one small item in each bin and must reject all the following items. The optimal algorithm behaves like First-Fit and accepts all items giving the following performance ratio: $\frac{n}{n+n-\frac{n}{k}} = \frac{1}{2-\frac{1}{k}}$. □

First-Fit's and Best-Fit's competitive ratios on accommodating sequences. Now we show that First-Fit's and Best-Fit's competitive ratios on accommodating sequences are at least $\frac{5}{8}$, which is strictly greater than Worst-Fit's competitive ratio on accommodating sequences, when $k > 2$. Thus, according to this performance measure, First-Fit and Best-Fit are better algorithms than Worst-Fit. The proof which shows this lower bound on Best-Fit's competitive ratio on accommodating sequences is essentially the same as the one for First-Fit, after the following two lemmas are proven giving lower bounds on the sizes of the first items Best-Fit places in a new bin. We let $A(\mathbb{A}, I)$ denote the set of items accepted by an on-line algorithm $\mathbb{A}$, and we let $R(\mathbb{A}, I)$ denote the set of items it rejects. The first lemma

shows that if Best-Fit packs items so that some bin is no more than half full, then the performance ratio is at least $2/3 > 5/8$, so we can safely ignore such sequences. In fact, this result holds for First-Fit, too, but we need only to apply it for Best-Fit.

LEMMA 3.15. *Suppose that OPT accepts all items in some request sequence I. Suppose further that in Best-Fit's packing of the sequence I, there is some bin which is no more than half full. Then Best-Fit accepts at least $2/3$ of the items in I.*

Proof. If Best-Fit leaves some bin b no more than half full, then every item in $R(\mathrm{BF}, I)$ must have size strictly greater than $k/2$. Suppose that among the items which Best-Fit accepts, there are x not in b which it places in bins alone. All of these x items must be larger than the empty space in bin b. Otherwise, they would have been placed in bin b or some of the contents in bin b would have been placed on top of them. Thus, these x items also have size strictly greater than $k/2$. We consider two cases:

Case 1. Bin b contains more than one item.

Case 2. Bin b contains exactly one item.

In the first case, since OPT accepts all of the items in I, there cannot be more than n items of size strictly greater than $k/2$, so $x + |R(\mathrm{BF}, I)| \leq n$. If Best-Fit rejects any items at all, it has at least two items in all except x bins, so the number of items accepted by Best-Fit, $A(\mathrm{BF}, I) \geq 2n - x$, and the total number of items in I is $A(\mathrm{BF}, I) + R(\mathrm{BF}, I)$. Thus, Best-Fit's performance ratio is at least $\frac{A(\mathrm{BF},I)}{A(\mathrm{BF},I)+R(\mathrm{BF},I)} \geq \frac{2n-x}{2n-x+R(\mathrm{BF},I)} \geq \frac{2n-x}{3n-2x} \geq \frac{2}{3}$.

The second case is argued similarly. Since b has only one item, it cannot fit with any of the x items which are placed alone or with any of the rejected items, so $1 + x + |R(\mathrm{BF}, I)| \leq n$. Another difference is that there is now an additional bin, b, which Best-Fit does not give at least two items. In this case, $A(\mathrm{BF}, I) \geq 2n - x - 1$. Thus, Best-Fit's performance ratio is at least $\frac{A(\mathrm{BF},I)}{A(\mathrm{BF},I)+R(\mathrm{BF},I)} \geq \frac{2n-x-1}{2n-x-1+R(\mathrm{BF},I)} \geq \frac{2n-x-1}{3n-2x-2} \geq \frac{2}{3}$. □

When considering Best-Fit in the following, we look only at sequences which Best-Fit packs such that all bins are more than half full. The next lemma shows that if Best-Fit packs more than one item in some bin, b, then either at least one of them has size greater than $k/2$ or at least two of them have size greater than the final empty space in any bin which was first used before bin b. We call a bin which was first used before that bin an "earlier" bin. This lemma can be seen to follow from Claim 2.2.2 in [21], but a direct proof is included below for completeness.

LEMMA 3.16. *Suppose that OPT accepts all items in some request sequence I, and that in Best-Fit's packing of the sequence I, all bins are more than half full. Then every bin contains either an item of size greater than $k/2$ or at least two items of size greater than the empty space in any earlier bin.*

Proof. Best-Fit places an item in an empty bin only when it will not fit in any earlier bin. Suppose that Best-Fit puts the first item in bin b at time t. This first item must be larger than the empty space in any earlier bin at time t and thus larger than the final empty space in any earlier bin. Suppose that this first item has size no more than $k/2$. Then, by the assumption that all bins are eventually more than half full, Best-Fit must put some other item x in bin b. In addition, all of the earlier bins must have been more than half full at time t. Thus, when this second item x is put in bin b, all earlier bins were more full than b, so this x was too large to fit in them. Thus, if the first item in a bin has size no more than $k/2$, the first two both have size larger than the final empty space in any earlier bin. □

Given a request sequence $I = \langle s_1, s_2, \ldots, s_t \rangle$, where s_i is the size of item i, we can

represent the final configuration of an algorithm $\mathbb{A}$ by $\mathit{conf}(\mathbb{A}, I) = \langle S_1, S_2, \ldots, S_n \rangle$, a list of n multisets, where the multiset S_j contains the sizes of the items in bin j. In order to prove a lower bound on First-Fit's or Best-Fit's competitive ratio on accommodating sequences, we compare $\mathit{conf}(\mathrm{FF}, I)$ or $\mathit{conf}(\mathrm{BF}, I)$ to $\mathit{conf}(\mathrm{OPT}, I)$. In the following, we write only First-Fit or FF, but everything applies to Best-Fit as well. The sizes which appear in $\mathit{conf}(\mathrm{FF}, I)$ will correspond to items in $A(\mathrm{FF}, I)$, but since we are assuming that OPT can accommodate all t items from I, there will be t items in $\mathit{conf}(\mathrm{OPT}, I)$. Consider rearranging the items in $\mathit{conf}(\mathrm{FF}, I)$ so that the items from $A(\mathrm{FF}, I)$ are placed in exactly the same bins as they are in $\mathit{conf}(\mathrm{OPT}, I)$, creating a new configuration $\mathit{good}(I) = \langle S'_1, S'_2, \ldots, S'_n \rangle$, which also contains exactly those items in $A(\mathrm{FF}, I)$.

First, we prove a lemma which relates the number of moves necessary for changing from $\mathit{conf}(\mathrm{FF}, I)$ to $\mathit{good}(I)$ to the number of items from I which First-Fit rejects. In fact, this lemma holds for any algorithm for Fair Bin Packing, not just for First-Fit and Best-Fit.

LEMMA 3.17. *For any request sequence I which could be accommodated by OPT, the minimal number of items which have to be moved, in order to change from $\mathit{conf}(\mathbb{A}, I)$ to $\mathit{good}(I)$, is an upper bound on the number of items from I which the algorithm $\mathbb{A}$ rejects.*

Proof. Consider any request sequence I and any algorithm $\mathbb{A}$ for Fair Bin Packing. The process of changing from $\mathit{conf}(\mathbb{A}, I)$ to $\mathit{good}(I)$ involves moving items out of some bins and into others. Those bins that become less full after this rearrangement may have space for one or more items from $R(\mathbb{A}, I)$. Let s denote the size of the smallest item in $R(\mathbb{A}, I)$ and write the size of item i as $s_i = q_i s + r_i$, where $0 \le r_i \le s - 1$, and the amount of empty space in bin j as $e_j = k - \sum_{s_i \in S_j} s_i$. The new empty space in bin j is $e'_j = e_j + \sum_{s_i \in S_j \setminus S'_j} s_i - \sum_{s_i \in S'_j \setminus S_j} s_i$. When item i moves from bin j to bin j', the value $s_i = q_i s + r_i$ is added to the empty space in bin j and subtracted from the empty space in bin j'. Therefore if we sum, over all bins in the configuration $\mathit{good}(I)$, the number of items of size s which could fit in their empty space, the value q_i is added to bin j and subtracted from bin j', giving a net gain of zero. Thus, the only real contribution comes from the original empty space, e_j, and the values r_i. Let $g(j)$ denote the net gain attributed to bin j. Then

$$\begin{aligned} g(j) &\le \lfloor \tfrac{1}{s}(e_j + \textstyle\sum_{s_i \in S_j \setminus S'_j} r_i) \rfloor \\ &\le \lfloor \tfrac{1}{s}(s - 1 + |S_j \setminus S'_j|(s-1)) \rfloor \\ &= \lfloor \tfrac{s-1}{s}(|S_j \setminus S'_j| + 1) \rfloor \\ &\le |S_j \setminus S'_j|, \end{aligned}$$

which is the number of items moved out of bin j. Since all of the items from $R(\mathbb{A}, I)$ fit in the empty space available in $\mathit{good}(I)$, the total number of items in $R(\mathbb{A}, I)$ is at most the total net gain after rearranging. This net gain is $\sum_{j=1}^{n} g(j) \le \sum_{j=1}^{n} |S_j \setminus S'_j|$, which is the total number of items moved from one bin to another to get from $\mathit{conf}(\mathbb{A}, I)$ to $\mathit{good}(I)$. ∎

Note that the ordering of the sets in $\mathit{conf}(\mathrm{OPT}, I)$ is irrelevant, so we may order them in any way. The above result holds for the minimum number of moves over all of these arrangements.

In order to prove a lower bound on the competitive ratio on accommodating sequences for First-Fit, we prove a lower bound on $|A(\mathrm{FF}, I)|$, which holds for all I. This is done by proving a lower bound on the number of items which do not move

when changing from *conf*(FF, I) to *good*(I). First, we prove a general result which applies to all algorithms for Fair Bin Packing, and then we use this result and the special properties of First-Fit and Best-Fit to prove the lower bound of 5/8.

Define $B(\mathbb{A}, I)$ to be the set of items in I of size greater than $k/2$ which $\mathbb{A}$ accepts. Some of the items moved, in changing from *conf*($\mathbb{A}, I$) to *good*(I), will be moved to be in the same bins as the items in $B(\mathbb{A}, I)$ or moved out of some bin containing an item from $B(\mathbb{A}, I)$. Suppose that these moves are done first. Let $O(\mathbb{A}, I)$ denote the items in $A(\mathbb{A}, I) \setminus B(\mathbb{A}, I)$ which have not moved yet. We use Vol[$\cdot$] to denote the volume of a collection of items; e.g., Vol[$O(\mathbb{A}, I)$] is the total volume of all items in $O(\mathbb{A}, I)$.

LEMMA 3.18. *There must be an ordering of the sets in conf*(OPT, I) *in which at least* $\frac{Vol[O(\mathbb{A},I)]}{k}$ *of the items in* $A(\mathbb{A}, I) \setminus B(\mathbb{A}, I)$ *are not moved.*

Proof. Consider the bipartite graph $G = ((X, Y), E)$ defined as follows:

1. For each bin b, there are two vertices $x_b \in X$ and $y_b \in Y$.
2. For each item in $O(\mathbb{A}, I)$, there is one edge. If the item is in bin i in *conf*(A, I) and in bin j in *good*(I), the corresponding edge is (x_i, y_j).

A matching in this graph G corresponds to a partial renumbering of the bins, and the edges in the matching correspond to items which have not been moved. A well-known result due to König [26] (see [6] for instance) states that the size of a maximum matching in a bipartite graph is equal to the size of a minimum vertex cover. However, a vertex cover of G corresponds to a set of bins (possibly with some from *conf*(A, I) and some from *good*(I)), which contain all of the items in $O(\mathbb{A}, I)$. Thus, the number of items which have not been moved is at least the minimum number of bins needed to contain all the items in $O(\mathbb{A}, I)$. The value $\frac{\text{Vol}[O(\mathbb{A},I)]}{k}$ is a lower bound on this number of bins. □

THEOREM 3.19. *The competitive ratios on accommodating sequences for First-Fit and Best-Fit are at least* $\frac{5}{8}$.

Proof. Within this proof, First-Fit and Best-Fit are the only on-line algorithms considered, and the sequence I will be fixed, but arbitrary, subject to the restriction that all requests would have been accepted by OPT (and for Best-Fit subject to the further restriction that, given I, Best-Fit packs all bins to more than half full). We use the following short hand notation for legibility: A is used for A(FF, I) or A(BF, I), B is used for B(FF, I) or B(BF, I), R is used for R(FF, I) or R(BF, I), and V is used for Vol[O(FF, I)] or Vol[O(BF, I)].

No two items from B can be placed in one bin, since they all have size greater than $k/2$. Therefore when counting moves as in Lemma 3.17, the items from B can be assumed not to have moved, since the bins can be reordered such that these items stay in their original bins.

According to Lemma 3.17, in order to get from *conf*(BF, I) to *good*(I), at least one item must move for each item in R. All of these items must be in A, along with the items which are not moved. Lemma 3.18 shows that at least $\frac{V}{k}$ of these items in $A \setminus B$ are not moved, so $|A| \geq |R| + |B| + \frac{V}{k}$.

We consider two cases based on the value of s, the size of the smallest item in R. In each case, we first prove a lower bound on V and then use that to prove a lower bound on the performance ratio.

Case $s > \frac{k}{3}$. Let T denote those bins which do not contain items of size greater than $k/2$, and let e_i denote the size of the empty space in the ith bin in T. For First-Fit, it is clear that, for $2 \leq i \leq n - |B|$, there are at least two items in bin i in T which have size greater than e_{i-1}. For Best-Fit, since we assume that $e_i < k/2$ for all bins i in T, this follows from Lemma 3.16. The empty space in those bins containing

items from B is at most $s-1$, since no items from R could fit in them. Thus, the total volume of those items which are moved to be in the same bins as the items in B is no more than $|B|(s-1)$. The first bin in T has volume $k-e_1$. Hence, the volume $V > (k-e_1)+2(\sum_{i=1}^{n-|B|-1} e_i)-|B|(s-1)$. The empty space above the first $n-|B|-1$ bins in T, plus the empty space above the other bins, must be large enough to contain all of the rejected items. Thus, $\sum_{i=1}^{n-|B|-1} e_i \geq |R|s-(|B|+1)(s-1)$. Hence,

$$\begin{aligned} V &> (k-e_1)+2(\textstyle\sum_{i=1}^{n-|B|-1} e_i)-|B|(s-1) \\ &\geq k-(s-1)+2|R|s-2(|B|+1)(s-1)-|B|(s-1) \\ &= k-3s+3+2|R|s-3|B|s+3|B| \\ &\geq k-3s+2|R|s-3|B|s. \end{aligned}$$

Note that either the performance ratio is greater than $\frac{5}{8}$ or $\frac{5}{8} \geq \frac{|A|}{|A|+|R|} \geq \frac{|R|+|B|}{2|R|+|B|}$, so $2|R| \geq 3|B|$. Thus, for $s \geq \frac{k}{3}$, this lower bound for V is at least the value at $s=\frac{k}{3}$. Hence, $V \geq \frac{2|R|k}{3}-|B|k$. This volume V requires at least $\frac{V}{k} \geq \frac{2}{3}|R|-|B|$ bins.

The performance ratio is then $\frac{|A|}{|A|+|R|} \geq \frac{\frac{5}{3}|R|}{\frac{8}{3}|R|} = \frac{5}{8}$.

Case $s \leq \frac{k}{3}$. Each of the bins must contain items with size adding up to at least $k-s+1$, or neither First-Fit nor Best-Fit would have rejected an item of size s. As in the previous case, the total volume of those items which are moved to be in the same bins as the items in B is no more than $|B|(s-1)$. Thus, the volume $V \geq (n-|B|)(k-s+1)-|B|(s-1) = nk-ns+n-|B|k$, and $\frac{V}{k} \geq n-\frac{ns}{k}-|B|$. For $s \leq \frac{k}{3}$, this is minimized when $s=\frac{k}{3}$, giving $\frac{V}{k} \geq \frac{2n}{3}-|B|$. In the proof of Theorem 3.12, it was shown that $|R| < n$, so the performance ratio is at least $\frac{|A|}{|A|+|R|} \geq \frac{|R|+\frac{2n}{3}}{2|R|+\frac{2n}{3}} \geq \frac{\frac{5n}{3}}{\frac{8n}{3}} = \frac{5}{8}$.

Therefore, the competitive ratios on accommodating sequences for First-Fit and Best-Fit are at least $\frac{5}{8}$. □

We now show an upper bound which applies to both First-Fit's and Best-Fit's competitive ratios on accommodating sequences.

THEOREM 3.20. *For Fair Bin Packing, First-Fit's and Best-Fit's competitive ratios on accommodating sequences are at most* $\frac{7}{11}$.

Proof. Assume that n is divisible by 13 and that k is greater than 72 and divisible by 3. An adversary can give the following request sequence, divided into four phases:

1. $\frac{3n}{13}$ of size $\frac{k}{3}-6$;
2. $\frac{6n}{13}$ pairs, one of size $\frac{k}{3}-1$ followed by one of size $\frac{k}{3}+3$;
3. $\frac{6n}{13}$ of size $\frac{2k}{3}+1$;
4. $\frac{12n}{13}$ of size $\frac{k}{3}$.

First-Fit and Best-Fit will pack phase 1 in $\frac{n}{13}$ bins, with three items in each bin. The assumption that $k > 72$ ensures that four items from this phase cannot be packed together. From phase 2, First-Fit and Best-Fit will pack one pair in each bin using $\frac{6n}{13}$ bins. In phase 3, each item will be placed in its own bin, using the last $\frac{6n}{13}$ bins. There will be no space for items from phase 4.

OPT can pack one item from phase 1 with two of the items of size $\frac{k}{3}+3$ from phase 2, using a total of $\frac{3n}{13}$ bins for this. Then, it can place one item of size $\frac{k}{3}-1$ from phase 2 together with one item from phase 3, using a total of $\frac{6n}{13}$ for this. There are now $\frac{4n}{13}$ empty bins which can each hold three items from phase 4. The ratio is thus $\frac{3+12+6}{3+12+6+12} = \frac{7}{11}$. □

This bound has been improved later to $\frac{5}{8}$ [1], showing that the lower bound from Theorem 3.19 is tight.

First-Fit's competitive ratio. Turning to the competitive ratio, we show that according to this measure, Worst-Fit is the better algorithm. First, we prove an upper bound on First-Fit's competitive ratio. Note that this result also applies to Best-Fit.

THEOREM 3.21. *For Fair Bin Packing, First-Fit has a competitive ratio which is no more than* $\frac{2-\frac{1}{k}}{k}$*, when* k *divides* n*.*

Proof. An adversary gives the following request sequence, divided into three phases:

1. n items of unit size;
2. $n - \frac{n}{k}$ items of size k;
3. $n(k-1)$ items of unit size.

First-Fit accepts the first two phases of requests. The optimal algorithm places each of the first n items in a separate bin and accepts all requests in phases 1 and 3, giving the following performance ratio: $\frac{n+n-\frac{n}{k}}{nk} = \frac{2-\frac{1}{k}}{k}$. □

For arbitrary n and k, a very similar request sequence gives an upper bound on the competitive ratio for First-Fit of $\frac{2-\frac{1}{k}+\frac{1}{n}}{k}$. By Theorem 3.11, this result is tight.

Worst-Fit's competitive ratio. Finally, we prove a lower bound on Worst-Fit's competitive ratio, but first we prove a tight upper bound, since it can provide some intuition for the lower bound $r_{\mathrm{WF}} \geq \frac{3}{2+k}$. The request sequence used to prove this upper bound involves items with sizes dependent on n, the number of bins. However, it is relatively easy to prove an upper bound of $\frac{4}{k+2}$ using a sequence where the sizes only depend on k, rather than also on n.

THEOREM 3.22. *For* $k \geq \beta n$ *and* $\beta \geq 1$*, the competitive ratio for Worst-Fit is no more than* $\frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k}$.

Proof. An adversary can give the following request sequence with four phases:

1. $n-1$ items of size $n-1$;
2. $n-1$ items of size 1;
3. n items of size $k-(n-1)$;
4. $(n-1)(k-n)+(n-1)$ items of size 1.

After phase 2, Worst-Fit has free space of size $k-(n-1)$ in every bin, and all items from phase 3 must be accepted. OPT places items from phase 1 in separate bins, and each item from phase 2 on top of an item from phase 1. OPT will then accept only one item from phase 3, making space for all the unit size items from phase 4. The performance ratio is then

$$\begin{aligned}
&\frac{(n-1)+(n-1)+n}{(n-1)+(n-1)+1+(n-1)(k-n)+(n-1)} \\
&= \frac{3n-2}{3n-2+(n-1)(k-n)} \\
&= \frac{3n-3+1}{3n-3+1+(n-1)(k-n)} \\
&= \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+k-n} \\
&\leq \frac{3+\frac{1}{n-1}}{3+\frac{1}{n-1}+(1-\frac{1}{\beta})k}.
\end{aligned}$$

□

In order to compare this ratio to the lower bound of $\frac{3}{2+k}$, note that it can be made arbitrarily close to $\frac{3}{3+k}$ if n and β are made large enough.

We now move on to the lower bound. Consider any request sequence I for Fair Bin Packing. Throughout this section, we assume that there are no items in I which both Worst-Fit and OPT reject. This cannot affect the results since any such item could simply be removed from the request sequence without affecting the competitive ratio.

The upper bound can give some intuition for the lower bound. In order to allow OPT to accept many more items than Worst-Fit, the adversary must give some large items which Worst-Fit will accept but OPT will reject. In a worst case example, Worst-Fit packs none of these large items alone. Since OPT is fair, OPT must pack two items in some bins such that these bins have more contents than the Worst-Fit bins just before the large items are accepted. The second small item OPT packs in each bin cannot be large enough to cause a rejection alone. Thus for each large item accepted by Worst-Fit it will accept additionally two items, while OPT will accept at most k small items. Unfortunately, there are many possibilities for sequences, so it is necessary to argue that it is possible to make certain assumptions about them.

We define the following sets:

X is the set of items which both Worst-Fit and OPT accept.
Y is the set of items which Worst-Fit accepts, but OPT rejects.
Z is the set of items which Worst-Fit rejects, but OPT accepts.

Let $y_{last} \in Y$ be the last item from Y in the request sequence I, and let i be the bin where Worst-Fit places it.

We define the following additional sets:

$X^f \subseteq X$ are those items from X which appear before y_{last} in I.
$Z^f \subseteq Z$ are those items from Z which appear before y_{last} in I.

For any item $z \in Z^f$, define the following two sets:

$Y(z)$ contains those items from Y which appear after z in I.
$Z^f(z)$ contains the item z and all z' from Z^f appearing after z in I.

If necessary, when more than one sequence is involved, we subscript these sets with the name of the sequence (Z^f_I, for instance).

Let e denote the maximal empty space in any of Worst-Fit's bins, after processing the request sequence I. The case $e \geq 1$ was considered in Lemma 3.10. We now turn to the case $e = 0$, beginning with some lemmas which allow us to make assumptions about the request sequences which give the worst performance ratio for Worst-Fit.

LEMMA 3.23. *If, for some sequence I, Worst-Fit places two items from Y_I in the same bin, then there exists another sequence I' on which Worst-Fit places all items from $Y_{I'}$ in separate bins and on which the performance ratio of Worst-Fit is smaller.*

Proof. First we show that if, for some sequence I, Worst-Fit places some item $x \in X_I$ on top of some item $y \in Y_I$ (call this an inversion), then there is another sequence I', containing exactly the same items as I, for which Worst-Fit and OPT accept exactly the same items as when given I, but Worst-Fit never places an item from $X_{I'}$ on top of an item from $Y_{I'}$.

We modify I to obtain I', correcting one of these inversions at a time. Suppose that Worst-Fit places $x \in X_I$ directly on top of $y \in Y_I$ in bin j. Clearly, y occurs before x in I. Due to fairness, y must also be larger than x since OPT accepts x but not y. Let I' be identical to I up until the point where y appears. Replace y by x. Then x will still be placed in bin j. Let the next items from I' be the same as the next items from I up to the point where Worst-Fit would put something on top of x in bin j. Assuming there was room for it, insert the item y at this point, and let the rest of I' be the same as I. Since x is smaller than y, the item y will appear in

I' no later than where x appeared in I. Thus, Worst-Fit will place the items from I' in exactly the same bins as the items from I, the only difference being that one item from X will be placed below an item from Y which it had been placed on top of when I was given. OPT will accept exactly the same items as before and place them exactly as before, since the only difference is that it receives an item it would accept anyway earlier and an item it would reject anyway later. Thus the performance ratio is unchanged. This process can be repeated until there are no inversions.

Suppose that for some sequence I, Worst-Fit places more than one item from Y_I in some bin. From the above, we may assume without loss of generality that Worst-Fit never places an item from X_I on top of an item from Y_I. Modify I to obtain I' as follows: Consider each bin which receives more than one item from Y_I, one at a time. Among the items from Y_I in the bin, choose the item O which occurs first in the request sequence I. Replace O in the sequence by a single item of size exactly equal to the sum of the sizes of all of the items from Y_I in that bin. Remove all of those other items from the request sequence. Note that all of the items which are merged had originally been placed directly on top of each other.

By induction, carrying out this modification for one bin at a time, it follows that Worst-Fit places a new item in the same bin as all of the old items from Y_I that it replaced and gives the same placement to all other items.

In addition, OPT cannot improve its ratio by accepting some of these new items, since then it could also have done it on the sequence I by accepting some of the items from Y_I. Thus, it accepts exactly the same items as from the sequence I. Hence, Worst-Fit accepts fewer items from I', while OPT accepts the same number, so the performance ratio becomes smaller. □

The following proposition is used in the next lemma and in the proof of the theorem.

PROPOSITION 3.24. *Given a request sequence I and a $z \in Z^f$ and suppose that for all $w \in Z^f(z)$ it is the case that $|Z^f(w)| \leq |Y(w)|$, then there exists a 1–1 mapping $g : Z^f(z) \to Y(z)$ such that for all $w \in Z^f(z)$, $g(w)$ occurs after w in I.*

Proof. Enumerating the items in $Y(z)$ and $Z^f(z)$ separately, starting from y_{last} and working in the direction of the beginning of the sequence I, g could be defined as the mapping which takes an item from $Z^f(z)$ numbered j to the item in $Y(z)$ numbered j. □

It would be tempting to move items accepted by OPT to the end of the sequence and then convert these to unit size items. Since OPT must be fair, there is no guarantee that it would accept exactly the same items. In fact, it might be forced to accept some items from Y, which could use up more volume than the moved elements from Z. However, under some circumstances, it is possible to perform these moves which will be used in the next lemma. It shows that an additional assumption can be made on worst-case request sequences, in those cases where Worst-Fit packs all bins so that they are completely full.

LEMMA 3.25. *If, for some sequence I for which $e = 0$, there exists an item $z \in Z_I^f$ such that $|Z_I^f(z)| \geq |Y_I(z)|$, then there exists another sequence I', where for all items $z \in Z_{I'}^f$, $|Z_{I'}^f(z)| < |Y_{I'}(z)|$, and on which the performance ratio of Worst-Fit is no larger.*

Proof. Let I be a sequence such that there exists a $z \in Z_I^f$ with $|Z_I^f(z)| \geq |Y_I(z)|$. Let z be the last item such that $|Z_I^f(z)| \geq |Y_I(z)|$. It must be the case that $|Z_I^f(z)| = |Y_I(z)|$, and $|Z_I^f(w)| < |Y_I(w)|$ for all $w \in Z_I^f$ which occur after z. From Proposition 3.24, we know that there must exist a 1–1 mapping $g : Z_I^f(z) \to Y_I$ such

that for all $w \in Z_I^f(z)$, $g(w)$ occurs after w in I. Since Worst-Fit is fair and rejects w while accepting $g(w)$, $g(w)$ must be smaller than w. This means that the total volume of all items in $Z_I^f(z)$ must be greater than the total volume in $Y_I(z)$.

Modify I to obtain I' by removing all items from $Z_I^f(z)$ and adding that many items of size 1 right after the last item accepted by Worst-Fit. This will not affect which items Worst-Fit accepts, since, by assumption, $e = 0$, which means that all bins are full at that point. On the other hand, from the new sequence I', OPT will accept some items from $Y_I(z)$, and since every moved item has size 1, it can accept at least one additional item for each item from $Y_I(z)$ which it rejects.

Since $|Z_I^f(z)| = |Y_I(z)|$, OPT accepts at least as many items from I' as from I, so the performance ratio of Worst-Fit is no larger. □

There is no problem in assuming the forms implied by Lemmas 3.23 and 3.25 simultaneously.

COROLLARY 3.26. *Given k and n, there exists a request sequence I such that the following hold:*

1. *There is no request sequence I' which Worst-Fit packs so that $e = 0$, for which we have that $\frac{WF(I')}{OPT(I')} < \frac{WF(I)}{OPT(I)}$.*
2. *Worst-Fit places all items from Y_I in separate bins.*
3. *If Worst-Fit packs I so that $e = 0$, then for all items $z \in Z_I^f$, $|Z_I^f(z)| < |Y_I(z)|$.*

Proof. Since one can assume that not both Worst-Fit and OPT reject the same item from I, there are only a finite number of sequences to be considered; one of them must give the worst performance ratio. Begin with that request sequence. The construction from Lemma 3.23 can be applied to ensure that the second condition holds. If $e = 0$ now, the construction from Lemma 3.25 can be applied, without changing Worst-Fit's behavior, so the last two properties can hold simultaneously. □

Now we are ready to prove a lower bound on the competitive ratio of Worst-Fit.

THEOREM 3.27. *For Fair Bin Packing, the competitive ratio of Worst-Fit is at least $\frac{3}{2+k}$.*

Proof. If for some sequence I, the largest empty space, e, remaining in Worst-Fit is greater than zero, then by Lemma 3.10, the performance ratio r is at least $\frac{3}{2+k}$. Therefore we assume that $e = 0$.

By Corollary 3.26, we can assume that Worst-Fit places no two items from Y in the same bin. Let B_{WF} be the set of bins which receive items from Y, and let $b = |B_{\mathrm{WF}}| = |Y|$. We first show that $|X| \geq 2(b - |Z^f|)$.

Let $U \subseteq X$ be those items from X which are placed in bins belonging to B_{WF}. For any item $x \in U$, OPT must place it with another item from X or from Z^f, or it would be unable to reject the item from Y, which Worst-Fit placed with x. Therefore if we let B_{OPT} be those bins which receive items from $U \cup Z^f$ from OPT, then every bin in B_{OPT} receives either at least two items from X or at least one from Z^f.

Consider those bins which are not in B_{OPT}. When the last item $y_{last} \in Y$ is given to OPT, such bins contain only items from $X^f \setminus U$. Recall that i is the bin where Worst-Fit placed the last item $y_{last} \in Y$. Let V be the total volume of all items from U placed in bin i by Worst-Fit. If j is a bin, not in B_{WF}, Worst-Fit placed at least one item from X^f there. If it received more than one item from X^f, only the last one could have size greater than V. Otherwise, by its strategy, Worst-Fit would have placed the next item (after the one of size greater than V) from bin j in bin i.

Thus, only this last item could be placed by OPT in a bin which has no other items from $X \cup Z^f$. The reason for this is that, since the other items have size at most V, a bin with such an item alone would have to accept the item y_{last}.

This means that only $n - b$ of OPT's bins could have no more than one item from X and no items from Z^f, so at least b bins have either at least two items from X or at least one from Z^f. Hence, even if all those from Z^f are in separate bins, $|X| \geq 2(b - |Z^f|)$.

Now Worst-Fit places the first n items, all of which must be in X, in different bins, so no item in Y can be larger than $k - 1$. By Corollary 3.26, Proposition 3.24, and fairness, items in Z^f must be larger than corresponding items in Y, so since $e = 0$, $|Z \setminus Z^f| \leq (k-1)(|Y| - |Z^f|)$, which implies that $|Z| \leq |Z^f| + (k-1)(b - |Z^f|)$.

Thus, the ratio r is bounded by
$\frac{|X|+|Y|}{|X|+|Z|} \geq \frac{|X|+b}{|X|+|Z^f|+(k-1)(b-|Z^f|)} \geq \frac{1}{\frac{|X|+|Z^f|}{|X|+b}+\frac{(k-1)(b-|Z^f|)}{3b-2|Z^f|}} \geq \frac{1}{1+\frac{k-1}{3}} = \frac{3}{2+k}$. □

3.5. The accommodating functions for First-Fit and Worst-Fit. Finally, we prove upper bounds on the accommodating functions for First-Fit and Worst-Fit. All of the results in this subsection also hold if one relaxes the restriction that all items must be integer-sized to a restriction that the bins have unit size and the smallest item has size $1/k$. The upper bound for First-Fit is close to the general lower bound of Theorem 3.13, so it is almost tight.

THEOREM 3.28. *For Fair Bin Packing, First-Fit has an accommodating function of at most* $\frac{\alpha}{1+(\alpha-1)(k-1)}$ *for* $1 < \alpha < 2$.

Proof. The adversary's request sequence is divided into four phases. First, give $\alpha n - n$ items of unit size, and second, give one item of size $k - (\alpha n - n) \bmod k$. If k divides $\alpha n - n$, this item has size 0 and is not given. Third, give $n - \lceil \frac{\alpha n - n}{k} \rceil$ items of size k. Fourth, give $(\alpha n - n)(k-1)$ items of unit size.

First-Fit accepts those items in the first three phases. The off-line algorithm places the first n items in separate bins, rejects the remaining long items, and accepts all items from phase 4. The performance ratio is at most $\frac{(\alpha n-n)+(n-\lfloor \frac{\alpha n-n}{k} \rfloor)}{n+(\alpha n-n)(k-1)} \leq \frac{\alpha n}{n+(\alpha n-n)(k-1)} = \frac{\alpha}{1+(\alpha-1)(k-1)}$. □

The proof of Theorem 3.22, giving an upper bound on Worst-Fit's competitive ratio, can be extended to prove an upper bound on Worst-Fit's accommodating function.

THEOREM 3.29. *For Worst-Fit, if* $1 \leq \alpha \leq 1 + \frac{1}{n}\lceil \frac{(n-1)(k-n)+(n-1)}{k} \rceil$ *and* $k \geq n$, *then* $\mathcal{A}_{WF}(\alpha) \leq \frac{3-\frac{2}{n}}{3-\frac{3}{n}+(\alpha-1)k(1-\frac{1}{k-n+1})}$.

Proof. In the proof of Theorem 3.22, since all of the items in the first three phases of the request sequence fit in n bins, to determine how many bins would be necessary for an optimal off-line algorithm one needs only to compute how many bins are necessary for phase 4. Thus, $\alpha = 1 + \frac{1}{n}\lceil \frac{(n-1)(k-n)+(n-1)}{k} \rceil$. Hence, for any value of α less than this, replacing phase 4 by $(\alpha-1)kn$ items of size 1 gives an α-sequence for which Worst-Fit will accept exactly those requests in the first three phases, while OPT can accept all items, except some of the items from phase 3. The number of items from phase 3 which OPT can accept is $\lfloor \frac{kn-((n-1)(n-1)+n-1+(\alpha-1)kn)}{k-(n-1)} \rfloor = \lfloor \frac{n(k-n+1)-(\alpha-1)kn}{k-n+1} \rfloor$. Thus, the accommodating function is

$$\begin{aligned} \mathcal{A}_{\mathrm{WF}}(\alpha) &\leq \frac{3n-2}{2n-2+(\alpha-1)kn+n-\lfloor \frac{(\alpha-1)kn}{k-n+1} \rfloor} \\ &\leq \frac{3-\frac{2}{n}}{3-\frac{3}{n}+(\alpha-1)k(1-\frac{1}{k-n+1})}. \end{aligned}$$ □

4. The unit price seat reservation problem. The competitive ratio on accommodating sequences was introduced in [9][3] in connection with the seat reservation problem, which was originally motivated by some ticketing systems for trains in Europe. The setup is as follows: A train with n seats travels from a start station to an end station, stopping at $k \geq 2$ stations, including the first and last. Reservations can be made for any trip from a station s to a station t. The passenger is given a single seat number when the ticket is purchased, which can be any time before departure. The algorithms (ticket agents) attempt to maximize income; i.e., the sum of the prices of the tickets sold. For political reasons, the problem must be solved in a *fair* manner, i.e., the ticket agent may not refuse a passenger if it is possible to accommodate him when he attempts to make his reservation. In this paper, we consider only the pricing policy in which all tickets have the same price, the *unit price problem*; for the proportional price problem, where the price of the ticket is proportional to the distance traveled, there does not appear to be any significant difference between the competitive ratio and the competitive ratio on accommodating sequences. We define the accommodating function $\mathcal{A}(\alpha)$ for the seat reservation problem to be the ratio of how well an on-line algorithm can do compared to the optimal off-line algorithm, OPT, when an optimal off-line algorithm could have accommodated all requests if it would have had $\alpha n \geq n$ seats. The accommodating function could help the management in determining how much benefit could be gained by adding an extra car to the train, given their current distribution of request sequences. Notice that the fairness criterion is a part of the problem specification. Thus, even though the optimal off-line algorithm knows the entire sequence in advance, it too must process the sequence in the given order and do so fairly.

The seat reservation problem is similar to the problem of coloring an interval graph on-line, which has been well studied because of applications to dynamic storage allocation. The difference is that with graph coloring, all vertices must be given a color and the goal is to minimize the number of colors. With the seat reservation problem, there is a fixed number of colors, and the goal is to maximize the number of vertices that get colors. We use, however, an interesting result from interval graph theory: Interval graphs are *perfect* [20], so the size of the largest clique is exactly the number of colors needed. Thus, when there is no pair of stations $(s, s+1)$ such that the number of people who want to be on the train between stations s and $s+1$ is greater than n, the optimal off-line algorithm will be able to accommodate all requests. The contrapositive is also clearly true; if there is a pair of stations such that the number of people who want to be on the train between those stations is greater than n, the optimal off-line algorithm will be unable to accommodate all requests. We will refer to the number of people who want to be on the train between two stations as the *density* between those stations.

4.1. Bounds on the accommodating function. In [9], the following lower bounds for the competitive ratio and the competitive ratio on accommodating sequences were proven: Any algorithm for the unit price seat reservation problem is $\frac{2}{k}$-competitive, and any algorithm for the unit price seat reservation problem is $\frac{1}{2}$-competitive on accommodating sequences. The key idea for the proof of the theorem bounding the competitive ratio on accommodating sequences is also used to prove a lower bound on the accommodating function, and the result generalizes the one for the competitive ratio on accommodating sequences.

[3] It was called the accommodating ratio there.

THEOREM 4.1. *$\mathcal{A}(\alpha) \geq \frac{1}{2+(k-2)(1-\frac{1}{\alpha})}$ is a lower bound for the unit price seat reservation problem.*

Proof. Consider any algorithm $\mathbb{A}$ for the unit price seat reservation problem and any request sequence, I, which an optimal off-line algorithm could have accommodated with $\rho(I) \geq n$ seats. Let $l = \rho(I) - n$, and suppose that $\mathbb{A}$ accepts h intervals. We first show that $\mathbb{A}$ rejects at most $h + l(k-1)$ intervals.

Let S denote the seating assignment found by the on-line algorithm, and let U be the set of unseated intervals. First, some of the intervals in U will be assigned to distinct intervals in S. Let S' be a seating assignment which is initialized to be the same as S, but which will be altered by the following process. Note that the only changes will be to increase the lengths of some intervals in S'.

First order the intervals in U by increasing left endpoint (starting station), breaking ties arbitrarily. Now process these intervals, one by one, in increasing order.

For a given interval $I \in U$, if there is no seat which is empty in S' from the point where the passenger wants to get on until at least the next station, leave I in U. Otherwise, find such a seat. Since $\mathbb{A}$ is fair and the interval I was rejected, the interval I could not be placed on that seat, so there must be a first (leftmost) interval J assigned to that seat in S' which overlaps the interval I. Assign the interval I to the interval J. Now remove I from U and replace J on this seat in S' by an interval K, which is as much of $I \cup J$ as will currently fit on that seat. Clearly, all of the intervals which are now seated in S' and all of the unseated intervals currently in U could be seated by an optimal algorithm on αn seats, since this operation cannot increase the density anywhere. This process can be repeated. The order of processing ensures that each interval $I \in U$ which gets assigned to an interval in S gets assigned to a distinct interval in S. Thus, after all of U has been processed, at most h intervals have been removed from it. For every interval I' remaining in U, the leftmost unit segment (the point where the passenger wants to get on until the next station) has density n in S', so there is now density at most l for that unit segment in U. The number of possible distinct leftmost segments in U is at most $k-1$, so the total number of leftmost segments remaining in U, and thus the total number of intervals in U, is at most $l(k-1)$. We have now shown that $\mathbb{A}$ rejects at most $h + l(k-1)$ intervals.

To compute a lower bound on the ratio of what $\mathbb{A}$ accepts to what OPT accepts, we need to have a lower bound on the number of intervals $\mathbb{A}$ accepts and an upper bound on the number of intervals OPT accepts. We may assume that there are no intervals in the request sequence which both $\mathbb{A}$ and OPT reject, since removing them from the sequence changes nothing. Since an optimal off-line algorithm could not have accommodated all of the requests with fewer than $\rho(I)$ seats, but OPT has only n seats, OPT must reject at least l intervals, and all of these must have been accepted by $\mathbb{A}$. Clearly, the first n intervals in the request sequence must have been accepted by both $\mathbb{A}$ and OPT. Thus, $\mathbb{A}$ accepts $h \geq n + l = \rho(n)$ intervals. Of the h intervals which $\mathbb{A}$ accepts, OPT rejects at least l of them. Additionally, there are at most $h + l(k-1)$ intervals which OPT accepts, but $\mathbb{A}$ does not. Thus, a lower bound on the accommodating function is $\mathcal{A}(\alpha) \geq \frac{h}{2h+l(k-1)-l} = \frac{1}{2+(k-2)\frac{l}{h}} \geq \frac{1}{2+(k-2)\frac{\rho(n)-n}{\rho(n)}} \geq \frac{1}{2+(k-2)(1-\frac{1}{\alpha})}$. ∎

In [9], the following upper bounds for the competitive ratio and the competitive ratio on accommodating sequences were proven: No deterministic algorithm for the unit price seat reservation problem is more than $\frac{8}{k+5}$-competitive, and no deterministic algorithm for the unit price seat reservation problem is more than $\frac{8k-9}{10k-15}$-competitive

on accommodating sequences, when k is divisible by 3. The proof proving an upper bound on the accommodating function for any algorithm for Unit Price Bin Packing, is very similar to the proof of the theorem in [9] giving the upper bound on the competitive ratio. The result obtained is very close to that for the competitive ratio when $\alpha > \frac{5}{4}$.

THEOREM 4.2. $\mathcal{A}(\alpha) \leq \frac{4}{3+2(k-2)\min\{\frac{1}{4},\alpha-1\}}$ *is an upper bound for the unit price seat reservation problem.*

Proof. The following is an adversary argument, so the request sequence depends on the on-line algorithm $\mathbb{A}$'s behavior. Assume that n is divisible by 2. The adversary begins with $\frac{n}{2}$ pairs of requests for $[1,2]$ and $[k-1,k]$ intervals. Suppose that the algorithm $\mathbb{A}$ places them such that after these requests there are exactly q seats which contain two intervals. Then $n-2q$ of the seats have exactly one short interval scheduled. Next, the adversary will give q requests for $[1,k]$ intervals, followed by $\frac{n-2q}{2}$ requests for $[1,k-1]$ intervals, $\frac{n-2q}{2}$ requests for $[2,k]$ intervals, and q requests for $[2,k-1]$ intervals, all of which can be accommodated by $\mathbb{A}$. Now the train is full. One of two cases will occur:

Case 1. $q \geq \frac{n}{4}$.

Case 2. $q < \frac{n}{4}$.

If Case 1 occurs, the adversary will give $\min\{q, \alpha n - n\}$ requests for each of the intervals $[1,2]$, $[2,3]$, $[3,4]$, $\ldots$, $[k-1,k]$, none of which can be accommodated by $\mathbb{A}$. On the other hand, OPT could put each of the short intervals on a separate seat, so that it would be unable to accommodate the q $[1,k]$ intervals, but all of the other intervals would fit. The on-line algorithm $\mathbb{A}$ is able to accommodate $2n$ requests, while OPT can accommodate $2n - q + \min\{q, \alpha n - n\}(k-1)$ requests. Since $\frac{n}{4} \leq q \leq \frac{n}{2}$, this ratio is less than $\frac{2n}{2n-\frac{n}{2}+\min\{\frac{n}{4},\alpha n-n\}(k-1)} = \frac{4}{3+2(k-1)\min\{\frac{1}{4},\frac{\alpha n-n}{n}\}}$.

If Case 2 occurs, the adversary will give $\min\{\frac{n-2q}{2}, \alpha n - n\}$ requests for each of the intervals $[2,3]$, $[3,4]$, $\ldots$, $[k-1,k]$, none of which can be accommodated by $\mathbb{A}$. On the other hand, OPT could pair up the short intervals, putting two per seat, so that it would be unable to accommodate the $\frac{n-2q}{2}$ $[2,k]$ intervals, but all of the other intervals would fit. The on-line algorithm $\mathbb{A}$ is able to accommodate $2n$ requests, while OPT can accommodate $2n - \frac{n-2q}{2} + (k-2)\min\{\frac{n-2q}{2}, \alpha n - n\}$ requests. Since $q < n/4$, this ratio is less than $\frac{2n}{2n-\frac{n}{4}+(k-2)\min\{\frac{n}{4},\alpha n-n\}} = \frac{8}{7+4(k-2)\min\{\frac{1}{4},\frac{\alpha n-n}{n}\}}$.

This argument assumes that $k \geq 4$, but the result also clearly holds for $k = 2$ and $k = 3$. □

As an example of a specific on-line algorithm, one might consider First-Fit, which always processes a new request by placing it on the first seat which is unoccupied for the length of that journey. The lower bound from Theorem 4.1, on the accommodating function for any algorithm, clearly applies to this specific algorithm. It also applies to Best-Fit, which always processes a new request by placing it on a seat so it leaves as little total free space as possible on that seat immediately before and after that passenger's trip. The following result, giving an upper bound on the accommodating function for these two specific algorithms, should be compared with the results in [9], which give upper bounds for the competitive ratio and the competitive ratio on accommodating sequences for First-Fit and Best-Fit: First-Fit and Best-Fit have competitive ratios which are no better than $\frac{2-\frac{1}{k-1}}{k-1}$ and competitive ratios on accommodating sequences no better than $\frac{k}{2k-6}$ for the unit price seat reservation problem. The proof is very similar to the proof of the theorem in [9] which gives the upper bound on the competitive ratios for First-Fit and Best-Fit.

THEOREM 4.3. *For the unit price seat reservation problem, First-Fit and Best-Fit have* $\mathcal{A}(\alpha) \leq \max\{\frac{2-\frac{1}{k-1}}{k-1}, \frac{2-\frac{1}{k-1}}{1+(k-1)(\alpha-1)}\}$.

Proof. We will state everything in terms of the First-Fit algorithm, but Best-Fit would behave exactly the same. We will assume that n is divisible by $k-1$. The request sequence will start with $\frac{n}{k-1}$ requests for each of the intervals $[1,2],[2,3],[3,4],\ldots,[k-1,k]$ which First-Fit will put in the first $\frac{n}{k-1}$ seats. Then there will be $n-\frac{n}{k-1}$ requests for $[1,k]$ intervals, which First-Fit will put in the remaining seats. At this point, the train will be full, but there will now be $\min\{n-\frac{n}{k-1}, \alpha n - n\}$ requests for each of the intervals $[1,2],[2,3],[3,4],\ldots,[k-1,k]$, all of which First-Fit will be unable to accommodate. It will accommodate a total of $2n-\frac{n}{k-1}$ requests. OPT will put each of the original first intervals on a different seat, thus arranging that it can reject the longest intervals. Then, it will be able to accommodate all of the additional short intervals. Thus, it will accommodate n of the original short intervals, plus $\min\{n-\frac{n}{k-1}, \alpha n - n\}(k-1)$ of the later short intervals. This gives a ratio of $\max\{\frac{2-\frac{1}{k-1}}{k-1}, \frac{2-\frac{1}{k-1}}{1+\frac{\alpha n-n}{n}(k-1)}\}$. This argument assumes that $k \geq 3$, but the result trivially holds for $k = 2$ too. □

5. Other problems. It is natural to ask if the accommodating function can be defined for any on-line problem. This is equivalent to asking if α-sequences can be defined for every on-line problem. The answer is clearly "yes" if there is no requirement that a relevant resource be considered; then the accommodating function is constant, and its value is the competitive ratio. This answer is not particularly interesting. It seems, however, that for most on-line problems, there is some relevant resource which can be used to define α-sequences and therefore also the accommodating function. For paging, the obvious resource is the number of pages in fast memory; for scheduling, it is the number of machines available; and for server problems, it is the number of servers.

Unfortunately, for some on-line problems, using the accommodating function with the obvious resource choice and $\alpha \geq 1$ fails to result in additional insight compared with what is already known from the competitive ratio. For example, one of the best known on-line problems is paging. The well-known lower bound results, which show that any deterministic on-line algorithm has a competitive ratio of at least k, where k is the number of pages in main memory, holds even if there are only $k+1$ pages in all. Thus, nothing further is said about how much it helps to have extra memory, unless one actually has enough extra memory to hold all of a program and its data. In fact, however, it was shown later [8] that the accommodating function for the paging problem becomes more interesting when $\alpha < 1$.

The situation is similar when considering the accommodating function for problems which generalize the paging problem, such as the k-server problem and metrical task systems.

5.1. Minimizing flow times on *m* identical machines. As an example of a very different type of problem where the accommodating function can be applied, we have considered a scheduling problem: the problem of minimizing flow time in a situation where there are m identical machines and preemption is allowed. Let J be the sequence of jobs. A job $j \in J$ arrives at its release time r_j, and its processing time p_j is known. The total flow time is $\sum_{j\in J}(C_j - r_j)$, where C_j denotes the completion time of job j. There are some very nice results in [27] showing that shortest remaining processing time (SRPT) has a competitive ratio of $O(\log P)$ for this problem, where

P is the ratio between the processing time for the longest job and the shortest job. They also show that any randomized algorithm for the problem has a competitive ratio of $\Omega(\log P)$.

The concept of an accommodating function can be applied to this problem, even though it is a minimization problem and no rejections are allowed. Given a request sequence J, there is an absolute minimum flow time—the sum s of the processing times for all jobs in J. Thus, one can define the competitive ratio on accommodating sequences by restricting the request sequences to those which OPT could schedule with total flow time s. This means that all jobs can be scheduled immediately when they arrive. Thus, any on-line algorithm which assigns an incoming job to some free processor, when such a processor exists, will also schedule that sequence with total flow time s, giving a competitive ratio on accommodating sequences of 1, which is significantly different from the competitive ratio. The accommodating function $\mathcal{A}(\alpha)$ can be defined by restricting the request sequences to those in which an optimal off-line algorithm could have begun each job immediately upon arrival if it had $\alpha m \geq m$ machines available.

Using the techniques from the result in [27], proving a lower bound on the competitive ratio for SRPT, and lengthening the adversary's sequence appropriately gives a lower bound of $\Omega(\log_m P)$ on the performance ratio of SRPT, even when OPT could have handled the request sequence with only $m+1$ processors. Thus, SRPT's behavior is similar to that of all paging algorithms; restricting to sequences which OPT could accommodate with only one extra unit of the resource gives essentially the same result as allowing any sequence whatsoever. It is also possible to show that all other algorithms for this problem also have a sudden change from the competitive ratio on accommodating sequences to the competitive ratio.

Although some of the ideas from the $\Omega(\log P)$ lower bound on the competitive ratio in [27] are used in the proof here, their adversary uses sequences with $\alpha = \frac{3}{2}$, while our adversary only needs $\alpha m = m + 1$.

THEOREM 5.1. *For $\alpha m = m + 1$ and $m > 4$, the performance ratio of any deterministic on-line algorithm is $\Omega(\log_m P)$.*

Proof. Consider a deterministic on-line algorithm $\mathbb{A}$ and assume P is a power of m. An adversary can give a request sequence consisting of $L = \lfloor \log_m P - 1 - \log_m 4 - 2\log_m \log_m P \rfloor$ phases. For phase $i = 0, \ldots, L-1$, let $p_i = \frac{P}{m^i}$, and $r_i = m * \sum_{j=0}^{i-1} p_j$. The adversary will repeat the same set of jobs m times. Let j denote the repetition number, and let $r_{ij} = r_i + j * p_i, j = 0, \ldots, m-1$. The following jobs are given:

1. One job of size p_i at time r_{ij};
2. m unit size jobs at each of the times $r_{ij} + k, k = 0, \ldots, p_i(1 - \frac{1}{m}) - 1$.

Let S_{ij} denote the set of all unit size jobs given in phase i and repetition j, and let U_{ij} count the number of jobs from S_{ij} that are not finished by $\mathbb{A}$ before time $w_{ij} = r_{ij} + p_i(1 - \frac{1}{m})$.

There are two cases depending on the U_{ij}.

Case 1. There exist i, j, such that $U_{ij} \geq m \log_m P$.

Case 2. For all $i, j, U_{ij} < m \log_m P$.

If Case 1 occurs for i_0, j_0, the above release pattern is stopped at time $w_{i_0 j_0}$. Instead, the adversary gives m unit size jobs at each of the next P^3 time units.

An off-line algorithm OPT could finish all jobs from one repetition before the next starts by processing the long job on one machine and the unit size jobs on the other machines. Beginning with phase i_0 and repetition j_0, OPT should process all unit size jobs, including the unit size jobs given after $w_{i_0 j_0}$, immediately when they

are released. Then OPT has only one long unfinished job at time $w_{i_0 j_0}$, which will be delayed for time P^3.

The total flow time for OPT is then at most $m^2P^2 \log P + (m+1)P^3 + P$, since from the first part there are less than $m^2 P \log P$ jobs, which will be delayed for at most P time.

Since Case 1 has occurred, the on-line algorithm has at least $m \log_m P$ jobs delayed at every time unit for P^3 time steps. The total flow time for the on-line algorithm is then more than $mP^3 \log_m P$.

In Case 1, the performance ratio is then more than $\frac{mP^3 \log_m P}{m^2P^2 \log P + (m+1)P^3 + P}$.

If Case 2 occurs, an off-line algorithm, OPT, can follow a pattern similar to Case 1 and finish every job from phase i before phase $i+1$ starts. Call the time just after the last phase ends r_L. Starting at time r_L, the adversary gives m unit size jobs at each of the next P^3 time units. OPT can process them immediately upon arrival.

In this case, OPT has a total flow time of less than $m^2P^2 \log P + mP^3$.

The on-line algorithm $\mathbb{A}$ will have many long jobs hanging at time r_L. Fixing a phase i, we want to calculate the possible processing time for long jobs in this and the following phases. In phase i, the long jobs appear one at a time, so in repetition j there are $j+1$ available. After time w_{ij} there are $\frac{P}{m^{i+1}}$ time units remaining in repetition j, and since this is Case 2, there is a total of less than $m \log_m P$ time units available on all the processors together before time w_{ij}. Thus, the maximum amount of time the m long jobs from phase i can be processed within phase i is in total bounded by $\sum_{j=1}^{m}(j\frac{P}{m^{i+1}} + m \log_m P)$. For the following phases the total time available for processing these m jobs from phase i is bounded by $\sum_{j=i+1}^{L-1}(m\frac{P}{m^{(j+1)}} + m^2 \log_m P)$. This adds up to at most $2m^2 \log_m^2 P + \frac{p_i}{m}\frac{m(m+1)}{2} + p_i \leq p_i \frac{m+2}{2} + p_i$, since by the definition of L, $2m^2 \log_m^2 P \leq \frac{p_i}{2}$ for all i. It follows that for a fixed phase i, at most $\lfloor \frac{m+4}{2} \rfloor$ of the long jobs could be run to completion. Since we have L phases, at least $\lceil \frac{m-4}{2} L \rceil$ long jobs are unfinished at time r_L. This gives a flow time of at least $\frac{m-4}{2} P^3 \log_m P$.

In Case 2, the performance ratio is at least $\frac{\frac{m-4}{2} P^3 \log_m P}{m^2P^2 \log P + mP^3}$. □

The difference between this lower bound and the lower bound on the competitive ratio from [27] is quite small: $\Omega(\log_m P)$ versus $\Omega(\log_2 P)$, i.e., for any fixed m, the bounds are the same.

6. Concluding remarks. It is now clear that in comparing on-line algorithms, the competitive ratio on accommodating sequences can give different information than the competitive ratio. This is true for Fair Bin Packing and two algorithms investigated in this paper, but these results also indicate that the competitive ratio on accommodating sequences and the accommodating function could be very useful measures generally.

With respect to Fair Bin Packing, the choice as to which algorithm to use depends on which ratio is more relevant in a specific situation. This, in turn, would depend on the actual distribution of request sequences. However, one might guess that the competitive ratio on accommodating sequences actually gives the more useful answer in most cases, since the sequences which cause First-Fit to perform so poorly with respect to the competitive ratio are in some sense rather artificial. The sequences are designed so that OPT can arrange to reject certain "difficult" requests but continue to be "fair." This may simply be a blatant example of how some unusual request sequences can cause the competitive ratio to be excessively pessimistic.

We believe that there is a broad range of on-line problems for which analysis using the competitive ratio on accommodating sequences and the accommodating function will give interesting insights. More of these problems should be investigated. In particular, an open problem left here is finding a minimization problem which has a more gradual change from the competitive ratio on accommodating sequences to the competitive ratio.

In this paper, the accommodating function was only investigated for $\alpha \geq 1$. It was natural to use this restriction, since it spans from the competitive ratio on accommodating sequences to the standard competitive ratio, the two known interesting points. It has later been discovered [8] that the accommodating function is also interesting for $\alpha < 1$. There appear to be more problems which have interesting accommodating functions if one considers $\alpha < 1$, further distinctions can be made between known algorithms, and new interesting algorithms can be developed.

Two of the upper bounds proven in this paper have since been improved. The upper bound on First-Fit's competitive ratio on accommodating sequences for Fair Bin Packing has been improved to $\frac{5}{8}$ [1]. Thus, we can conclude that the competitive ratio of First-Fit on accommodating sequences is exactly $\frac{5}{8}$. In addition, it has been shown [3] that for any deterministic algorithm for the fair unit price seat reservation problem with three seats, the competitive ratio on accommodating sequences is at most $\frac{1}{2} + \frac{3}{k+5}$, where $k \geq 7$ and $k \equiv 1 \pmod 6$. This gives an upper bound of $\frac{1}{2} + \frac{3n-3}{2k+6n-(8+2c)}$ for n seats with the same restrictions on k.

Recently, the restriction of input sequences has proven useful in another context, congestion control on the Internet [24], where the competitive ratio is a function of this restriction. It does not appear that the functions in [24] can be viewed as accommodating functions.

Acknowledgments. We would like to thank the referees for their careful reading of the paper and their helpful suggestions for improvements.

REFERENCES

[1] Y. Azar, J. Boyar, L. Epstein, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Fair Versus Unrestricted Bin Packing*, Technical report PP-2000-20, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, 2000; also available via ftp://ftp.imada.sdu.dk/pub/papers/pp-2000/20.ps.gz.

[2] Y. Azar, J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Fair versus unrestricted bin packing*, in Proceedings of the Seventh Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 1851, Springer-Verlag, Berlin, 2000, pp. 200–213.

[3] E. Bach, J. Boyar, L. Epstein, L. M. Favrholdt, T. Jiang, K. S. Larsen, G.-H. Lin, and R. van Stee, *Tight bounds on the competitive ratio on accommodating sequences for the seat reservation problem*, J. Sched., to appear.

[4] E. Bach, J. Boyar, T. Jiang, K. S. Larsen, and G.-H. Lin, *Better bounds on the accommodating ratio for the seat reservation problem*, in Proceedings of the Sixth Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 1858, Springer-Verlag, Berlin, 2000, pp. 221–231.

[5] S. Ben-David and A. Borodin, *A new measure for the study of on-line algorithms*, Algorithmica, 11 (1994), pp. 73–91.

[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, Amsterdam, 1976.

[7] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, J. Comput. System Sci., 50 (1995), pp. 244–258.

[8] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen, *Extending the accommodating function*, submitted.

[9] J. Boyar and K. S. Larsen, *The seat reservation problem*, Algorithmica, 25 (1999), pp. 403–417.

[10] J. Boyar, K. S. Larsen, and M. N. Nielsen, *The accommodating function — a generalization of the competitive ratio*, in Proceedings of the Sixth International Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 1663, Springer-Verlag, Berlin, 1999, pp. 74–79.
[11] M. Chrobak and J. Noga, *LRU is better than FIFO*, Algorithmica, 23 (1999), pp. 180–185.
[12] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: A survey*, in Approximation Algorithms for NP-Hard Problems, Chapter 2, D. S. Hochbaum, ed., PWS Publishing Company, Boston, 1997, pp. 46–93.
[13] E. G. Coffman, Jr. and J. Y-T. Leung, *Combinatorial analysis of an efficient algorithm for processor and storage allocation*, SIAM J. Comput., 8 (1979), pp. 202–217.
[14] E. G. Coffman, Jr., J. Y-T. Leung, and D. W. Ting, *Bin packing: Maximizing the number of pieces packed*, Acta Inform., 9 (1978), pp. 263–271.
[15] J. Csirik and G. Woeginger, *On-line packing and covering problems*, in Online Algorithms, G. J. Woeginger and A. Fiat, eds., Lecture Notes in Comput. Sci. 1442, Springer-Verlag, Berlin, 1998, pp. 147–177.
[16] A. Fiat and G. J. Woeginger, *Competitive odds and ends*, in Online Algorithms, G. J. Woeginger and A. Fiat, eds., Lecture Notes in Comput. Sci. 1442, Springer-Verlag, Berlin, 1998, pp. 385–394.
[17] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell Systems Tech. J., 45 (1966), pp. 1563–1581.
[18] S. Irani and A. R. Karlin, *Online computation*, in Approximation Algorithms for NP-Hard Problems, Chapter 13, D. S. Hochbaum, ed., PWS Publishing Company, Boston, 1997, pp. 521–564.
[19] S. Irani, A. R. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, SIAM J. Comput., 25 (1996), pp. 477–497.
[20] T. R. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley and Sons, New York, 1995.
[21] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp. 299–325.
[22] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, in Proceedings of the 36th Annual IEEE Foundations of Computer Science, 1995, pp. 214–221.
[23] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.
[24] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, *Optimization problems in congestion control*, in Proceedings of the 41th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 66–74.
[25] E. Koutsoupias and C. H. Papadimitriou, *Beyond competitive analysis*, SIAM J. Comput., 30 (2000), pp. 300–317.
[26] D. König, *Graphs and matrices*, Mat. Fiz. Lapok, 38 (1931), pp. 116–119 (in Hungarian).
[27] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 110–119.
[28] C. A. Philips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 140–149.
[29] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.
[30] E. Torng, *A unified analysis of paging and caching*, Algorithmica, 20 (1998), pp. 175–200.
[31] N. Young, *The k-server dual and loose competitiveness for paging*, Algorithmica, 11 (1994), pp. 525–541.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 259–268

GENERATING BRACELETS IN CONSTANT AMORTIZED TIME*

JOE SAWADA†

Abstract. A bracelet is the lexicographically smallest element in an equivalence class of strings under string rotation and reversal. We present a fast, simple, recursive algorithm for generating (i.e., listing) k-ary bracelets. Using simple bounding techniques, we prove that the algorithm is optimal in the sense that the running time is proportional to the number of bracelets produced. This is an improvement by a factor of n (where n is the length of the bracelets being generated) over the fastest, previously known algorithm to generate bracelets.

Key words. bracelet, necklace, CAT algorithm, generate, forbidden substring

AMS subject classifications. 05-04, 68R05, 68R15

PII. S0097539700377037

1. Introduction.

The rapid growth in the fields of combinatorial chemistry and computational biology is resulting in an increased demand for efficient algorithms which produce exhaustive lists of combinatorial objects [1]. Dan Gusfield (see [9, p. xv]) claims that "significant contributions to computational biology might be made by extending or adapting [string] algorithms from computer science, even when the original algorithm has no clear utility in biology." In particular, correspondences between DNA sequences and restricted classes of circular strings are described in [3].

Within the mathematical sciences, researchers are constantly trying to find patterns hidden in the structure of combinatorial objects. The growing trend of using computers and algorithms to produce lists of such objects is allowing researchers to obtain more information about the objects themselves. Often, this will lead to a more thorough understanding of an object which may lead to new and interesting discoveries. In some cases, algorithms which produce exhaustive lists can be used to prove the existence of a related object [12].

An important consideration for any algorithm is its running time. For generation algorithms, the ultimate performance goal is an algorithm with computation proportional to the number of objects generated (where the computation reflects the total amount of change to the data structures, and not the time required to print out the object). Such algorithms are said to be CAT, for constant amortized time.

Strings with equivalence under rotation is one of the most fundamental types of combinatorial objects. Such objects, more commonly known as necklaces, arise naturally in many areas including knot theory, color printing, DNA sequencing, and the theory of free Lie algebras. Algorithms for generating necklaces and Lyndon words (aperiodic necklaces) were first developed by Fredricksen and Kessler [6] and Fredricksen and Maiorana [7]. These algorithms were proven to be CAT by Ruskey, Savage, and Wang [11].

Many applications, however, do not require all necklaces, but instead only those satisfying a particular restriction. A recursive necklace generation algorithm outlined in [2] has led to several algorithms which efficiently generate restricted classes of

*Received by the editors August 24, 2000; accepted for publication (in revised form) January 23, 2001; published electronically July 25, 2001. This research was supported by NSERC and partial support of Czech grant GAČR 201/99/0242 and ITI under project LN-00A 056.

http://www.siam.org/journals/sicomp/31-1/37703.html

†Department of Computer Science, University of Sydney, Sydney, Australia (sawada@cs.usyd.edu.au).

necklaces including binary unlabeled necklaces [2], fixed density necklaces [12], and necklaces with forbidden substrings [13].

Another restricted class of necklaces are bracelets. More specifically, bracelets are necklaces with equivalence under string reversal. Lists of bracelets are shown to have application in the calibration of color printers by Emmel [5]. However, the problem of efficiently generating these lists has remained open for some time. Previously, the fastest known algorithm to generate bracelets was a modification of Savage and Wang's necklace algorithm [11] by Lisonek [10]. This algorithm has running time $O(n \cdot B_k(n))$ (where $B_k(n)$ denotes the number of k-ary bracelets of length n), which is the same as the second algorithm outlined in the beginning of section 3.

The problem of efficiently generating bracelets is answered in this paper with the development of a bracelet generation algorithm that runs in constant amortized time. We begin with some background and definitions of the relevant objects in section 2. In section 3, we outline our bracelet generation algorithm. In section 4, we discuss strings with no 0^i substring (forbidden substrings). These strings are then used when we analyze our bracelet generation algorithm in section 5.

2. Background. We define a *necklace* to be the lexicographically smallest element of an equivalence class of k-ary strings under rotation. The set of all necklaces of length n is denoted $\mathbf{N}_k(n)$. The cardinality of $\mathbf{N}_k(n)$ is denoted $N_k(n)$. An aperiodic necklace is called a *Lyndon word*. The set of all k-ary Lyndon words of length n is denoted $\mathbf{L}_k(n)$ and has cardinality $L_k(n)$. A word α is called a *prenecklace* if it is the prefix of some necklace. The set of all k-ary prenecklaces of length n is denoted $\mathbf{P}_k(n)$. The cardinality of $\mathbf{P}_k(n)$ is denoted $P_k(n)$.

A *bracelet* is the lexicographically smallest element of an equivalence class of k-ary strings under string rotation and reversal (or a necklace that is also lexicographically minimal among the circular rotations of its reversal). The set of all k-ary bracelets is denoted $\mathbf{B}_k(n)$ and has cardinality $B_k(n)$. In each equivalence class associated with a given bracelet, there exists at most two necklaces: the bracelet itself and the necklace corresponding to the reversal of the bracelet. (In some cases, the two may be the same.) For example, the equivalence class that contains the bracelet 00112012 also contains the necklace 00210211.

Necklaces, Lyndon words, and prenecklaces can all be generated using the recursive necklace generation algorithm GenNecklaces(t, p) shown in Figure 2.1. It is important to have a solid understanding of this algorithm because it will be the basis for the bracelet generation algorithm developed in the following section. The basic idea behind the algorithm is to generate all length n prenecklaces, and then perform an appropriate test in the function PrintIt(p) to obtain the desired object. If necklaces are required, then the prenecklace is printed only if p divides n; if Lyndon words are required, then the prenecklace is printed if $n = p$. If $\alpha = a_1 \cdots a_{t-1}$ is a prenecklace with its longest Lyndon prefix having length p, then a length t prenecklace can be obtained by appending any value greater than or equal to a_{t-p} to α. The initial call is GenNecklaces(1,1) and a_0 is initialized to 0.

The following theorem provides enumeration formulas for necklaces, Lyndon words, prenecklaces, and bracelets.

THEOREM 2.1. *The following formulas are valid for all $n \geq 1$, $k \geq 1$:*

$$L_k(n) = \frac{1}{n} \sum_{d|n} \mu(d) k^{n/d}, \tag{2.1}$$

```
procedure GenNecklaces ( t, p : integer );
local j : integer;
begin
    if t > n then PrintIt( p )
    else begin
        a_t := a_{t-p};
        GenNecklaces( t + 1, p );
        for j ∈ {a_{t-p} + 1, ..., k - 2, k - 1} do begin
            a_t := j;
            GenNecklaces( t + 1, t );
end; end; end;
```

FIG. 2.1. *The recursive necklace algorithm.*

$$N_k(n) = \frac{1}{n}\sum_{d|n}\phi(d)k^{n/d}, \tag{2.2}$$

$$P_k(n) = \sum_{i=1}^{n} L_k(i), \tag{2.3}$$

$$B_k(n) = \begin{cases} \frac{1}{2}(N_k(n) + \frac{k+1}{2}k^{n/2}), & n \text{ even,} \\ \\ \frac{1}{2}(N_k(n) + k^{(n+1)/2}), & n \text{ odd.} \end{cases} \tag{2.4}$$

Proof. The equations for $L_k(n)$, $N_k(n)$, and $B_k(n)$ are proved by Gilbert and Riordan in [8]. The equation for $P_k(n)$ is proved in [2]. □

In the analysis of our bracelet algorithm it will be useful to look at another way to count prenecklaces. Let $P_k^0(n)$ count all k-ary prenecklaces of length n that begin with 0. Notice that the number of k-ary prenecklaces of length n beginning with 1 is equal to $P_{k-1}^0(n)$. Similarly the number of k-ary prenecklaces of length n beginning with 2 is $P_{k-2}^0(n)$. This observation leads to the following equation:

$$P_k(n) = \sum_{j=1}^{k} P_j^0(n). \tag{2.5}$$

3. Generating bracelets. In this section we outline a fast algorithm to generate bracelets. Since when $k = 1$, the only bracelet is 0^n, we assume $k \geq 2$. One algorithm for generating bracelets is to generate all k-ary necklaces of length n and then test each necklace against all rotations of its reversal. If no reversed rotation is less than the generated necklace, then the necklace is a bracelet. Since there are n rotations and each test takes $O(n)$ time, this naïve approach will give us an overall running time of $O(n^2 \cdot B_k(n))$ to generate all k-ary bracelets of length n.

A more sophisticated approach will use a necklace finding algorithm, which determines the necklace of a length n string in $O(n)$ time. Such an algorithm is easily derived from Duval's algorithm for factoring a string into Lyndon words [4] or from Theorem 2.1 in [2]. Using this technique, we need only compare the generated necklace with the necklace of its reversal. This approach yields a much better running time of $O(n \cdot B_k(n))$ to generate bracelets; however, it is still far from being CAT.

In the quest to find a faster algorithm to generate bracelets, we return to the original idea of comparing a generated necklace to every rotation of it reversal. We

start by making a simple observation.

OBSERVATION 1. *If a necklace α is of the form $a^i a_{i+1} \cdots a_n$ for some character $a \neq a_{i+1}$, then we need only test the reversed rotations that also begin with a^i.*

Taking this observation into account, we are making a large improvement on the number of reversed rotations we must check. For example, for the necklace 0010023003 we need only check the three reversed rotations that begin with 00: 0030032001, 0010030032, and 0032001003. To test each reversal we could wait until the entire necklace has been generated, but this will take $O(n)$ time per reversal and we will see no improvement over the naïve algorithms. Instead, if a character is generated in position j that satisfies the condition stated in Observation 1, we immediately compare the prenecklace $a_1 \cdots a_j$ with its reversal $a_j \cdots a_1$. This comparison will yield one of three outcomes. If $a_1 \cdots a_j > a_j \cdots a_1$, then we terminate the generation from this node since appending characters to the end of these strings will not affect their relative ordering. If $a_1 \cdots a_j < a_j \cdots a_1$, then no additional testing is required for this reversal. However, if $a_1 \cdots a_j = a_j \cdots a_1$, then more testing must be done on the tail of the strings which has yet to be generated.

Following the above approach, we still need to perform additional testing for the reversals starting at position j where $a_1 \cdots a_j = a_j \cdots a_1$. The number of such reversals could be as many as $n/2$. The following theorem addresses this issue.

THEOREM 3.1. *If $a_1 \cdots a_n$ is a necklace where $a_1 \cdots a_q = a_q \cdots a_1$ and there exists an r in $\{q+1, \ldots, n\}$ such that $a_1 \cdots a_r = a_r \cdots a_1$ and $a_{r+1} \cdots a_n \leq a_n \cdots a_{r+1}$, then $a_{q+1} \cdots a_n \leq a_n \cdots a_{q+1}$.*

Proof. Let $P_q = a_1 \cdots a_q$, $P_r = a_1 \cdots a_r$, $x = a_{q+1} \cdots a_r$, and $y = a_{r+1} \cdots a_n$. Let $\hat{x}$ and $\hat{y}$ denote the reversals of x and y, respectively. Since P_r and P_q are palindromes $P_r = P_q x = \hat{x} P_q$. Thus, $\alpha = P_q xy = \hat{x} P_q y$. But since α is a necklace, $\alpha = \hat{x} P_q y \leq P_q y \hat{x}$. Thus, since $y \leq \hat{y}$, $xy \leq y\hat{x} \leq \hat{y}\hat{x}$ as required. □

This theorem implies that we need only perform extra testing on the reversal starting at the largest position r such that $a_1 \cdots a_r = a_r \cdots a_1$. This extra testing is the comparison of $a_{r+1} \cdots a_n$ to $a_n \cdots a_{r+1}$. If $a_{r+1} \cdots a_n > a_n \cdots a_{r+1}$, then the generated string is *not* a bracelet. This test can be performed in constant time per character for each character generated after position $(n-r)/2 + r$.

Finally, we note that if $a_1 = a_n$, then the only strings that are bracelets (or necklaces) must be of the form a^n for some character a.

The following is a summary of the modifications required to transform GenNecklaces(t, p) into an algorithm which generates bracelets. Notice that each modification requires only a constant amount of computation per character generated except the addition of the function CheckRev(t, i).

- Add the parameter u to maintain the value of i from Observation 1: the number of consecutive equivalent characters at the start of the prenecklace (i.e., the prenecklace starts with a^u).
- Add the parameter v to maintain the number of consecutive a's at the end of the prenecklace, where $a = a_1$.
- Add the function CheckRev(t, i) to compare the prenecklace to its reversal (when $u = v$). If the prenecklace is greater than its reversal, it returns -1; if the prenecklace is less than its reversal, it returns 0; otherwise, the prenecklace is the same as its reversal and 1 is returned.
- Add the parameter r to maintain the length of the longest prenecklace equal to its reversal (i.e., the largest value r for which $a_1 \cdots a_r = a_r \cdots a_1$).
- Add a test to each character in a position greater than $(n-r)/2 + r$ which

```
function CheckRev( t, i: integer ) returns integer;
local j: integer;
begin
    for j from i + 1 to (t + 1)/2 do begin
        if a_j < a_{t-j+1} then return 0;
        if a_j > a_{t-j+1} then return -1;
    end;
    return 1;
end;

procedure GenBracelets( t, p, r, u, v: integer; RS: boolean );
local rev, i: integer;
begin
    if t − 1 > (n − r)/2 + r then begin
        if a_{t-1} > a_{n-t+2+r} then RS := FALSE;
        else if a_{t-1} < a_{n-t+2+r} then RS := TRUE;
    end;
    if t > n then begin
        if RS = FALSE and n mod p = 0 then PrintIt();
    end
    else begin
        a_t := a_{t-p};
        if a_t = a_1 then v := v + 1;
        else v := 0;
        if u = t − 1 and a_{t-1} = a_1 then  u := u + 1;
        if t = n and u ≠ n and a_n = a_1 then begin end;
        else if u = v then begin
            rev := CheckRev( t, u );
            if rev = 0 then GenBracelets( t + 1, p, r, u, v, RS );
            if rev = 1 then GenBracelets( t + 1, p, t, u, v, FALSE );
        end;
        else GenBracelets(t + 1, p, r, u, v, RS );
        if u = t then u := u − 1;
        for j ∈ {a_{t-p} + 1, . . . , k − 1} do begin
            a_t := j;
            if t = 1 then GenBracelets( t + 1, t, r, 1, 1, RS )
            else GenBracelets( t + 1, t, r, u, 0, RS );
end; end; end;
```

FIG. 3.1. *Bracelet generation algorithm.*

will determine whether or not $a_r \cdots a_n$ is greater than its reversal. This will involve the additional parameter RS to hold intermediate boolean values indicating whether or not the reversal is smaller (RS).

- Reject the string if $a_1 = a_n$ and the string is not equal to a^n (i.e., $t = n$ and $u \neq n$).

The resulting algorithm GenBracelets(t, p, r, u, v, RS) is shown in Figure 3.1. The initial call is GenBracelets(1,1,0,0,0,FALSE). To illustrate this algorithm we trace the parameters as the string 0010023003 gets generated:

α	-	0	0	1	0	0	2	3	0	0	3
t	1	2	3	4	5	6	7	8	9	10	11
p	1	1	1	3	3	3	6	7	7	7	10
r	0	1	2	2	2	5	5	5	5	5	5
u	0	1	2	2	2	2	2	2	2	2	2
v	0	1	2	0	1	2	0	0	1	2	0
RS	F	F	F	F	F	F	F	F	F	F	T

In the following section we give several counting results for strings with no 0^i substring. These results will then be applied when we analyze the algorithm, showing that it runs in constant amortized time.

4. Forbidden substrings. We denote the set of all k-ary strings of length n with no 0^i substring by $\mathbf{I}_k(n,i)$. The cardinality of this set, denoted $I_k(n,i)$, is given by the following recurrence relation:

$$I_k(n,i) = \begin{cases} k^n & \text{if } 0 \leq n < i, \\ (k-1)\sum_{j=1}^{i} I_k(n-j,i) & \text{if } n \geq i\ . \end{cases}$$

It is easy to verify the correctness of this formula. If $n < i$, then the set $\mathbf{I}_k(n,i)$ will contain all k-ary strings. Otherwise, we categorize the strings in $\mathbf{I}_k(n,i)$ by the number of consecutive 0's found at the tail of each string. Since there are $k-1$ choices for the character appearing before this string of 0's, we arrive at the given recurrence relation.

We obtain another recurrence relation by considering a string $\alpha = a_1 \cdots a_{n-1}$ in the set $\mathbf{I}_k(n-1,i)$. If we append a character a_n to α, then the string $a_1 \cdots a_n$ is in $\mathbf{I}_k(n,i)$ as long as $a_{n-i+1} \cdots a_n \neq 0^i$. The number of strings where $a_{n-i+1} \cdots a_n = 0^i$ is exactly equal to $I_k(n-i,i)$. Thus we arrive at a second recurrence relation:

$$I_k(n,i) = \begin{cases} k^n & \text{if } 0 \leq n < i, \\ kI_k(n-1,i) - (k-1)I_k(n-i-1,i) & \text{if } n \geq i\ . \end{cases}$$

LEMMA 4.1. *If $k, i \geq 2$, then*

$$I_k(n,i) \geq \sum_{j=1}^{n-2} I_k(j,i).$$

Proof. The base cases when $n \leq i$ are trivial. If $n > i$, then we induct on n:

$$\begin{aligned} I_k(n,i) &\geq I_k(n-1,i) + I_k(n-2,i) \\ &\geq \sum_{j=1}^{n-3} I_k(j,i) + I_k(n-2,i) \\ &= \sum_{j=1}^{n-2} I_k(j,i). \end{aligned}$$ □

LEMMA 4.2. *If $n > 2$ and $k, i \geq 2$, then*

$$\frac{I_k(n,i)}{n} \geq \frac{I_k(n-1,i)}{n-1}.$$

Proof.

$$\begin{aligned}(n-1)I_k(n,i) &= k(n-1)I_k(n-1,i) - (k-1)(n-1)I_k(n-i-1,i)\\ &\geq nI_k(n-1,i) + (kn-n-k)I_k(n-1,i) - (k-1)(n-1)I_k(n-3,i)\\ &\geq nI_k(n-1,i) + 2(kn-n-k)I_k(n-3,i) - (kn-n-k+1)I_k(n-3,i)\\ &= nI_k(n-1,i) + (kn-n-k-1)I_k(n-3,i)\\ &\geq nI_k(n-1,i). \qquad \square\end{aligned}$$

We now prove a theorem that will be used in the analysis of our bracelet generation algorithm. The proof of the theorem uses the previous two lemmas.

THEOREM 4.3. *If $n > 2$ and $k, i \geq 2$, then*

$$\sum_{j=1}^{n} \frac{1}{j} I_k(j,i) \leq \frac{8}{n} I_k(n,i).$$

Proof.

$$\begin{aligned}\sum_{j=1}^{n} \frac{1}{j} I_k(j,i) &\leq 2 \sum_{j=\lceil n/2 \rceil}^{n} \frac{1}{j} I_k(j,i)\\ &\leq \frac{2}{n} I_k(n,i) + \frac{2}{n-1} I_k(n-1,i) + 2 \sum_{j=\lceil n/2 \rceil}^{n-2} \frac{1}{j} I_k(j,i)\\ &\leq \frac{4}{n} I_k(n,i) + \frac{4}{n} \sum_{j=\lceil n/2 \rceil}^{n-2} I_k(j,i)\\ &\leq \frac{8}{n} I_k(n,i). \qquad \square\end{aligned}$$

5. Analysis of the algorithm. In this section we show that the algorithm GenBracelets for generating bracelets is CAT. We analyze the algorithm by looking at the computation tree and determining the amount of computation done at each node. To get a bound on the size of the bracelet computation tree, we observe the following bounds obtained from (2.1) and (2.2) along with Lemma 4.4 from [12]:

$$L_k(n) \leq \frac{k^n}{n} \leq N_k(n) \leq 2\frac{k^n}{n}.$$

Now using (2.4) we get the following bounds on the number of bracelets:

$$\frac{k^n}{2n} \leq B_k(n) \leq 2\frac{k^n}{n}. \tag{5.1}$$

Since the necklace algorithm GenNecklaces is CAT [2], the size of its computation tree is less than ck^n/n for some constant c. This bound is also true for GenBracelets since its computation tree is smaller than that of GenNecklaces. However, unlike the necklace computation tree, the bracelet computation tree has some nodes that require more than a constant amount of computation. From our algorithm, these nodes are the ones that make a call to CheckRev. Thus, to prove the bracelet generation algorithm GenBracelets is CAT, we must show that the computation performed by all calls to CheckRev is bounded by some constant times the total number of bracelets generated. The task of analyzing this extra computation is divided into the following four subsections.

5.1. Identifying the prenecklaces. From the algorithm, each node that makes a call to CheckRev is a prenecklace of the form a^i or $a^i\gamma a^i$ where the nonempty string γ begins and ends with a character lexicographically greater than a. Note that the length of such prenecklaces is at most $n-1$. Each call to CheckRev results in computation proportional to $(t-2i)/2$, where t is the length of the prenecklace. Since any prenecklace of the form a^i requires only constant computation in CheckRev (because $t=i$), we can restrict our attention to prenecklaces of the form $a^i\gamma a^i$. To simplify this task we consider only the prenecklaces beginning with 0, later using (2.5) to account for the remaining prenecklaces. We also ignore the fact that many of these prenecklaces are never generated by the algorithm (i.e., the prenecklace 002100300 is never generated since the prenecklace 002100 is terminal).

The next series of observations are crucial to the success of the analysis. Notice that the number of prenecklaces of the form $0^i\gamma 0^i$ is less than or equal to the number of prenecklaces of the form $0^i\gamma$. We now group these prenecklaces together according to length. Such strings will have length of at least 2, but not greater than $n-2$. Define the set of all k-ary prenecklaces of length n beginning with 0, ending with a nonzero character, and with no 0^i substring to be $\mathbf{P}'_k(n,i)$. Equivalently, the set $\mathbf{P}'_k(n,i)$ contains all prenecklaces with length n of the form $0^j\gamma$ for $1 \le j < i$. The cardinality of this set is denoted as $P'_k(n,i)$. If we let $E_k(n)$ denote the extra computation that results from all calls made to CheckRev by prenecklaces beginning with 0 (while generating $\mathbf{B}_k(n)$), then we obtain the following bound:

$$E_k(n) \le \sum_{i=2}^{n-2} \frac{n-i}{2} P'_k(n-i,i). \tag{5.2}$$

5.2. Bounding the restricted prenecklaces. In this subsection we find an upper bound for $P'_k(n,i)$ first using restricted Lyndon words, and then in terms of strings with forbidden substrings. Because every prenecklace is obtained as a prefix of a β^* where β is some Lyndon word, we arrive at the formula given in (2.3):

$$P_k(n) = \sum_{j=1}^{n} L_k(j).$$

If we let $L_k(j,i)$ denote the number of Lyndon words of length j with no 0^i substring, then we obtain the following upper bound for $P'_k(n,i)$:

$$P'_k(n,i) \le \sum_{j=1}^{n} L_k(j,i). \tag{5.3}$$

Recall that the number of k-ary strings of length n with no 0^i substring is denoted by $I_k(n,i)$. Using these strings we obtain an upper bound for $L_k(n,i)$.

LEMMA 5.1. *If $n \ge 1$ and $i \ge 1$, then*

$$L_k(n,i) \le \frac{1}{n} I_k(n,i).$$

Proof. Each string counted by $L_k(n,i)$ is a representative of an equivalence class of strings each with n elements. If we add up the elements from each equivalence class we get $nL_k(n,i)$ unique strings each of length n with no 0^i substring. The expression $I_k(n,i)$ counts the total number of strings with length n and no 0^i substring. Therefore $L_k(n,i) \le \frac{1}{n} I_k(n,i)$. ∎

Using the previous lemma and Theorem 4.3 ($n > 2$) we can simplify the upper bound in (5.3). Note that the latter bound is also satisfied when $n = 2$.

$$\begin{aligned} P'_k(n,i) &\le \sum_{j=1}^{n} \frac{1}{j} I_k(j,i) \\ &\le \frac{8}{n} I_k(n,i). \end{aligned}$$

5.3. Converting back to prenecklaces. Using the bound discovered in the previous subsection, we can now substitute back into (5.2) and simplify:

$$\begin{aligned} E_k(n) &\le \sum_{i=2}^{n-2} \frac{n-i}{2} P'_k(n-i,i) \\ &\le 4 \sum_{i=2}^{n-2} I_k(n-i,i). \end{aligned}$$

We now use a clever trick to bound this sum in terms of prenecklaces. Observe that we can insert 0^i1 at the front of each string in $\mathbf{I}_k(n-i,i)$ to obtain a new a set of strings of length $n+1$. Notice that each new string is a unique prenecklace regardless of the parameter i. Thus the number of strings in the union of the sets $\mathbf{I}_k(n-i,i)$ for $i = 2, \ldots, n-1$ is less than $P_k(n+1)$. We can divide this total by $k-1$, since we could have arbitrarily chosen any of $k-1$ characters to insert after 0^i. Thus

$$\begin{aligned} E_k(n) &\le \frac{4}{k-1} P_k(n+1) \\ &\le \frac{4k}{k-1} P_k(n) \\ &\le 8 \sum_{j=1}^{n} L_k(j) \\ &\le 8 \sum_{j=1}^{n} \frac{k^j}{j} \\ &\le 24 \frac{k^n}{n}. \end{aligned} \tag{5.4}$$

The simplification found in (5.4) is valid for $k \ge 2$ and can easily be proved by induction.

5.4. Accounting for all prenecklaces. Because the bound on $E_k(n)$ is only for prenecklaces beginning with 0, we use (2.5) to get an upper bound on the extra computation performed by all prenecklaces. Note that $E_1(n) = 0$.

$$\begin{aligned} ExtraWork &\le \sum_{j=2}^{k} E_j(n) \\ &\le \frac{24}{n} \sum_{j=2}^{k} j^n \\ &\le 48 \frac{k^n}{n}. \end{aligned}$$

From (5.1), the total number of bracelets generated is bounded below by $k^n/2n$. Thus, the running time of the algorithm GenBracelets is proportional to the number of bracelets generated, which proves the following theorem.

THEOREM 5.2. *The k-ary bracelet generation algorithm GenBracelets is CAT.*

Experimentally, the constant is less than 8 where we compare the number of calls to GenBracelets plus the number of iterations of the for loop in CheckRev to the number of bracelets generated.

Acknowledgments. The author would like to thank Frank Ruskey for many helpful discussions in all aspects of this paper, as well as the anonymous referee who suggested a simpler proof for Theorem 3.1.

REFERENCES

[1] L. BATTON, C. BOHUN, A. BONA, K. CHENG, T. DOMAN, J. DREW, R. EDWARDS, S. KUTAY, C. LAFLAMME, D. MCCREA, W. MYRVOLD, F. RUSKEY, J. SAWADA, P. VAN DEN DRIESSCHE, J. VANDER KLOET, AND K. WOOD, *Classification of chemical compound pharmacophore structures*, in Proceedings of the Third PIMS Industrial Problem Solving Workshop, C. Bose, ed., 1999, pp. 83–93.

[2] K. CATTELL, F. RUSKEY, J. SAWADA, C.R. MIERS, AND M. SERRA, *Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF*(2), J. Algorithms, 37 (2000), pp. 267–282.

[3] W. CHEN AND J. LOUCK, *Necklaces, MSS sequences, and DNA sequences*, Adv. Appl. Math., 18 (1997), pp. 18–32.

[4] J.-P. DUVAL, *Factoring words over an ordered alphabet*, J. Algorithms, 4 (1983), pp. 363–381.

[5] P. EMMEL AND R. HERSCH, *Exploring ink spreading*, in Proceedings of the 8th IS & T/SID Color Imaging Conference: Color Science and Engineering, Scottsdale, AZ, 2000, pp. 335–341.

[6] H. FREDRICKSEN AND I.J. KESSLER, *An algorithm for generating necklaces of beads in two colors*, Discrete Math., 61 (1986), pp. 181–188.

[7] H. FREDRICKSEN AND J. MAIORANA, *Necklaces of beads in k colors and k-ary de Bruijn sequences*, Discrete Math., 23 (1978), pp. 207–210.

[8] E.N. GILBERT AND J. RIORDAN, *Symmetry types of periodic sequences*, Illinois J. Math., 5 (1961), pp. 657–665.

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

[10] P. LISONEK, *Computer-Assisted Studies in Algebraic Combinatorics*, dissertation, Johannes Kepler University, Linz, Austria, 1994.

[11] F. RUSKEY, C.D. SAVAGE, AND T. WANG, *Generating necklaces*, J. Algorithms, 13 (1992), pp. 414–430.

[12] F. RUSKEY AND J. SAWADA, *An efficient algorithm for generating necklaces with fixed density*, SIAM J. Comput., 29 (1999), pp. 671–684.

[13] F. RUSKEY AND J. SAWADA, *Generating necklaces and strings with forbidden substrings*, in Computing and Combinatorics, Lecture Notes in Comput. Sci. 1858, D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, eds., Springer-Verlag, New York, 2000, pp. 330–339.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 269–288

DISJUNCTIONS OF HORN THEORIES AND THEIR CORES*

THOMAS EITER†, TOSHIHIDE IBARAKI‡, AND KAZUHISA MAKINO§

Abstract. In this paper, we study issues on disjunctions of propositional Horn theories. In particular, we consider the problems of deciding whether a disjunction of Horn theories is Horn, and, if not, computing a Horn core (i.e., a maximal Horn theory included in this disjunction) and the Horn envelope (i.e., the minimum Horn theory including the disjunction), where a Horn core and the Horn envelope are important approximations of the original theory in artificial intelligence. The problems are investigated for two different representations of Horn theories, namely, for Horn conjunctive normal forms (CNFs) and characteristic models. While the problems are shown to be intractable in general, in the case of bounded disjunctions, we present polynomial time algorithms for testing the Horn property in both representations and for computing a Horn core in the CNF representation. Even in the case of bounded disjunction, no polynomial algorithm exists (unless P=NP) for computing a Horn core in the characteristic model representation. Computing the Horn envelope is polynomial in the characteristic model representation, while it is exponential in the CNF representation, even for bounded disjunction.

Key words. computational issues in artificial intelligence, logic in computer science, Horn theory

AMS subject classifications. 68Q25, 68T27

PII. S0097539799350840

1. Introduction.

Since deduction from a set of propositional clauses is a well-known co-NP-complete problem, different approximation methods for reasoning from a clausal theory Σ have been investigated (e.g., [21, 22, 19, 20, 4, 5, 1]). One of these approaches [21, 22] uses a *greatest Horn lower bound* (also called *Horn core* [15]), which is a maximal Horn theory $\Sigma_c \subseteq \Sigma$, if we view theories as sets of models, and the *least Horn upper bound* (*Horn envelope* [15]), which is the minimal Horn theory $\Sigma_e \supseteq \Sigma$ such that Σ logically implies Σ_e. Note that, in general, different Horn cores may exist, while it is known that the Horn envelope is always unique (e.g., [21, 22, 15]).

Computing Horn envelopes and Horn cores has been investigated in [21, 22, 15, 2, 3, 4, 5, 10, 7, 1]. It has been shown that a Horn core of a theory Σ, represented by a given conjunctive normal form (CNF) φ, is computable in polynomial time with an oracle for NP [2, 3], and that all Horn cores can be generated with polynomial delay (i.e., the time between consecutive outputs is bounded by a polynomial in the input size, and the first (resp., last) output also occurs in polynomial time after (resp., before) the start (resp., halt) of the algorithm) if the theory Σ is given by the set of

*Received by the editors February 2, 1999; accepted for publication (in revised form) January 25, 2001; published electronically July 25, 2001. An extended abstract of this paper appears in *Proceedings of the 9th Annual International Symposium on Algorithms and Computation (ISAAC' 98)*, Korea, 1998 [8]. This research was partially supported by the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan and the Austrian Science Fund under project N Z29-INF. Part of this research was conducted while the first author visited Kyoto University in 1995 and 1998 by the support of the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan grant 06044112.

http://www.siam.org/journals/sicomp/31-1/35084.html

†Institut und Ludwig Wittgenstein Labor für Informationssysteme, Technische Universität Wien, Treitlstraße 3, A-1040 Wien, Austria (eiter@kr.tuwien.ac.at).

‡Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan (ibaraki@i.kyoto-u.ac.jp).

§Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan (makino@sys.es.osaka-u.ac.jp).

its models. However, in the latter setting, computing a maximum (in terms of the numbers of models) Horn core is co-NP-hard [15].

In this paper, we consider the issue of computing Horn cores of the disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i represented either by Horn CNFs or by their *characteristic models* [13]. Characteristic models have been proposed as a model-based alternative to the formula-based theory representation. These two approaches are orthogonal with respect to space requirements, in the sense that one approach sometimes allows for an exponentially smaller representation than the other; see [14, 17]. Observe that a disjunction Σ of Horn theories is in general not Horn. Hence, in particular, it is of interest to know whether Σ is Horn, since in this case the Horn core and the Horn envelope coincide with Σ.

Disjunctions of Horn theories may be encountered in different applications. For example, suppose that two groups have, respectively, formed logical hypotheses about an application domain (the "world"), e.g., relationships between medical tests and diseases, and they believe that the relationships amount to a Horn theory. The hypotheses Σ_1 and Σ_2 of two groups (which are the sets of models) may be obtained from actual and conjectured cases, respectively, i.e., concrete measurement data (obtained by experiments) and data which are believed to be true. Suppose that the hypotheses Σ_1 and Σ_2 are merged. Then, at the logic level, the disjunction, i.e., union $\Sigma = \Sigma_1 \cup \Sigma_2$ describes the merged hypotheses. It is of particular interest to know whether Σ is Horn; if so, then the original hypotheses are compatible in the sense that no further cases have to be adopted in order to preserve the Horn property, which may indicate that the individual hypotheses are sound. However, if Σ is not Horn, then either further cases have to be added to maintain the Horn property (which corresponds to the logical consequence of the new case knowledge), or some of the adopted hypothetical cases have to be abandoned. Applying Occam's razor, it is natural to add or to abandon a minimal set of cases. In the former case, this amounts to finding the Horn envelope of Σ, and in the latter case, to finding a Horn core Π of Σ. In finding a Horn core Π, it may also be asked to add such a constraint because Π includes all actual cases Σ_1. That is, computing a Horn core Π satisfying $\Sigma_1 \subseteq \Pi \subseteq \Sigma$ can be seen as one of the important problems.

Another application concerns inference from knowledge bases. Suppose that Horn theories $\Sigma_1, \Sigma_2, \ldots, \Sigma_l$ represent knowledge bases which are located at different sites $s_1, s_2, \ldots, s_l$, respectively. A formula φ is a logical consequence of all Σ_i, for $i = 1, 2, \ldots, l$, only if φ is a logical consequence of the disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$. Thus, in order to test whether φ is a consequence in all Σ_i, it is equivalent to test whether φ is a consequence of Σ. This may be profitably used if the on-line access to the theories Σ_i at the individual sites is for instance, costly or unreliable. If Σ is stored at a distinguished site s^*, then by accessing this single site the query φ to all knowledge bases can be answered. In the case where Σ is Horn, we can store a Horn CNF (resp., the characteristic set) of Σ, which can be more compact than simply mirroring the disjunction of Horn CNFs (resp., the characteristic sets) of the individual Horn theories Σ_i at s^*, since redundancies can be avoided. For example, if each Σ_i is represented by a Horn CNF $\varphi_i = \psi \wedge (x \vee \overline{y}_i)$, for $i = 1, 2, \ldots, l$, then the Horn CNF $\varphi = \psi \wedge (x \vee \overline{y}_1 \vee \overline{y}_2 \vee \cdots \vee \overline{y}_l)$ represents the disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$. Note that, in this case, storing the individual φ_i requires more than l times space compared to φ.

In this paper, we first address the problem of checking if a disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i, represented either by Horn CNFs or by their characteristic models, is a Horn theory. We show that the problem is in general co-NP-complete for both

representations but is polynomially solvable if l is bounded by a constant. These results indicate that computing a Horn core is difficult in general. The polynomiality follows from syntactical and semantical characterizations of a disjunction of Horn theories.

We next deal with the problem of computing a Horn core Π of Σ satisfying $\Sigma_1 \subseteq \Pi \subseteq \Sigma$. We show that, if l is bounded by some constant, then the problem is polynomially solvable for Horn CNFs, while it is co-NP-hard for the representation by characteristic models, even if $l = 2$. For the formula-based representations, we first present a polynomial time algorithm CORE to compute a Horn core of the disjunction of two Horn theories Σ_1 and Σ_2. Since the algorithm CORE cannot be generalized directly to the case $l\,(\geq 3)$, we develop an algorithm CORE* which computes a Horn core of the disjunction of $l\,(\geq 3)$ Horn theories by making use of algorithm CORE repeatedly. It turns out that algorithm CORE* runs in polynomial time if l is bounded by some constant. On the other hand, we show that computing a Horn core in polynomial time is not possible for the representation by characteristic models, by exhibiting a family of theories Σ_1 and Σ_2 having small characteristic sets, while every Horn core of $\Sigma_1 \cup \Sigma_2$ is exponentially large. We also give structural characterizations of Horn cores of a disjunction of Horn theories.

As for the Horn envelope, we show that it can be computed for the representation by characteristic models in polynomial time but cannot be efficiently computed for Horn CNFs, even if $l = 2$. The negative result follows from the fact that there exist Horn theories Σ_1 and Σ_2 such that their Horn CNFs are small, but the CNF representing the Horn envelope of $\Sigma = \Sigma_1 \cup \Sigma_2$ is exponentially large.

The rest of this paper is organized as follows. In the next section, we recall some basic concepts and introduce notations. In sections 3 and 4, we consider Horn cores for formula-based and model-based representations, respectively. Section 5 considers the Horn envelope. Finally, section 6 discusses related work and concludes the paper. The complexity results of all the above problems are summarized in Table 1 of section 6.

2. Preliminaries. We assume a supply of propositional variables (atoms) x_1, $x_2, \ldots, x_n$, where each x_i evaluates to either 1 (true) or 0 (false). Negated variables are denoted by $\overline{x}_i$. These x_i and $\overline{x}_i$ are called literals. A clause is a disjunction $c = L_1 \vee \cdots \vee L_k$ of literals, while a term is a conjunction $t = L_1 \wedge \cdots \wedge L_h$ of literals. By $P(c)$ and $N(c)$ (resp., $P(t)$ and $N(t)$), we denote the sets of variables occurring positively and negatively in c (resp., t). By $\bot$ (resp.,$\top$) we denote the empty clause (resp., empty term) representing falsity (resp., truth). A formula is composed of literals and the following operators: or ($\vee$), and ($\wedge$), negation ($^-$). In particular, a conjunction of clauses $\varphi = \bigwedge_i c_i$ (resp., a disjunction of terms $\varphi = \bigvee_i t_i$) is called *conjunctive normal form* (CNF) (resp., *disjunctive normal form* (DNF)).

A *model* is a vector $v \in \{0,1\}^n$, whose ith component is denoted by v_i, and a *theory* is any set $\Sigma \subseteq \{0,1\}^n$ of models. We denote by $v \leq w$ the usual componentwise ordering of models, i.e., $v_i \leq w_i$ for all $i = 1, 2, \ldots, n$, where $0 < 1$. By $\min(\Sigma)$ and $\max(\Sigma)$ we denote the sets of minimal and maximal models in Σ under $<$, respectively, where $v \in \Sigma$ is a *maximal* (resp., *minimal*) model in Σ, if there is no $w \in \Sigma$ such that $w > v$ (resp., $w < v$). For $B \subseteq \{1, 2, \ldots, n\}$, we denote by x^B the model v such that $v_i = 1$ for $i \in B$ and $v_i = 0$ for $i \notin B$.

For any formula φ, let $T(\varphi) = \{v \in \{0,1\}^n \mid \varphi(v) = 1\}$ denote the set of models of φ. We say that a formula φ represents a theory Σ if $T(\varphi) = \Sigma$. We sometimes do not distinguish a formula from the theory it represents if no confusion arises. For any formulas φ and ψ, we write $\varphi \leq \psi$ (also $\varphi \models \psi$) if $T(\varphi) \subseteq T(\psi)$ holds. A clause c is

called an *implicate* of a theory Σ if $c(v) = 1$ for all $v \in \Sigma$ (i.e., $T(c) \supseteq \Sigma$); it is *prime* if no proper subclause is an implicate of Σ. Similarly, a term t is called an *implicant* of a theory Σ if $t(v) = 0$ for all $v \notin \Sigma$ (i.e., $T(t) \subseteq \Sigma$)); it is *prime* if no proper subterm is an implicant of Σ.

A theory Σ is *Horn* if $\Sigma = Cl_\wedge(\Sigma)$ holds, where $Cl_\wedge(S)$ is the closure of a theory $S \subseteq \{0,1\}^n$ under componentwise AND (i.e., intersection) of models v, w, denoted by $v \bigwedge w$. Observe that any Horn theory Σ has the least (unique smallest) model, denoted $lm(\Sigma)$, which is given by $\bigwedge_{v\in\Sigma} v$.

A clause c is *Horn* if $|P(c)| \leq 1$, and a CNF φ is *Horn* if it contains only Horn clauses. It is well known that a theory Σ is Horn if and only if it is represented by some Horn CNF (e.g., [18]) and that all prime implicates of a Horn theory are Horn (e.g., [11]). If φ is a Horn CNF and c is any clause, then it is known that $\varphi \leq c$ can be checked in linear time (cf. [6]).

A Horn theory Σ_c is a *Horn core* of a theory Σ if $\Sigma_c \subseteq \Sigma$ holds and no Horn theory Σ' exists such that $\Sigma_c \subset \Sigma' \subseteq \Sigma$. Observe that, in general, Σ has more than one Horn core; e.g., $\Sigma = \{(110), (101)\}$ has two Horn cores $\Sigma_c^1 = \{(110)\}$ and $\Sigma_c^2 = \{(101)\}$. The *Horn envelope* of Σ is the Horn theory $\Sigma_e \supseteq \Sigma$ for which no Horn theory Σ' satisfies $\Sigma_e \supset \Sigma' \supseteq \Sigma$. For the above Σ, we have $\Sigma_e = \{(110), (101), (100)\}$. As is easily seen, the Horn envelope is always unique (cf., e.g., [21, 22, 15]). Let φ be a formula representing a theory Σ, and let φ_c and φ_e be formulas representing a Horn core and the Horn envelope of Σ, respectively. Then φ_c and φ_e are also called a *Horn core* and the *Horn envelope* of φ, respectively.

3. CNF representations. In this section, we deal with the case in which each Horn theory is represented by a Horn CNF.

3.1. Horn property of a disjunction of Horn theories. Our first result deals with the Horn property for a disjunction of Horn CNFs.

THEOREM 3.1. *Given Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$, deciding whether $\varphi = \bigvee_{i=1}^{l} \varphi_i$ is Horn is* co-NP*-complete.*

Proof. The problem is in co-NP, since we can guess and verify models v and w of φ such that $v \bigwedge w$ is not a model of φ in polynomial time.

For the hardness part, we reduce the problem of checking whether a DNF $\psi = \bigvee_{i=1}^{m} t_i$ is a tautology (which is known to be co-NP-complete) to our problem. Note that each t_i can be seen as a Horn CNF φ_i. Let x_{n+1} and x_{n+2} be two fresh variables, and let $t_{m+1} = x_{n+1}$ and $t_{m+2} = x_{n+2}$. Then we claim that $\varphi = \psi \vee t_{m+1} \vee t_{m+2}$ $(= \bigvee_{i=1}^{m+2} t_i)$ is Horn if and only if $\psi \equiv \top$ (and hence $\varphi \equiv \top$). The if-direction is obvious.

For the only-if-direction, suppose $\psi \not\equiv \top$. Then there exists a (nontautological) prime implicate c of ψ. It is then easy to see that $c' = c \vee x_{n+1} \vee x_{n+2}$ is a prime implicate of φ. Since c' is not Horn and all prime implicates of a Horn theory are Horn, it follows that no Horn CNF is equivalent to φ. This proves the result. □

Observe that the reduction proof of the above theorem makes use of an unbounded number of Horn theories. For the case in which l is bounded by some constant k, we have a positive result.

Let

$$\varphi_i = \bigwedge_{j=1}^{m_i} c_{i,j}, \quad i = 1, 2, \ldots, l,$$

where $c_{i,j}$ are Horn clauses. Then it can be easily seen that $\varphi = \bigvee_{i=1}^{l} \varphi_i$ is equivalent

to the CNF

$$\varphi' = \bigwedge_{i_1=1}^{m_1} \bigwedge_{i_2=1}^{m_2} \cdots \bigwedge_{i_l=1}^{m_l} (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l}). \tag{3.1}$$

We now introduce the following set.

$HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$: the set of all Horn clauses c defined by $N(c) = \bigcup_{i=1}^{l} N(c_{i,j_i})$ and $P(c) = P(c_{i,j_i})$ for some i, where all the choices of $j_i \in \{1, 2, \ldots, m_i\}$ for $i = 1, 2, \ldots, l$ are considered under the constraint that the disjunction $c_{1,j_1} \vee c_{2,j_2} \vee \cdots \vee c_{l,j_l}$ of the selected clauses is not a tautology.

Namely, any Horn clause c in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ can be constructed by choosing one clause c_{i,j_i} from each φ_i and then by disjuncting all negative literals in $c' = (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l})$ and a positive literal in c' (more precisely, if some c_{i,j_i} contains no positive literal, then c might have no positive literal). Note that $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ contains all maximal Horn clauses c with $c \leq c'$ for each nontautological clause $c' = (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l})$ in φ' of (3.1).

Example 3.2. Let

$$\varphi_1 = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_2 \vee \overline{x}_5 \vee x_6),$$
$$\varphi_2 = (\overline{x}_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_3 \vee x_6).$$

Then

$$\varphi' = (\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6),$$

where tautological clauses are eliminated. The set HC is given as follows:

$$HC(\varphi_1, \varphi_2) = \{(\overline{x}_1 \vee \overline{x}_3 \vee x_4), (\overline{x}_1 \vee \overline{x}_3 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6),$$
$$(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)\}.$$

The following lemma relates a disjunction of Horn CNFs with clauses from $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$.

LEMMA 3.3. *Let $\varphi_1, \varphi_2, \ldots, \varphi_l$ be Horn CNFs, and let $\varphi = \bigvee_{i=1}^{l} \varphi_i$. Then φ represents a Horn theory if and only if there exists an $S \subseteq HC(\varphi_1, \varphi_2, \ldots.\varphi_l)$ such that $\varphi \equiv \bigwedge_{c \in S} c$.*

Proof. The if-part is obvious because all clauses $c \in HC(\varphi_1, \varphi_2, \ldots.\varphi_l)$ are Horn. For the only-if-part, let φ represent a Horn theory Σ. Recall that φ is equivalent to the CNF φ' of (3.1). Since this φ' also represents the Horn theory Σ and all prime implicates of a Horn theory are Horn, for each nontautological clause $c' = (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l})$ in φ', there exists a Horn prime implicate c of Σ satisfying $\varphi' \leq c \leq c'$. Since $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ contains all maximal Horn clauses c^* with $c^* \leq c'$, some clause c^* in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ satisfies

$$\varphi' (\equiv \varphi) \ \leq \ c \ \leq c^* \ \leq \ c'.$$

Thus, replacing each c' in φ' by such a c^* again produces Σ, which implies the only-if-part. □

THEOREM 3.4. *If l is bounded by a constant k, the problem in Theorem* 3.1 *can be solved in polynomial time.*

Proof. By Lemma 3.3, for each clause c' in φ' of (3.1), we find only a clause c^* in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ such that $\varphi \leq c^* \leq c'$. If every c' has such a c^*, we can conclude that φ is Horn; otherwise, we can conclude that φ is not Horn.

As for the time complexity, since $\varphi \leq c^*$ is equivalent to the condition that $\varphi_i \leq c^*$ holds for all i, and since each φ_i is Horn, $\varphi \leq c^*$ can be checked in polynomial time. Furthermore, if l is bounded by a constant, we have at most a polynomial number of c' and c^*. Thus, the overall time is polynomial. □

3.2. Horn cores of a disjunction of Horn theories. Let us now consider the problem of computing a Horn core. The following proposition, together with Theorem 3.1, implies that this is a difficult problem if l is not bounded by some constant k. In particular, a polynomial time computation of some Horn core is infeasible, as well as the enumeration of Horn cores. As for enumeration problems, we measure their complexity in the combined size of input and output. An algorithm is called of *polynomial total time* (also called *output polynomial*) [12] if its running time is polynomial in the length of input and output.

COROLLARY 3.5. *Unless* P=NP*, there is no polynomial total time algorithm which, given Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$, computes all Horn cores, or any given number $k \geq 1$ of Horn cores, of $\varphi = \bigvee_{i=1}^{l} \varphi_i$.*

Proof. Let us assume that there is a polynomial total time algorithm A for generating all Horn cores (resp., k Horn cores) of φ with polynomial running time $p(I, O)$, where I is the input length and O is the output length. Now in order to solve the tautology problem in polynomial time, we apply this algorithm A to the instance which is used to prove Theorem 3.1. Recall that, in this case, φ is Horn if and only if $\psi \equiv \varphi \equiv \top$. Let us execute A until either (i) it halts or (ii) time $p(I, 1)$ is reached, where the length of $\top$ is considered to be 1, and if A computes k Horn cores the number k is set to 1. In the case of (i), if A outputs exactly one Horn CNF $\top$, then output "Yes"; otherwise, "No." In the case of (ii), output "No," since it implies that the output length is more than 1. Therefore, the tautology problem can be solved in polynomial time, implying P=NP. □

PROPOSITION 3.6. *A theory Σ has a unique Horn core if and only if Σ is Horn.*

Proof. The if-direction is trivial. For the only-if-direction, let Π be an arbitrary Horn core of Σ, and let $v \in \Sigma \setminus \Pi$. Since theory $\{v\}$ is Horn, there exists a Horn core Π' such that $v \in \Pi'$, for which $\Pi' \neq \Pi$ holds. □

COROLLARY 3.7. *Given Horn CNFs φ_1, φ_2, ..., φ_l, deciding whether $\varphi = \bigvee_{i=1}^{l} \varphi_i$ has a unique Horn core is* co-NP*-complete.*

These are rather negative results. However, the proofs do not apply in the case of the disjunction of a small (bounded by a constant) number of Horn theories, as will be seen in the next subsection.

3.2.1. Horn cores of a disjunction of two Horn theories. In this subsection, we describe a polynomial time algorithm which computes a Horn core of the disjunction of two Horn theories. We start with the following lemma showing that any Horn core of a disjunction of Horn CNFs can be represented by a Horn CNF consisting of some clauses from $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$. This is a generalization of the only-if-part of Lemma 3.3.

LEMMA 3.8. *Let $\varphi_1, \varphi_2, \ldots, \varphi_l$ be Horn CNFs, and let ψ be any Horn core of $\varphi = \bigvee_{i=1}^{l} \varphi_i$. Then $\psi \equiv \bigwedge_{c \in S} c$ holds for some subset $S \subseteq HC(\varphi_1, \varphi_2, \ldots . \varphi_l)$.*

Proof. Recall that φ is equivalent to the CNF φ' of (3.1) (which may not be Horn). Since ψ is a Horn core of φ, each nontautological clause c' in φ' is subsumed by some prime Horn implicate c of ψ (i.e., $c \leq c'$). Recall that $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ is

the set of all the maximal Horn clauses subsuming at least one of the clauses in φ'. From the maximality of the clauses in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$, therefore, we have a clause $c^* \in HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ such that $c \le c^* \le c'$. Let ψ^* be the CNF obtained from φ' by replacing each c' by such a c^*. Then $\psi \le \psi^* \le \varphi' (\equiv \varphi)$ holds. Since ψ is a Horn core of φ, it then follows that $\psi \equiv \psi^*$. □

Note that the converse is not true in general; that is, not all Horn CNFs $\psi = \bigwedge_{c \in S} c$, where $S \subseteq HC(\varphi_1, \varphi_2, \ldots.\varphi_l)$, represent Horn cores of φ, even if ψ is a CNF obtained from φ' by replacing each nontautological clause c' in φ' by a clause $c^* \in HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ with $c^* \le c'$. The following example gives such an instance.

Example 3.9. Let φ_1 and φ_2 be given in Example 3.2. Then φ' of (3.1) is written as

$$\varphi' = (\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6),$$

where we exclude the tautological clauses from φ'. Let us consider a Horn CNF

$$\psi = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6).$$

Note that all clauses in ψ are contained in $HC(\varphi_1, \varphi_2)$ (see Example 3.2), and that this ψ is obtained from φ' in the desired way, because $(\overline{x}_1 \vee \overline{x}_3 \vee x_4) \le (\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6)$, $(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5) \le (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)$, and $(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6) \le (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)$. However, this ψ is not a Horn core of $\varphi = \varphi_1 \vee \varphi_2$, since

$$\psi' = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6) \tag{3.2}$$

satisfies $\psi < \psi' \le \varphi' (\equiv \varphi)$. In fact, $\psi \le \psi'$ follows from $\psi \le (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)$, and this combined with $\psi(110111) = 0$ and $\psi'(110111) = 1$ implies $\psi < \psi'$. As will be shown later, ψ' of (3.2) is a Horn core of φ.

Now we give an algorithm that obtains a Horn core of the disjunction of two Horn theories.

ALGORITHM CORE
Input: Horn CNFs $\varphi_1 = \bigwedge_{i=1}^{m_1} c_{1,i}$ and $\varphi_2 = \bigwedge_{j=1}^{m_2} c_{2,j}$.
Output: A Horn core ψ of $\varphi = \varphi_1 \vee \varphi_2$.

Step 1. Set $S := \{c^*_{i,j} = c_{1,i} \vee c_{2,j} \mid c^*_{i,j} \not\equiv \top,\ i = 1, 2, \ldots, m_1,\ j = 1, 2, \ldots, m_2\}$ and

$S_2 := \{c \in S \mid |P(c)| = 2\}$.

For each $c^*_{i,j} \in S$, let $c^1_{i,j}$ (resp., $c^2_{i,j}$) denote the Horn clause c such that

$N(c) = N(c^*_{i,j})$ and $P(c) = P(c_{1,i})$ (resp., $P(c) = P(c_{2,j})$).

(Observe that $HC(\varphi_1, \varphi_2)$ is the set of all clauses $c^1_{i,j}$ and $c^2_{i,j}$.)

Step 2. $S_a := S \setminus S_2$; $S_b := S_2$;

for each $c^*_{i,j} \in S_2$ **do**

if $\varphi_1 \le c^1_{i,j}$ and $\varphi_2 \le c^1_{i,j}$ **then**

begin $S_a := S_a \cup \{c^1_{i,j}\}$; $S_b := S_b \setminus \{c^*_{i,j}\}$ **end**

elseif $\varphi_1 \le c^2_{i,j}$ and $\varphi_2 \le c^2_{i,j}$ **then**

begin $S_a := S_a \cup \{c^2_{i,j}\}$; $S_b := S_b \setminus \{c^*_{i,j}\}$ **end**;

Step 3. Output $\psi := \bigwedge_{c \in S_a} c \wedge \bigwedge_{c^*_{i,j} \in S_b} c^1_{i,j}$.

Example 3.10. Let us apply algorithm CORE to Horn CNFs φ_1 and φ_2 given in Example 3.2. In Step 1, we have $S = \{(\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)\}$ and $S_2 = \{(\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)\}$. A clause

$c^*_{2,1} = (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)$ satisfies the if-statement of Step 2 (i.e., $\varphi_1 \leq c^1_{2,1}$ and $\varphi_2 \leq c^1_{2,1}$ holds for $c^1_{2,1} = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)$); any other clause in S_2 satisfies neither the if- nor the elseif-statement. Thus, Step 3 outputs the Horn CNF of (3.2).

This algorithm runs in polynomial time. Indeed, both implication tests in Step 2 are done in linear time since φ_1 and φ_2 are Horn (cf. [6]), and all other steps are clearly polynomial.

THEOREM 3.11. *Let φ_1 and φ_2 be Horn CNFs. Then, algorithm* CORE *computes a Horn core ψ of $\varphi = \varphi_1 \vee \varphi_2$ in polynomial time. Moreover, $\varphi_1 \leq \psi$ holds.*

Proof. Observe that

$$\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2 \tag{3.3}$$

obviously holds. Indeed, each clause c in S_a of Step 3 is an implicate of φ_1, and each clause $c^1_{i,j} \in S_b$ is subsumed by some clause of φ_1; hence, the first implication holds. The second holds since $\varphi_1 \vee \varphi_2 \equiv \bigwedge_{c^*_{i,j} \in S} c^*_{ij}$ holds by definition, and each clause $c = c^*_{i,j}$ in $\varphi_1 \vee \varphi_2$ is subsumed by some clause in ψ.

We claim that this ψ is a Horn core. Towards a contradiction, suppose that there exists a Horn core ψ^* such that $\psi^* > \psi$. Then, $\psi^* \equiv \bigwedge_{c \in S_a} c \wedge \bigwedge_{c^*_{i,j} \in S_b} c^{h(i,j)}_{i,j}$ must hold for some $h(i,j) \in \{1,2\}$, where S_a and S_b are the ones in Step 3. (The proof is similar to that of Lemma 3.8.) Since $\psi < \psi^*$, it follows that $\psi \leq c^2_{i,j}$ must hold for some $c^*_{i,j} \in S_b$, and hence $\varphi_1 \leq c^2_{i,j}$ by (3.3). On the other hand, since $c^2_{i,j}$ is subsumed by some clause in φ_2, $\varphi_2 \leq c^2_{i,j}$ also holds. However, this means that the clause $c^*_{i,j}$ is removed from S_b in Step 2 and hence $c^*_{i,j} \notin S_b$ in Step 3, which is a contradiction. Consequently, ψ is maximal. □

An analysis of the algorithm CORE reveals that it bears no nondeterminism in computing a Horn core ψ that satisfies $\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2$. This may indicate that a Horn core including φ_1 is unique. This is in fact the case.

PROPOSITION 3.12. *Let Σ_1 and Σ_2 be Horn theories, and let $\Sigma = \Sigma_1 \cup \Sigma_2$. Then, there exists a unique Horn core Π that satisfies $\Sigma_1 \subseteq \Pi \subseteq \Sigma$.*

Proof. We show that for any Horn theories Π_1 and Π_2 such that $\Sigma_1 \subseteq \Pi_1 \subseteq \Sigma$ and $\Sigma_1 \subseteq \Pi_2 \subseteq \Sigma$, it holds that $\Pi' = Cl_\wedge(\Pi_1 \cup \Pi_2)$ is Horn and satisfies $\Pi_1, \Pi_2 \subseteq \Pi' \subseteq \Sigma$; the uniqueness follows from this. Indeed, if Π_1 and Π_2 were different Horn cores of Σ such that $\Sigma_1 \subseteq \Pi_1 \subseteq \Sigma$ and $\Sigma_1 \subseteq \Pi_2 \subseteq \Sigma$, then $\Pi' = Cl_\wedge(\Pi_1 \cup \Pi_2)$ would be a Horn theory such that $\Pi_1, \Pi_2 \subset \Pi' \subseteq \Sigma$. This, however, contradicts that Π_1 and Π_2 are Horn cores of Σ.

Let $w \in \Pi' \setminus (\Pi_1 \cup \Pi_2)$, and we show that $w \in \Sigma$. Note that w is of the form $w = \bigwedge_{u \in S_1} u \wedge \bigwedge_{v \in S_2} v$, where $S_1 \subseteq \Pi_1$ and $S_2 \subseteq \Pi_2$. Since Π_1 and Π_2 are Horn, $u' = \bigwedge_{u \in S_1} u \in \Pi_1$, $v' = \bigwedge_{v \in S_2} v \in \Pi_2$, and $w = u' \bigwedge v'$. Since $\Pi_1, \Pi_2 \subseteq \Sigma_1 \cup \Sigma_2$, the following four cases may occur: (1) $u', v' \in \Sigma_1$, (2) $u', v' \in \Sigma_2$, (3) $u' \in \Sigma_1$, $v' \in \Sigma_2$, and (4) $u' \in \Sigma_2$, $v' \in \Sigma_1$. Clearly, $w = u' \wedge v' \in \Sigma$ holds for case (1) or (2). As for case (3), $u' \in \Pi_2$ holds by $\Sigma_1 \subseteq \Pi_2$. This implies that $u', v' \in \Pi_2$, and hence $w \in \Pi_2 \subseteq \Sigma$ holds. Case (4) is similar to (3). □

From Proposition 3.12, we call the two Horn cores Π_1 and Π_2 satisfying $\Sigma_1 \subseteq \Pi_1 \subseteq \Sigma_1 \cup \Sigma_2$ and $\Sigma_2 \subseteq \Pi_2 \subseteq \Sigma_1 \cup \Sigma_2$, respectively, the *canonical Horn cores* of Σ_1 and Σ_2 with respect to $\Sigma = \Sigma_1 \cup \Sigma_2$ and denote them by $can(\Sigma_1, \Sigma)$ and $can(\Sigma_2, \Sigma)$, respectively.

Observe that the generalization of Proposition 3.12 to a disjunction of k (≥ 3) Horn theories does not hold. To see this, consider $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ and assume that $\Sigma_1 = \emptyset$ holds. Then, if Σ_1 had the unique Horn core Π such that $\Sigma_1 \subseteq \Pi \subseteq \Sigma$ (=

$\Sigma_2 \cup \Sigma_3$), then this Π would be the unique Horn core of $\Sigma_2 \cup \Sigma_3$. Clearly, this conflicts with Proposition 3.6.

Finally, we have the following structural result.

PROPOSITION 3.13. *Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_l$ be Horn theories. Then $\Delta = \bigcap_{i=1}^{l} \Sigma_i$ is contained in every Horn core of $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$.*

Proof. Let Π be any Horn core of Σ. We show that $\Omega = Cl_\wedge(\Pi \cup \Delta)$ is contained in Σ; this completes the proof, since the maximality of Π then implies $\Delta \subseteq \Pi = \Omega$.

Take any model $w \in \Omega$. Since Π and Δ are Horn, we need only to consider the case of $w = u \bigwedge v$ for some $u \in \Pi$ and $v \in \Delta$. (The same reasoning is used in the proof of Proposition 3.12.) Now, $u \in \Sigma_i$ holds for some i. Since $v \in \Delta$ is also contained in this Σ_i and Σ_i is Horn, it holds that $w \in \Sigma_i$. Hence, $w \in \Sigma$ holds, implying $\Omega \subseteq \Sigma$. □

3.2.2. Horn cores of a disjunction of more than two Horn theories. In this subsection we develop an algorithm to compute a Horn core of a general disjunction of $\varphi = \varphi_0 \vee \varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_l$ of Horn CNFs φ_i. Let us first show that the direct generalization of algorithm CORE does not produce a Horn core for the case $l \geq 2$.

Example 3.14. Let

$$\begin{aligned}
\varphi_0 &= (\overline{x}_1 \vee \overline{x}_2 \vee x_4)(\overline{x}_2 \vee x_7)(\overline{x}_3 \vee x_5 \vee \overline{x}_7),\\
\varphi_1 &= (\overline{x}_3 \vee x_5),\\
\varphi_2 &= (\overline{x}_1 \vee \overline{x}_3 \vee x_6),
\end{aligned}$$

and let $\varphi = \varphi_0 \vee \varphi_1 \vee \varphi_2$. Then φ is equivalent to the CNF

$$\begin{aligned}
\varphi' = {} & (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4 \vee x_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5 \vee x_6 \vee x_7)\\
& (\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee x_6 \vee \overline{x}_7).
\end{aligned} \tag{3.4}$$

Since no clause c^* in φ' satisfies $|P(c^*)| = 1$ in this case, the S_b in the beginning of Step 2 of the generalized algorithm CORE becomes the set of all clauses in φ' of (3.4). Thus, in Step 3, we have

$$\psi = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_7)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

However, this ψ is not a Horn core of φ, since

$$\psi' = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7)$$

satisfies $\psi < \psi' \leq \varphi' (\equiv \varphi)$. We will show in Example 3.16 that this ψ' is in fact a Horn core of φ.

However, by using algorithm CORE repeatedly, we can get an algorithm to compute a Horn core of $\varphi = \vee_{i=0}^{l} \varphi_i$. This algorithm is polynomial if l is bounded by a constant k. Informally, it constructs a sequence of nondecreasing Horn theories $\psi_0 \leq \psi_1 \leq \psi_2 \leq \cdots$ which is contained in φ. The sequence converges to ψ^*, which is a Horn core of φ.

Let $\varphi_0, \varphi_1, \ldots, \varphi_l$ be Horn CNFs, and let CORE(μ_1, μ_2) denote a Horn CNF of the canonical Horn core $can(\mu_1, \mu_1 \vee \mu_2)$. We now define the following sequence ψ_k, $k \geq 0$:

$$
\begin{aligned}
\psi_0 &= \varphi_0;\\
\psi_1 &= \mathrm{CORE}(\psi_0, \varphi_1);\\
\psi_2 &= \mathrm{CORE}(\psi_1, \varphi_2);\\
&\vdots \quad \vdots\\
\psi_l &= \mathrm{CORE}(\psi_{l-1}, \varphi_l);\\
&\vdots \quad \vdots\\
\psi_{i\cdot l+j} &= \mathrm{CORE}(\psi_{i\cdot l+j-1}, \varphi_j) \qquad (i \geq 0,\ 1 \leq j \leq l);\\
&\vdots \quad \vdots
\end{aligned}
$$

This sequence ψ_k is monotonically nondecreasing from φ_0 to

$$\psi^* = \lim_{k\to\infty} \psi_k, \tag{3.5}$$

by first trying to increase it in φ_1, then in φ_2 and so on. However, note that ψ_l is not necessarily a Horn core of φ. The reason is that in building ψ_1 from ψ_0, say, some models of φ_1 may have been excluded which now can be added to the ψ_l such that the Horn property is still preserved. To catch this aspect, the algorithm cycles over φ_1, φ_2, ..., φ_l, until no further change is possible. Observe that $\psi^* = \psi_k$ holds for some finite $k \geq 0$ since there exist only finite models. Now we have the next lemma.

LEMMA 3.15. *Given Horn CNFs $\varphi_0, \varphi_1, \ldots, \varphi_l$, the CNF ψ^* as defined in* (3.5) *is a Horn core of $\varphi = \vee_{i=0}^{l} \varphi_i$.*

Proof. By an inductive argument, it is easy to show that every ψ_k is a Horn CNF satisfying $\psi_k \leq \varphi$. Hence, ψ^* is a Horn CNF satisfying $\psi^* \leq \varphi$. Now assuming that ψ^* is not a Horn core of φ, we derive a contradiction.

In this case, by Lemma 4.5 (to be shown later), there exists a model $v \in \Sigma \setminus \Pi^*$ such that $\Pi^* \cup \{v\}$ is Horn, where Π^* and Σ are the theories represented by ψ^* and φ, respectively. This model v satisfies $v \in \Sigma_i$ for some i, where Σ_i is the Horn theory represented by φ_i. Let k^* be the index such that $\psi_{k^*} = \psi^*$. Since ψ^* is the limit, $\psi_{k'} = \psi^*$ holds for all $k' > k^*$. Consider the smallest index $k' > k^*$ such that $\psi_{k'} = \mathrm{CORE}(\psi_{k'-1}, \varphi_i)$. Since $\psi_{k'-1} = \psi^*$ holds and $\Pi^* \cup \{v\}$ is Horn, this implies that ψ^* is not a Horn core of $\psi^* \vee \varphi_i$, a contradiction. □

Based on this lemma, we get the following algorithm.

ALGORITHM CORE*

Input: Horn CNFs $\varphi_i = \bigwedge_{j=1}^{m_i} c_{i,j}$, for $i = 0, 1, \ldots, l$.

Output: A Horn core ψ^* of $\varphi = \varphi_0 \vee \varphi_1 \vee \cdots \vee \varphi_l$.

Step 1. Set $\psi := \varphi_0$; $i := 0$; changes := 0;

Step 2. **while** changes $< l$ **do begin**
  if $i < l$ **then** $i := i + 1$ **else** $i := 1$;
  $\psi_{new} := \mathrm{CORE}(\psi, \varphi_i)$;
  if $\psi < \psi_{new}$ **then**
  begin $\psi := \bigwedge_{c \in HC(\varphi_0, \varphi_1, \ldots, \varphi_l) \text{ s.t. } \psi_{new} \leq c} c$;
    changes := 0;
  end
  else changes := changes + 1;
end{while};

Step 3. Output the Horn CNF $\psi^* = \psi$.

Note that, whenever $\psi_{new} = \mathrm{CORE}(\psi, \varphi_i)$ increases from ψ during the iteration of Step 2, the algorithm replaces it by

$$\psi = \bigwedge_{c \in HC(\varphi_0, \varphi_1, \ldots, \varphi_l) \text{ s.t. } \psi_{new} \leq c} c. \tag{3.6}$$

By Lemma 3.8, this replacement does not affect the correctness of the algorithm. The algorithm halts if no change within l consecutive calls is detected, i.e., $\mathrm{CORE}(\psi, \varphi_i) = \mathrm{CORE}(\psi, \varphi_{i+1}) = \cdots = \mathrm{CORE}(\psi, \varphi_{i-1})$ holds, where $i \geq 1$ and we interpret $i - 1 = l$ if $i = 1$ (arithmetically, the last index in the sequence is $(l + i - 2 \bmod l) + 1$).

Example 3.16. Let us apply the algorithm CORE* to the disjunction $\varphi = \varphi_0 \vee \varphi_1 \vee \varphi_2$, where φ_0, φ_1 and φ_2 are given in Example 3.14.

In the first iteration of Step 2, by calling algorithm CORE described in subsection 3.2.1, we have

$$\psi_{new} = \mathrm{CORE}(\varphi_0, \varphi_1) = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

Based on

$$\begin{aligned} HC(\varphi_0, \varphi_1, \varphi_2) = \{&(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_6), \\ &(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_7), (\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7), (\overline{x}_1 \vee \overline{x}_3 \vee x_6 \vee \overline{x}_7)\}, \end{aligned}$$

we have

$$\psi_1 = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

In the second iteration, we have $\psi_{new} = \mathrm{CORE}(\psi_1, \varphi_2) = \psi_1$, and hence $\psi_2 = \psi_1$ holds. Since $l\,(= 2)$ consecutive members ψ_1 and ψ_2 satisfy $\psi_1 = \psi_2$, the algorithm outputs ψ_1 and halts.

THEOREM 3.17. *Algorithm* CORE* *outputs a Horn core* ψ *of* $\varphi = \varphi_0 \vee \varphi_1 \vee \cdots \vee \varphi_l$ *satisfying* $\varphi_0 \leq \psi \leq \varphi$. *Moreover, it runs in polynomial time if* l *is bounded by some constant.*

Proof. The correctness of the algorithm is established by Lemma 3.15. Notice that in the algorithm, each $\psi_{new} = \mathrm{CORE}(\psi, \varphi_i)$ is replaced by the increased Horn theory ψ of (3.6) for which $\psi_{new} \leq \psi \leq \varphi$ holds. The sequence ψ also converges to ψ^* of (3.5).

It thus remains to show that the algorithm is polynomial in l. Observe that each CNF ψ (except for the initial φ_0) contains only clauses from $HC(\varphi_0, \varphi_1, \ldots, \varphi_l)$. There are at most $l \cdot \Pi_{i=0}^{l} m_i$ clauses in $HC(\varphi_0, \varphi_1, \ldots, \varphi_l)$, where m_i is the number of clauses in φ_i. Clearly, for a constant l, this number is polynomial in the input size of $\varphi_0, \varphi_1, \ldots, \varphi_l$. Moreover, since $T(\psi)$ (i.e., its set of models) strictly increases in every l iterations, the set of clauses $HC(\varphi_0, \varphi_1, \ldots, \varphi_l)$ which are logically entailed by ψ monotonically decreases if ψ changes. Thus, the number of iterations is bounded by $l \cdot |HC(\varphi_0, \varphi_1, \ldots, \varphi_l)|$, which is polynomial if l is bounded by a constant. Since a call to $\mathrm{CORE}(\psi, \varphi_i)$ is done in polynomial time by Theorem 3.11, this implies that CORE* requires polynomial time. □

Remarks. (1) To keep ψ of (3.6) small, we may further replace it by a prime CNF or an irredundant prime CNF, where a CNF ψ is called *prime* if it contains only prime implicates of the function it represents and *irredundant* if no clause can be removed from ψ while preserving logical equivalence. Note that an (irredundant) prime CNF for any Horn CNF ψ is computable in quadratic time [11].

(2) We can use CORE* as a reasonable "any time algorithm" for computing an approximation of some Horn core. In fact, we can interrupt the algorithm at any step, and the contents of ψ is a Horn theory that implies φ. The approximation improves with iterations and eventually yields a Horn core.

4. Characteristic models. For a Horn theory Σ, a model $v \in \Sigma$ is called *characteristic* [13] if $v \notin Cl_\wedge(\Sigma \setminus \{v\})$ holds. The set of all characteristic models of Σ, the *characteristic set of* Σ, is denoted by $C^*(\Sigma)$. Note that every Horn theory Σ has the unique characteristic set $C^*(\Sigma)$ and that $\max(\Sigma) \subseteq C^*(\Sigma)$. For example, a Horn theory $\Sigma = \{(0101), (1001), (1000), (0001), (0000)\}$ has $C^*(\Sigma) = \{(0101), (1001), (1000)\}$ and $\max(\Sigma) = \{(0101), (1001)\}$. Given the characteristic set $C^*(\Sigma)$ of a Horn theory Σ, and an arbitrary vector v, it can be checked in polynomial time whether $v \in \Sigma$ holds or not [13].

In the rest of this section, we reconsider the problems in the previous section, assuming that Horn theories are now represented by their characteristic models.

4.1. Horn property of a disjunction of Horn theories. Like in the case of CNFs, deciding whether a disjunction of Horn theories is Horn is intractable in general.

THEOREM 4.1. *Given characteristic sets $C^*(\Sigma_1)$, $C^*(\Sigma_2)$, ..., $C^*(\Sigma_l)$ of Horn theories Σ_1, Σ_2, ..., Σ_l, respectively, deciding whether $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ is Horn is* co-NP*-complete.*

Proof. The membership part of co-NP is similar to the proof of Theorem 3.1.

The hardness part is also similar to the proof in Theorem 3.1. Observe that in this reduction, each Horn theory Σ_i is represented by a single term t_i, and hence it has a small number of characteristic models.

For a term t, let $\Sigma(t)$ denote the theory represented by t, i.e., $\Sigma(t) = T(t)$. Then the characteristic set of $\Sigma(t)$ is represented by

$$C^*(\Sigma(t)) = \{x^{V \setminus A} \mid N \subseteq A \subseteq V \setminus P, |A \setminus N| \leq 1\},$$

where a term t has positive (resp., negative) literals whose indices are from $P \subseteq V = \{1, 2, \ldots, n\}$ (resp., $N \subseteq V$); recall that x^B is the model v such that $v_i = 1$ for $i \in B$ and $v_i = 0$ for $i \notin B$. Informally, $C^*(\Sigma(t))$ contains all models of t in which at most one of the variables not occurring in t is set to 0 and all other such variables are set to 1. Clearly, $C^*(\Sigma(t))$ is constructible from t in polynomial time. Hence, the DNFs in the proof of Theorem 3.1 can be transformed in polynomial time into equivalent theories Σ_i, $i = 1, 2, \ldots, l$, and $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$, which are given by $C^*(\Sigma_i)$ and $C^*(\Sigma)$, respectively. Thus the argument in the proof of Theorem 3.1 proves the result. ∎

The next result follows from Theorem 4.1, where its proof is similar to that of Corollary 3.5.

COROLLARY 4.2. *Unless P=NP, there is no polynomial total time algorithm which, given a positive integer k and characteristic sets $C^*(\Sigma_1)$, $C^*(\Sigma_2)$, ..., $C^*(\Sigma_l)$ of Horn theories Σ_1, Σ_2, ...,$\Sigma_l \subseteq \{0,1\}^n$, computes k Horn cores of $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$.*

The next result gives a precise semantical characterization of the Horn property of a disjunction.

LEMMA 4.3. *Let $M_i \subseteq \{0,1\}^n$, $i = 1, 2, \ldots, l$, be sets of models. Then $\Sigma = \bigcup_{i=1}^{l} Cl_\wedge(M_i)$ is Horn if and only if*

$$\bigwedge_{v \in S} v \in \Sigma \quad \textit{holds for every } S \subseteq \bigcup_{i=1}^{l} M_i \textit{ such that } 1 \leq |S| \leq l. \tag{4.1}$$

Proof. Since a Horn theory is closed under intersection, the only-if-part is obvious. For the if-part, we show by induction on $k \geq l$ that (4.1) implies

$$\bigwedge_{v \in S} v \in \Sigma \text{ holds for every } S \subseteq \bigcup_{i=1}^{l} M_i \text{ such that } 1 \leq |S| \leq k, \tag{4.2}$$

which clearly implies that $\Sigma = \bigcup_{i=1}^{l} Cl_\wedge(M_i)$ is Horn. For the base $k = l$, the statement holds by (4.1). For the induction step, assume that (4.2) holds for some $k \geq l$ and consider the case $k+1$.

Take a set $S \subseteq \bigcup_{i=1}^{l} M_i$ such that $|S| = k+1$ arbitrarily and consider the model $w = \bigwedge_{v \in S} v$. For each $S_j \subseteq S$ such that $|S_j| = |S| - 1$, we denote $u^{(j)} = \bigwedge_{v \in S_j} v$. All $u^{(j)}$ belong to Σ by the induction hypothesis, and there are $k+1$ such S_j. If these contain different S_{j_1} and S_{j_2} such that $u^{(j_1)} = u^{(j_2)}$, then $w = u^{(j_1)} \wedge u^{(j_2)} = u^{(j_1)}$ holds, and $w \in \Sigma$ follows from the induction hypothesis. Otherwise, there are $k+1 \geq l+1$ different models $u^{(1)}, u^{(2)}, \ldots, , u^{(k+1)}$. Hence, by the pigeonhole principle, $u^{(1)}$ and $u^{(2)}$, say, are both contained in $Cl_\wedge(M_{i^*})$ for some i^*. Since $w = u^{(1)} \bigwedge u^{(2)} \in Cl_\wedge(M_{i^*})$ by definition, we have $w \in \Sigma$ from the induction hypothesis. □

As an immediate consequence, we obtain the following result.

THEOREM 4.4. *If l is bounded by a constant, the problem in Theorem* 4.1 *can be solved in polynomial time.*

In fact, the proof shows this result not only for sets of characteristic models but for any sets of models representing Horn theories as in Lemma 4.3.

Theorems 4.1 and 4.4, respectively, are the characteristic model versions of Theorems 3.1 and 3.4.

4.2. Computing Horn cores of a disjunction of Horn theories. Assuming Horn CNF representations, we have presented in section 3 a polynomial time algorithm for computing a Horn core ψ satisfying $\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2$. For characteristic models, a similar algorithm is hard to find. This is suggested by the following theorem (proved after Lemma 4.5), which implies that recognizing a Horn core is intractable, even for the case of two Horn theories.

LEMMA 4.5. *Let Σ_1 be a Horn theory and Σ_2 an arbitrary theory. Then, Σ_1 is not a Horn core of $\Sigma = \Sigma_1 \cup \Sigma_2$ if and only if there exists some model $v \in \Sigma_2 \setminus \Sigma_1$ such that $v \bigwedge w \in \Sigma_1 \cup \{v\}$ holds for all $w \in C^*(\Sigma_1)$.*

Proof. Suppose Σ_1 is not a Horn core, but Π is a Horn core such that $\Sigma_1 \subset \Pi \subseteq \Sigma_1 \cup \Sigma_2$. Let v be any minimal model in $\Pi \setminus \Sigma_1$. Then $v \bigwedge w \in \Pi$ holds for all $w \in C^*(\Sigma_1)$. By the minimality, however, this implies $v \bigwedge w \in \Sigma_1 \cup \{v\}$. Conversely, if there is a model v satisfying $v \bigwedge w \in \Sigma_1 \cup \{v\}$ for all $w \in C^*(\Sigma_1)$, then $v \bigwedge w \in \Sigma_1 \cup \{v\}$ holds for all $w \in \Sigma_1$, since each $w \in \Sigma_1$ can be represented by $w = \bigwedge_{u \in S} u$ for some $S \subseteq C^*(\Sigma_1)$ and $v \bigwedge u \in \Sigma_1$ for all $u \in S$. Hence, $\Sigma_1 \cup \{v\}$ is Horn by Lemma 4.3, which means that Σ_1 is not a Horn core. □

THEOREM 4.6. *Given characteristic sets $C^*(\Sigma_1)$ and $C^*(\Sigma_2)$ of Horn theories $\Sigma_1, \Sigma_2 \subseteq \{0,1\}^n$, respectively, deciding whether Σ_1 is a Horn core of $\Sigma = \Sigma_1 \cup \Sigma_2$ is co-NP-complete.*

Proof. By Lemma 4.5, Σ_1 is not a Horn core if and only if some $v \in \Sigma_2 \setminus \Sigma_1$ exists such that $v \bigwedge w \in \Sigma_1 \cup \{v\}$ holds for every $w \in C^*(\Sigma_1)$. Such a v can be guessed and verified in polynomial time; hence the problem is in co-NP.

We then prove the co-NP-hardness by a reduction from 3SAT [9]. For a given 3-CNF formula $\varphi = \bigwedge_{i=1}^{m} c_i$ on n variables $x_1, x_2, \ldots, x_n$, we now define polynomially computable sets M_1 and M_2 of models in $\{0,1\}^{2n+m+1}$ such that $M_1 = C^*(\Sigma_1)$,

$M_2 = C^*(\Sigma_2)$ for Horn theories Σ_1 and Σ_2, and Σ_1 is not a Horn core of $\Sigma = \Sigma_1 \cup \Sigma_2$ if and only if φ is satisfiable.

Without loss of generality, we assume that all literals in $L = \{x_j, \overline{x}_j \mid 1 \leq j \leq n\}$ appear in φ. Obviously, this restriction on φ does not affect the NP-completeness of 3SAT. Define $V = V_L \cup V_C \cup V_T$, where

$$\begin{aligned} V_L &= \{1, 2, \ldots, n, \overline{1}, \overline{2}, \ldots, \overline{n}\}, \\ V_C &= \{n+1, n+2, \ldots, n+m\}, \\ V_T &= \{n+m+1\}. \end{aligned}$$

Intuitively, the elements in V_L correspond to the literals in L, the elements i in V_C to the clauses c_i in φ, and $n+m+1$ in V_T is a special tag column. Now we define the instance of our problem as follows: $M_1 = M_{1,1} \cup M_{1,2} \cup M_{1,3}$, where

$$\begin{aligned} M_{1,1} &= \{x^{(V_C \setminus \{n+i\}) \cup (V_L \setminus \{q\})} \mid n+i \in V_C,\, q \in c_i\}, \\ M_{1,2} &= \{x^{(V_L \setminus \{j, \overline{j}\}) \cup V_T} \mid 1 \leq j \leq n\}, \\ M_{1,3} &= \{x^{(V_L \setminus \{j, \overline{j}, q\}) \cup V_T} \mid 1 \leq j \leq n,\, q \in V_L \setminus \{j, \overline{j}\}\}, \end{aligned}$$

and $M_2 = M_{2,1} \cup M_{2,2}$, where

$$\begin{aligned} M_{2,1} &= \{x^{(V_L \setminus \{q\}) \cup V_T} \mid q \in V_L\}, \\ M_{2,2} &= \{x^{V_L \setminus \{j, \overline{j}\}} \mid 1 \leq j \leq n\}. \end{aligned}$$

Intuitively, each model in $M_{1,1}$ corresponds to the selection of a literal q from a clause c_i. The intersection of such models after choosing at least one model for each clause c_i corresponds to a choice of one literal from each clause. The models in $M_{1,2}$ and $M_{1,3}$ serve to cover all choices in which two opposite literals are chosen for at least one j. The models in $M_{2,1}$ similarly correspond to the choice of a single literal, but the special component $n+m+1$ is set to 1. The models in $M_{2,2}$ have a similar role as those in $M_{1,2}$.

First, it is not hard to see that $M_i = C^*(M_i)$ holds, for $i = 1, 2$, and hence M_i is indeed the characteristic set of $\Sigma_i = Cl_\wedge(M_i)$. The sets M_1 and M_2 are constructible in polynomial time from φ.

We claim that Σ_1 is not a Horn core of Σ if and only if φ is satisfiable.

Let us first show the only-if-part. Suppose that Σ_1 is not a Horn core of Σ. By Lemma 4.5, there exists some $v \in \Sigma_2 \setminus \Sigma_1$ such that $v \bigwedge w \in \Sigma_1 \cup \{v\}$ holds for all $w \in M_1$. Consider the following two cases.

Case 1. The component v_{n+m+1} is 0. Then, $v = \bigwedge_{w \in S} w$ for some $S \subseteq M_2$ such that $S \cap M_{2,2} \neq \emptyset$. Hence, for some $j \in \{1, 2, \ldots, n\}$, it holds that components j and $\overline{j}$ of v are both 0. Thus, the model v can be generated in Σ_1 by taking the intersection of a model in $M_{1,2}$ and models from $M_{1,1}$ (recall that every literal occurs in some clause). This means $v \in \Sigma_1$, which is a contradiction.

Case 2. The component v_{n+m+1} is 1. Then, v must be the intersection of some models in $M_{2,1}$. Moreover, for each $j = 1, 2, \ldots, n$, at least one of v_j and $v_{\overline{j}}$ is 1; indeed, any model $v \in \Sigma_2$ with $v_{n+m+1} = 1$ and $v_j = v_{\overline{j}} = 0$ for some j can be generated by intersections of models in $M_{1,2} \cup M_{1,3}$, which would again imply $v \in \Sigma_1$, a contradiction. Now, by assumption, $v \bigwedge w \in \Sigma_1 \cup \{v\}$ holds for all $w \in M_1$. By choosing a model $w \in M_{1,1}$ suitably, we obtain the vector $u = v \bigwedge w$ which coincides with v on V_L and has all other components (including $n+m+1$) being 0. This implies

$u \in \Sigma_1$, and hence u can be generated in Σ_1 only by the intersection of at least m models from $M_{1,1}$ including $v^{(1)}, v^{(2)}, \ldots, v^{(m)}$ such that $v^{(i)}_{n+i} = 0$ holds for all i. By the definition of $M_{1,1}$, this corresponds to a choice of one literal from each clause c_i such that no opposite literals are chosen. It then follows that φ is satisfiable.

For the if-part, suppose that φ is satisfiable. Then, there exists a choice of one literal q_i from each clause c_i, $i = 1, \ldots, m$, such that no opposite literals are chosen. We claim that the model $v = x^B$, where $B = (V_L \setminus \{q_i \mid i = 1, \ldots, m\}) \cup V_T$, satisfies $v \in \Sigma_2 \setminus \Sigma_1$ and $v \bigwedge w \in \Sigma_1$ for all $w \in \Sigma_1$. Hence, by Lemma 4.5, it follows that Σ_1 is not a Horn core of Σ.

Clearly, such a v satisfies $v \in \Sigma_2$ (i.e., by the intersection of models in $M_{2,1}$) and $v \notin \Sigma_1$. Observe that the model v' which results from v by switching component $n + m + 1$ to 0 is in Σ_1 (by the intersection of models in $M_{1,1}$). Hence, if $w \in M_{1,1}$, then $v \bigwedge w = v' \bigwedge w \in \Sigma_1$. On the other hand, if $w \in M_{1,2} \cup M_{1,3}$, then $v \bigwedge w$ (whose component $n + m + 1$ takes value 1) is obtainable as the intersection of models in $M_{1,2} \cup M_{1,3}$; hence, $v \bigwedge w \in \Sigma_1$ also holds. □

Thus, computing a canonical Horn core is presumably difficult in general. We next point out that the difficulty can be avoided if Σ_2 is restricted in the following sense. Call a Horn theory Σ *sparse* if $|\Sigma| \leq p(|C^*(\Sigma)|)$ holds for some polynomial p. Then, based on the next lemma proved in [7], we have the following theorem.

LEMMA 4.7. *Given the characteristic set $C^*(\Sigma)$ of a Horn theory $\Sigma \subseteq \{0,1\}^n$, the models of Σ can be enumerated with $O(n^2|C^*(\Sigma)|)$ time delay.*

THEOREM 4.8. *Given characteristic sets $C^*(\Sigma_1)$ and $C^*(\Sigma_2)$ of Horn theories $\Sigma_1, \Sigma_2 \subseteq \{0,1\}^n$, respectively, computing $C^*(\Pi_1)$ for $\Pi_1 = can(\Sigma_1, \Sigma_1 \cup \Sigma_2)$ is polynomial if Σ_2 is known to be sparse.*

Proof. We can compute $C^*(\Pi_1)$ for $\Pi_1 = can(\Sigma_1, \Sigma_1 \cup \Sigma_2)$ as follows.

(1) Using the algorithm in [7], construct Σ_2 from $C^*(\Sigma_2)$. Set $\Pi := C^*(\Sigma_1)$. (Here $Cl_\wedge(\Pi)$ will eventually become Π_1.)
(2) Using Lemma 4.5, check whether some model $v \in \Sigma_2 \setminus Cl_\wedge(\Pi)$ exists such that $Cl_\wedge(\Pi) \cup \{v\}$ is Horn.
(3) If such a model v is found, then set $\Pi := \Pi \cup \{v\}$ and return to (2).
(4) Compute $C^*(\Pi)$ and output it as the result.

Step (1) can be done in polynomial time by Lemma 4.7, since Σ_2 is sparse. In step (2), $Cl_\wedge(\Pi) \cup \{v\}$ is Horn if and only if $v \bigwedge w \in Cl_\wedge(\Pi) \cup \{v\}$ holds for all $w \in \Pi$ by Lemma 4.3 (with $l = 2$), which can be checked in polynomial time; this avoids the recomputation of $C^*(\Pi)$ in each iteration. Overall, this procedure is polynomial since Σ_2 is sparse. □

Observe that the Horn theory Σ_2 in the proof of Theorem 4.6 is not sparse, and hence the above algorithm requires exponential time.

4.3. Exponential sizes of Horn cores. The intractability of recognizing a Horn core as well as that of computing a canonical Horn core does not immediately rule out the possibility of computing one of the Horn cores in polynomial time. However, our next result implies that a polynomial time algorithm (in the input length) for this task is not possible.

THEOREM 4.9. *For every $n \geq 1$, there exist Horn theories $\Sigma_1, \Sigma_2 \subseteq \{0,1\}^{4n+1}$ such that $|C^*(\Sigma_1)| = |C^*(\Sigma_2)| = 2n$, but every Horn core Π of $\Sigma = \Sigma_1 \cup \Sigma_2$ has size $|C^*(\Pi)|$ not bounded by a polynomial in n.*

Proof. Given a fixed $n\,(\geq 1)$, define two sets of models $S_1, S_2 \subseteq \{0,1\}^{4n+1}$ as follows. Let $V_k = \{(k-1)n + j \mid j = 1, 2, \ldots, n\}$ for $k = 1, 2, 3, 4$, and $V = \cup_{k=1}^4 V_k \cup \{4n+1\} = \{1, 2, \ldots, 4n+1\}$. Observe that $V_1 = \{1, 2, \ldots, n\}$ contains the

first n components, V_2 the next n components, and so on. Then,

$$S_1 = \{x^{V\setminus(V_3\cup\{i,3n+i\})}, x^{V\setminus(V_3\cup\{n+i,3n+i\})} \mid 1 \le i \le n\},$$
$$S_2 = \{x^{V\setminus(V_4\cup\{i,2n+i,4n+1\})}, x^{V\setminus(V_4\cup\{n+i,2n+i,4n+1\})} \mid 1 \le i \le n\}.$$

Informally, each model v in $S_1 \cup S_2$ consists of four blocks $b_1(v), \ldots, b_4(v)$, each of which has n bits (addressed by $V_1, \ldots, V_4$), plus an extra bit v_{4n+1}. The set S_1 contains all models v such that (1) exactly $n+2$ bits are 0, (2) $b_4(v)_i = 0$ (i.e., the ith bit in $b_4(v)$ is 0) and either $b_1(v)_i = 0$ or $b_2(v)_i = 0$ for some $i \in \{1, 2, \ldots, n\}$, and (3) $b_3(v)$ is the zero vector (0,0, …,0) (for short, $\mathbf{0}$). Similarly, S_2 contains all models v such that (1) exactly $n+3$ bits are 0, (2) $b_3(v)_i = 0$ and either $b_1(v)_i = 0$ or $b_2(v)_i = 0$ for some $i \in \{1, 2, \ldots, n\}$, and (3) $b_4(v)$ is $\mathbf{0}$ and $v_{4n+1} = 0$. Intuitively, each $v \in S_1$ represents a choice from two alternatives (either $b_1(v)$ or $b_2(v)$) for assigning 0 to a position i. This is similar for each $v \in S_2$.

Clearly, $|S_1| = |S_2| = 2n$, and $S_l = C^*(S_l)$ holds for $l = 1, 2$, since all the models in S_l are incomparable. Hence S_l is the characteristic set of $\Sigma_l = Cl_\wedge(S_l)$. Note that $\Sigma_1 \cap \Sigma_2 = \emptyset$ holds.

Let

$$H_0 = H_{0,1} \cup H_{0,2}, \tag{4.3}$$

where

$$H_{0,1} = \{x^{I_1\cup I_2} \mid I_1 \subseteq V_1, I_2 \subseteq V_2, I_1 \cap \{j - n \mid j \in I_2\} = \emptyset\},$$
$$H_{0,2} = \{x^{I_1\cup I_2\cup\{4n+1\}} \mid I_1 \subseteq V_1, I_2 \subseteq V_2, I_1 \cap \{j - n \mid j \in I_2\} = \emptyset\},$$

i.e., the set of all models v which have (1) $b_1(v)_i = 0$ or $b_2(v)_i = 0$ for all $i = 1, 2, \ldots, n$, (2) $b_3(v) = b_4(v) = \mathbf{0}$, and (3) v_{4n+1} is set arbitrarily.

Fact 1. Every Horn core Π of $\Sigma = \Sigma_1 \cup \Sigma_2$ satisfies $\Pi \supseteq H_0$.

To prove this fact, we note that $H_0 \subseteq \Sigma$, and, for each $v \in H_0$ and $w \in \Sigma$, it holds that $v \bigwedge w \in H_0$. Let Π be a Horn core of Σ. Then $\Pi \cup H_0 \subseteq \Sigma$ holds, and moreover, by Lemma 4.3 (with $l = 1$), $\Pi \cup H_0$ is Horn. Hence, the maximality of a Horn core implies $H_0 \subseteq \Pi$.

Let us first consider the canonical Horn cores $\Pi_l = can(\Sigma_l, \Sigma)$ for $l = 1, 2$. It follows from Fact 1 that $H_{0,l} \subseteq \Pi_l$ holds. Denote, for any model v,

$$\chi(v) = \{j \in \{1, 2, \ldots, n\} \mid b_1(v)_j = 1 \text{ or } b_2(v)_j = 1\},$$

i.e., $\chi(v)$ contains the positions of 1 in $b_1(v)$ or $b_2(v)$. Then we obtain

$$H_{0,l} = \{v \mid \chi(v) = \{1, 2, \ldots, n\},\ b_3(v) = b_4(v) = \mathbf{0},\ \text{ and }\ v_{4n+1} = l - 1\};$$

hence, $|\max(H_{0,l})|$ (i.e., the number of maximal models in $H_{0,l}$) is obviously exponential in n. We now claim that

$$\max(H_{0,l}) \subseteq C^*(\Pi_l) \quad \text{for } l = 1, 2, \tag{4.4}$$

which implies that $|C^*(\Pi_l)|$ is exponential in n.

We consider only the case of $l = 1$, since the case of $l = 2$ is similar. Assume that some $v \in \max(H_{0,1})$ is not contained in $C^*(\Pi_1)$. Note that $v_{4n+1} = 0$, and furthermore $w_{4n+1} = 1$ holds for every $w \in \Sigma_1$. Hence $C^*(\Pi_1)$ must contain at least one model $u \in \Sigma_2$ such that $u > v$. Since $u \in \Sigma_2$, there exists an index j such

that either $b_1(u)_j = 0$ or $b_2(u)_j = 0$. Let $w \in S_1(\subseteq \Sigma_1)$ be the vector that satisfies $b_1(w)_j = 0$ if $b_1(u)_j = 0$ (resp., $b_2(w)_j = 0$ if $b_2(u)_j = 0$). Then, for $x = u \wedge w$, we have $b_1(x) = b_1(u)$, $b_2(x) = b_2(u)$, $b_3(x) = b_4(x) = \mathbf{0}$, and $x_{4n+1} = 0$. Since $x \in \Sigma$ must hold, it follows that $x \in \Sigma_2$. Thus, $x = \bigwedge_{y \in S} y$ for some $S \subseteq S_2$. Since $b_3(x) = \mathbf{0}$, it follows that either $b_1(x)_j = b_1(u)_j = 0$ or $b_2(x)_j = b_2(u)_j = 0$ holds for all $j = 1, \ldots, n$. However, this implies $u \in H_{0,1}$, which is a contradiction to $v \in \max(H_{0,1})$. Hence (4.4) holds.

As is easily seen, besides Π_1 and Π_2, further Horn cores exist. Proposition 3.12 and Fact 1 imply that for any such Horn core Π, the sets $H_1 = (\Sigma_1 \cap \Pi) \setminus H_0$ and $H_2 = (\Sigma_2 \cap \Pi) \setminus H_0$ are both nonempty. Indeed, if $H_1 = \emptyset$ were true, then $\Sigma_1 \cap \Pi \subseteq H_0$ would hold and Fact 1 would imply $\Pi = H_0 \cup (\Sigma_2 \cap \Pi)$; thus, by Proposition 3.12, $\Pi \subseteq \Pi_2$, which is a contradiction. For $H_2 = \emptyset$, the argument is analogous. Moreover, Π has the following property.

Fact 2. For all $v^{(1)} \in H_1$ and $v^{(2)} \in H_2$, it holds that $v^{(1)} \bigwedge v^{(2)} \in H_0$.

To see this, let $w = v^{(1)} \bigwedge v^{(2)}$. Then $b_3(w) = b_4(w) = \mathbf{0}$ holds, since $v^{(1)} \in \Sigma_1$ and $v^{(2)} \in \Sigma_2$. This together with $w \in \Pi \subseteq \Sigma_1 \cup \Sigma_2$ implies $b_1(w)_j = 0$ or $b_2(w)_j = 0$ for all j, and hence $w \in H_0$.

By Fact 2, $w = v^{(1)} \bigwedge v^{(2)}$ satisfies either $b_1(w)_j = 0$ or $b_2(w)_j = 0$ (or both) for all j. Thus the models $v^{(1)}$ and $v^{(2)}$ satisfy $\chi(v^{(1)}) \cup \chi(v^{(2)}) = \{1, 2, \ldots, n\}$. Since this holds for all such $v^{(1)}$ and $v^{(2)}$, every set $\chi(v^{(1)})$ must include $\{1, 2, \ldots, n\} \setminus \bigcap_{v^{(2)} \in H_2} \chi(v^{(2)})$, and by symmetry every $\chi(v^{(2)})$ must include $\{1, 2, \ldots, n\} \setminus \bigcap_{v^{(1)} \in H_1} \chi(v^{(1)})$. This implies the following fact.

Fact 3. There exists a set $I \subset \{1, 2, \ldots, n\}$ of size $|I| \geq n/2$ such that either (1) $I \subseteq \chi(v^{(1)})$ holds for every $v^{(1)} \in H_1$, or (2) $I \subseteq \chi(v^{(2)})$ holds for every $v^{(2)} \in H_2$.

To see this, let $J = \bigcap_{v^{(1)} \in H_1} \chi(v^{(1)})$. If $|J| \geq n/2$, then (1) clearly holds for $I = J$. On the other hand, if $|J| < n/2$, then we have (2) for $I = \{1, 2, \ldots, n\} \setminus J$, since $\chi(v^{(2)})$ must include I.

Assume that (1) of Fact 3 holds; the case of (2) is similar. Choose $v^{(1)} \in \max(H_1)$ arbitrarily. Then, the maximality of Horn core Π implies that every model $w \in \Sigma_1 \setminus H_0$ such that $\chi(w) = \chi(v^{(1)})$ and w coincides with $v^{(1)}$ on the remaining part (i.e., on $V_3 \cup V_4 \cup \{4n+1\}$) is contained in Π (cf. Facts 1 and 2). There exist $2^{|\chi(v^{(1)})|}$ maximal such models w, each of which chooses 0 either in block $b_1(w)$ or in $b_2(w)$, but not in both, for the positions in $\chi(v^{(1)})$. Since $I \subseteq \chi(v^{(1)})$, at least $2^{|I|}$ models from $\Sigma_1 \setminus H_0$ are contained in Π. Note that these models are maximal in Π. Since $\max(\Pi) \subseteq C^*(\Pi)$, it follows that $|C^*(\Pi)| \geq 2^{|I|} \geq 2^{n/2}$. Thus, the size of $C^*(\Pi)$ is not bounded by a polynomial in n. This proves the theorem statement. □

COROLLARY 4.10. *There is no polynomial time algorithm for computing $C^*(\Pi)$ of any Horn core Π for $\Sigma_1 \cup \Sigma_2$, from the characteristic sets $C^*(\Sigma_1)$, $C^*(\Sigma_2)$ of Horn theories $\Sigma_1, \Sigma_2 \subseteq \{0,1\}^n$, respectively.*

Let us remark that Theorem 4.9 and Corollary 4.10 do not rule out the existence of a polynomial total time algorithm for this problem.

5. Horn envelope of a disjunction of Horn theories. In this section, we briefly discuss the Horn envelope of a disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i, $i = 1, 2, \ldots, l$. For model-based representation, we can compute the Horn envelope in polynomial time. Indeed, the characteristic set of Σ is given by $C^*(M)$, where $M = \bigcup_{i=1}^{l} C^*(\Sigma_i)$. Clearly, $C^*(M)$ is computable from M in polynomial time by removing all models v which are represented by the intersection of other models in M. Thus we have the following theorem.

TABLE 6.1
Complexity results for disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$.

	Horn		*Core*		*Envelope*	
	CNF	C^*	CNF	C^*	CNF	C^*
l is constant	P	P	P	EXP	EXP	P
l is general	co-NPC	co-NPC	co-NPH	EXP	EXP	P

P: polynomial time, co-NPH: co-NP-hard
EXP: exponential time, co-NPC: co-NP-complete

THEOREM 5.1. *Given characteristic sets* $C^*(\Sigma_1)$, $C^*(\Sigma_2)$, ..., $C^*(\Sigma_l)$ *of Horn theories* Σ_1, Σ_2, ..., Σ_l, *respectively, the characteristic set of the Horn envelope of* $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ *can be computed in polynomial time.*

As for the CNF representation, computing the Horn envelope requires, in general, exponential time (and space). This is because any Horn CNF that represents the envelope of Σ may be exponential in the size of Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$ of Σ_1,Σ_2,..., Σ_l, even if $l = 2$. Note that for CNFs representing non-Horn theories, any Horn CNF for the Horn envelope may be exponential even if $l = 1$ (cf., e.g., [22]).

For example, let $\varphi_1 = x_0$ and $\varphi_2 = (\overline{x}_1 \vee \overline{x}_2 \ldots \vee \overline{x}_m) \wedge \bigwedge_{j=1}^{m} (x_j \vee \overline{y}_j)$. Then $\varphi = \varphi_1 \vee \varphi_2$ is equivalent to the CNF $(x_0 \vee \overline{x}_1 \vee \overline{x}_2 \ldots \vee \overline{x}_m) \wedge \bigwedge_{j=1}^{m} (x_0 \vee x_j \vee \overline{y}_j)$. Thus the envelope of φ can be represented by

$$\psi = \bigwedge_{z_1 \in \{x_1, y_1\}} \bigwedge_{z_2 \in \{x_2, y_2\}} \cdots \bigwedge_{z_m \in \{x_m, y_m\}} \left(x_0 \vee \bigvee_{j=1}^{m} \overline{z}_j \right).$$

This CNF has 2^m clauses. Since all clauses in ψ are prime and removing some clause in ψ does not produce the envelope, we have the following theorem.

THEOREM 5.2. *There exist Horn CNFs* φ_i, $i = 1, 2, \ldots, l$, *such that any Horn CNF for the Horn envelope of* $\varphi = \bigvee_{i=1}^{l} \varphi_i$ *has size exponential in the size of* φ_i, $i = 1, 2, \ldots, l$, *even if* $l = 2$. *Hence, the Horn envelope of* φ *cannot be computed in polynomial time.*

6. Related work and conclusion. In this paper, we have considered the problems with disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i. The results for the three problems, i.e., recognition of the Horn property, computation of cores, and computation of envelope, are shown in Table 6.1, where the results are categorized by the representations of theories (i.e., CNFs and characteristic sets) and the values of l (i.e., constant and general).

The computation of a Horn core of a propositional theory Σ has been considered by several authors, and different algorithms have been proposed for different representations; see, e.g., [21, 22, 2, 3, 1, 15]. The papers [21, 22, 2, 3, 1] present algorithms for theories represented by CNFs, which require exponential time in the worst case. It should be noted that these algorithms are not immediately applicable if Σ is given by the disjunction of two Horn CNF formulas φ_1 and φ_2. This is because these algorithms require a CNF formula φ ($\equiv \varphi_1 \vee \varphi_2$) for input, but, since a smallest CNF φ may be exponential in the sizes of φ_1 and φ_2, the resulting procedure requires exponential time (and space) in general.

Concerning the model based representation, our intractability result on computing a Horn core for a disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i represented by their

characteristic sets $C^*(\Sigma_i)$ contrasts with a positive result in [15] that all Horn cores of Σ can be enumerated with polynomial delay if all models of Σ are given for input. Intuitively, this is explained by the fact that $C^*(\Sigma)$ can be a succinct representation of Σ, and eliciting all models from it, as needed by the algorithm in [15], is not feasible in polynomial time in the input size of $C^*(\Sigma)$.

Algorithms for computing the Horn envelope for certain classes of formula representations of a theory Σ are contained in [4, 5, 15, 16, 21, 22]. In particular, [22, 4, 5] explicitly consider formula representations, while [15, 16] implicitly cover formula representations of theories in terms of disjunctions of minterms (i.e., terms such that every variable occurs either positively or negatively in them). The papers [21, 22] suggested a general framework for knowledge compilation, in which the concept of Horn envelope is generalized to the envelope (i.e., least upper bound) of the knowledge base, which is given in a source language, in a target language for compilation.

This approach was also pursued in the papers [4, 5], which made several contributions: [4] presented improvements to [21] and preliminary versions of [22], gave conditions on target clausal languages under which the algorithm in [22] for computing the envelope is correct, by which new classes of theories could be handled, and presented a modified procedure which works for all clausal propositional target languages. The paper [5] presented a new algorithm for computing theory envelopes, which works for a broad subclass of the target languages handled by the algorithm in [21] and can have an exponential increase in efficiency in some cases. Furthermore, it was shown in [5] that the algorithms in [21, 5] for envelope computation in the propositional case can be lifted to the first-order case under certain conditions.

As for the case of computing a Horn CNF for the Horn envelope of a theory represented by a CNF, the algorithms in [21, 22, 4, 5] are either known to be exponential in the worst case (in particular, as shown in [4] the algorithm in [21] uses exponential space) or not polynomial unless P=co-NP. Indeed, a given CNF φ representing a theory Σ is unsatisfiable if and only if any Horn CNF ψ for the Horn envelope of Σ is unsatisfiable, which given ψ can be decided in polynomial time. Moreover, we can also conclude from this that a polynomial total time (output-polynomial) algorithm, i.e., polynomial in the size of the input and the output of any irredundant prime CNF for the envelope, does not exist unless P=co-NP.

We have shown that the characteristic set of the Horn envelope of a disjunction of Horn theories $\Sigma_1, \ldots, \Sigma_l$ can be computed from their characteristic sets $C^*(\Sigma_1)$, $\ldots$, $C^*(\Sigma_l)$ in polynomial time. On the other hand, a Horn CNF for the Horn envelope cannot be computed from Horn CNFs $\varphi_1, \ldots, \varphi_l$ representing $\Sigma_1, \ldots, \Sigma_l$ in polynomial time in general, even if $l = 2$, since it may have exponential size. As for a possible polynomial total time algorithm, we observe that results in [15, 16] imply that such an algorithm—if one exists—is hard to find, even if each Horn theory Σ_i has a single model (which is computable from a Horn CNF φ_i for Σ_i in polynomial time). More precisely, if each φ_i is a minterm, then the problem is easily seen to be polynomially equivalent to the problem SID in [16], which is computing an irredundant prime CNF for the Horn envelope of a theory Horn Σ, given the set $C^*(\Sigma)$ of its characteristic models. By Theorem 1 in [15] and Theorem 9 in [16], this problem is at least as hard computing the transversal hypergraph of a given hypergraph for which no polynomial total time algorithm is known; see [16] for details.

Some problems remain for further work. One issue is on Horn cores when l is bounded by some constant; construct an efficient algorithm for generating all Horn cores for both representations and construct an output-efficient computation of a Horn

core (under a suitable notion) for the characteristic set representation. Another issue is a polynomial total time algorithm for computing the Horn envelope from Horn CNFs.

Acknowledgment. The authors thank the anonymous referees for their helpful and constructive comments which improved the presentation of this paper.

REFERENCES

[1] Y. Boufkhad, *Algorithms for propositional KB approximation*, in Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI, 1998, pp. 280–285.

[2] M. Cadoli, *Semantical and computational aspects of Horn approximations*, in Proceedings of the IJCAI-93, Chambery, France, 1993, pp. 39–44.

[3] M. Cadoli and F. Scarcello, *Semantical and computational aspects of horn approximations*, Artificial Intelligence, 119 (2000), pp. 1–17.

[4] A. del Val, *An analysis of approximate knowledge compilation*, in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, 1995, pp. 830–836.

[5] A. del Val, *Approximate knowledge compilation: The first-order case*, in Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), Portland, OR, 1996, pp. 498–503.

[6] W. Dowling and J. H. Gallier, *Linear-time algorithms for testing the satisfiability of propositional Horn theories*, J. Logic Programming, 3 (1984), pp. 267–284.

[7] T. Eiter, T. Ibaraki, and K. Makino, *Computing intersections of Horn theories for reasoning with models*, Artificial Intelligence, 110 (1999), pp. 57–101.

[8] T. Eiter, T. Ibaraki, and K. Makino, *Disjunctions of Horn theories and their cores*, in Proceedings of the Ninth Annual Symposium on Algorithms and Computation (ISAAC '98), Korea, Lecture Notes in Comput. Sci. 1533, K.-Y. Chwa and O.H. Ibarra, eds., Springer, Berlin, 1998, pp. 49–58.

[9] M. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[10] G. Gogic, C. Papadimitriou, and M. Sideri, *Incremental recompilation of knowledge*, J. Artificial Intelligence Res., 8 (1998), pp. 23–37.

[11] P. Hammer and A. Kogan, *Horn functions and their DNFs*, Inform. Process. Lett., 44 (1992), pp. 23–29.

[12] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.

[13] H. Kautz, M. Kearns, and B. Selman, *Reasoning with characteristic models*, in Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington, D.C., 1993, pp. 34–39.

[14] H. Kautz, M. Kearns, and B. Selman, *Horn approximations of empirical data*, Artificial Intelligence, 74 (1995), pp. 129–245.

[15] D. Kavvadias, C. Papadimitriou, and M. Sideri, *On Horn envelopes and hypergraph transversals*, in Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC-93), Hong Kong, Lecture Notes in Comput Sci. 762, W. Ng., ed., Springer, Berlin, 1993, pp. 399–405.

[16] R. Khardon, *Translating between Horn representations and their characteristic models*, J. Artificial Intelligence Res., 3 (1995), pp. 349–372.

[17] R. Khardon and D. Roth, *Reasoning with models*, Artificial Intelligence, 87 (1996), pp. 187–213.

[18] J. McKinsey, *The decision problem for some classes of sentences without quantifiers*, J. Symbolic Logic, 8 (1943), pp. 61–76.

[19] D. Roth, *On the hardness of approximate reasoning*, Artificial Intelligence, 82 (1996), pp. 273–302.

[20] M. Schaerf and M. Cadoli, *Tractable reasoning via approximation*, Artificial Intelligence, 74 (1995), pp. 249–310.

[21] B. Selman and H. Kautz, *Knowledge compilation using Horn approximations*, in Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), Anaheim, 1991, pp. 904–909.

[22] B. Selman and H. Kautz, *Knowledge compilation and theory approximation*, J. ACM, 43 (1996), pp. 193–224.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 289–305

A FULLY DYNAMIC ALGORITHM FOR RECOGNIZING AND REPRESENTING PROPER INTERVAL GRAPHS*

PAVOL HELL†, RON SHAMIR‡, AND RODED SHARAN‡

Abstract. In this paper we study the problem of recognizing and representing dynamically changing proper interval graphs. The input to the problem consists of a series of modifications to be performed on a graph, where a modification can be a deletion or an addition of a vertex or an edge. The objective is to maintain a representation of the graph as long as it remains a proper interval graph, and to detect when it ceases to be so. The representation should enable one to efficiently construct a realization of the graph by an inclusion-free family of intervals. This problem has important applications in physical mapping of DNA.

We give a near-optimal fully dynamic algorithm for this problem. It operates in $O(\log n)$ worst-case time per edge insertion or deletion. We prove a close lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per operation in the cell probe model with word-size b. We also construct optimal incremental and decremental algorithms for the problem, which handle each edge operation in $O(1)$ time. As a byproduct of our algorithm, we solve in $O(\log n)$ worst-case time the problem of maintaining connectivity in a dynamically changing proper interval graph.

Key words. fully dynamic algorithms, graph algorithms, proper interval graphs, lower bounds

AMS subject classifications. 68Q25, 05C85, 68W40

PII. S0097539700372216

1. Introduction.

A graph G is called an *interval graph* if its vertices can be assigned to intervals on the real line so that two vertices are adjacent in G if and only if their assigned intervals intersect. The set of intervals assigned to the vertices of G is called a *realization* of G. If the set of intervals can be chosen to be inclusion-free, then G is called a *proper interval graph*. Proper interval graphs have been studied extensively in the literature (cf. [7, 16]), and several linear-time algorithms are known for their recognition and realization [3, 4].

This paper deals with the problem of recognizing and representing dynamically changing proper interval graphs. The input is a series of operations to be performed on a graph, where an operation is any of the following: adding a vertex (along with the edges incident to it), deleting a vertex (and the edges incident to it), adding an edge and deleting an edge. The objective is to maintain a representation of the dynamic graph as long as it is a proper interval graph, and to detect when it ceases to be so. The representation should enable one to efficiently construct a realization of the graph. In the *incremental* version of the problem, only addition operations are permitted, i.e., the set of operations includes only the addition of a vertex and the addition of an edge. In the *decremental* version of the problem only deletion operations are allowed.

*Received by the editors May 18, 2000; accepted for publication (in revised form) November 28, 2000; published electronically July 31, 2001. Portions of this paper appeared in the Proceedings of the Seventh Annual European Symposium on Algorithms [8].

http://www.siam.org/journals/sicomp/31-1/37221.html

†School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A1S6 (pavol@cs.sfu.ca). The research of this author was supported by the NSERC.

‡School of Computer Science, Tel Aviv University, Tel Aviv, Israel (rshamir@post.tau.ac.il, roded@post.tau.ac.il). The research of the second author was supported in part by grants from the Ministry of Science, Israel, and by the Israeli Science Foundation of the Israeli Academy for the Sciences and the Arts. The research of the third author was supported by an Eshkol fellowship from the Ministry of Science, Israel.

The motivation for this problem comes from its application to *physical mapping* of DNA [1]. Physical mapping is the process of reconstructing the relative position of DNA fragments, called *clones*, along the target DNA molecule, prior to their sequencing, based on information about their pairwise overlaps. In some biological frameworks the set of clones is virtually inclusion-free—for example, when all clones have similar lengths. (This is the case for instance for cosmid clones.) In this case, the physical mapping problem can be modeled using proper interval graphs as follows. A graph G is built according to the biological data. Each clone is represented by a vertex and two vertices are adjacent if and only if their corresponding clones overlap. The physical mapping problem then translates to the problem of finding a realization of G, or determining that none exists.

Had the overlap information been accurate, the two problems would have been equivalent. However, some biological techniques may occasionally lead to an incorrect conclusion about whether two clones intersect, and additional experiments may change the status of an intersection between two clones. The resulting changes to the corresponding graph are the deletion of an edge, or the addition of an edge. The set of clones is also subject to changes, such as adding new clones or deleting "bad" clones (such as chimerics [18]). These translate into addition or deletion of vertices in the corresponding graph. Thus, we would like to be able to dynamically change our graph, so as to reflect the changes in the biological data, as long as they allow us to construct a map, i.e., as long as the graph remains a proper interval graph.

Several authors have studied the problem of dynamically recognizing and representing various graph families. Hsu [11] has given an $O(m+n\log n)$-time incremental algorithm for recognizing interval graphs. (Throughout, we denote the number of vertices and edges in a graph by n and m, respectively.) Deng, Hell, and Huang [4] have given a linear-time incremental algorithm for recognizing and representing connected proper interval graphs. This algorithm requires that the graph will remain connected throughout the modifications. In both algorithms [11, 4] only vertex additions are handled. Recently, Ibarra [12] devised a fully dynamic algorithm for recognizing chordal graphs which handles each edge operation in $O(n)$ time. He also gave an optimal fully dynamic algorithm for recognizing split graphs, which handles each edge operation in $O(1)$ time.

Our results are as follows: For the general problem of recognizing and representing proper interval graphs we give a fully dynamic algorithm which handles each operation in time $O(d+\log n)$, where d denotes the number of edges involved in the operation. Thus, in case a vertex is added or deleted, d equals its degree, and in case an edge is added or deleted, $d = 1$. Our algorithm builds on the representation of proper interval graphs given in [4]. We prove a close lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per edge operation in the cell probe model of computation with word-size b [20]. It follows that our algorithm is nearly optimal (up to a factor of $O(\log\log n)$). We also give a fast $O(n)$ time algorithm for computing a realization of a proper interval graph given its representation, improving the $O(m+n)$ bound of [4].

For the incremental version of the problem we give an optimal algorithm (up to a constant factor) which handles each operation in time $O(d)$. This generalizes the result of [4] to arbitrary instances. The same bound is achieved for the decremental problem.

As a part of our general algorithm we give a fully dynamic procedure for maintaining connectivity in proper interval graphs. The procedure receives as input a

sequence of operations each of which is a vertex addition or deletion, an edge addition or deletion, or a query whether two vertices are in the same connected component. It is assumed that the graph remains proper interval throughout the modifications, since otherwise our main algorithm detects that the graph is no longer a proper interval graph and halts. We show how to implement this procedure in $O(d + \log n)$ worst-case time per operation involving d edges. In comparison, the best known algorithms for fully dynamic connectivity in general graphs require $O(\log n(\log\log n)^3)$ expected amortized time per edge operation [17], or $O(\log^2 n)$ amortized time per edge operation [10], or $O(\sqrt{n})$ worst-case time per edge operation [5]. Furthermore, we show that the lower bound of Fredman and Henzinger [9] of $\Omega(\log n/(\log\log n + \log b))$ amortized time per edge operation (in the cell probe model with word-size b) for fully dynamic connectivity in general graphs applies also to the problem of maintaining connectivity in proper interval graphs.

The paper is organized as follows: In section 2 we give the basic background and describe our representation of proper interval graphs and the realization it defines. In sections 3 and 4 we present the incremental algorithm. In section 5 we extend the incremental algorithm to a fully dynamic algorithm for proper interval graph recognition and representation. We also derive an optimal decremental algorithm. In section 6 we give a fully dynamic algorithm for maintaining connectivity in proper interval graphs. Finally, in section 7 we prove lower bounds on the amortized time per edge operation of fully dynamic algorithms for recognizing proper interval graphs and for maintaining connectivity in proper interval graphs.

2. Preliminaries. Let $G = (V, E)$ be a graph. We denote its set of vertices also by $V(G)$ and its set of edges also by $E(G)$. For a vertex $v \in V$ we define $N(v) \equiv \{u \in V : (u, v) \in E\}$ and $N[v] \equiv N(v) \cup \{v\}$. Similarly, for a set $S \subseteq V$ we define $N(S) \equiv \cup_{v \in S} N(v)$ and $N[S] = N(S) \cup S$. Let R be an equivalence relation on V defined by uRv if and only if $N[u] = N[v]$. Each equivalence class of R is called a *block* of G. Note that every block of G is a complete subgraph of G. The *size* of a block is the number of vertices in it. Two blocks A and B are *adjacent*, or *neighbors*, in G if some (and hence all) vertices $a \in A, b \in B$, are adjacent in G. A *straight enumeration* of G is a linear ordering Φ of the blocks in G, such that for every block, the block and its neighboring blocks are consecutive in Φ.

Let $\Phi = B_1 < \cdots < B_l$ be a linear ordering of the blocks of G. For any $1 \leq i < j \leq l$, we say that B_i is ordered *to the left of* B_j and that B_j is ordered *to the right of* B_i in Φ. The *out-degree* of a block B with respect to Φ, denoted by $o(B)$, is the number of neighbors of B which are ordered to its right in Φ. A *chordless cycle* is an induced cycle of length greater than 3. A *claw* is an induced $K_{1,3}$ (a 3-degree vertex connected to three 1-degree vertices). A graph is called *claw-free* if it contains no induced claw. For other definitions in graph theory see, e.g., [7].

We now quote some well-known properties of proper interval graphs that will be used in what follows.

THEOREM 2.1 (see [13]). *An interval graph (and in particular a proper interval graph) contains no chordless cycle.*

THEOREM 2.2 (see [19]). *A graph is a proper interval graph if and only if it is an interval graph and is claw-free.*

THEOREM 2.3 (see [4]). *A graph is a proper interval graph if and only if it has a straight enumeration.*

LEMMA 2.4 ("the umbrella property" [14]). *Let Φ be a straight enumeration of a connected proper interval graph G. If A, B, and C are blocks of G, such that*

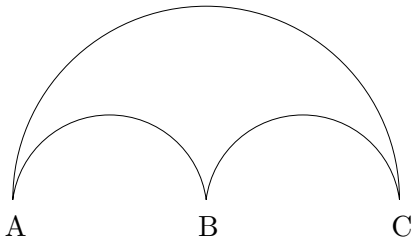


FIG. 2.1. *The umbrella property.*

$A < B < C$ in Φ and A is adjacent to C, then B is adjacent to A and to C (see Figure 2.1*).*

It is shown in [4] that a connected proper interval graph has a unique straight enumeration up to reversal. This motivates our representation of proper interval graphs: For each connected component of the dynamic graph we maintain a straight enumeration. (In fact, for technical reasons we shall maintain both the enumeration and its reversal.) The details of the data structure containing this information will be described in section 3.1.

This information implicitly defines a realization of the dynamic graph (cf. [4]) as follows: Assign to each vertex in block B_i the interval $[i, i + o(B_i) + 1 - \frac{1}{i}]$. We show in section 3.1 how to compute a realization of the dynamic graph from our data structure in time $O(n)$.

3. An incremental algorithm for vertex addition. In the following two sections we describe an optimal incremental algorithm for recognizing and representing proper interval graphs. The algorithm receives as input a series of addition operations to be performed on a graph. Upon each operation the algorithm updates its representation of the graph and halts if the current graph is no longer a proper interval graph. The algorithm handles each operation in time $O(d)$, where d denotes the number of edges involved in the operation. (Thus, $d = 1$ in case of an edge addition, and d is the degree in case of a vertex addition.) It is assumed that initially the graph is empty, or alternatively, that the representation of the initial graph is known. We also show how to compute in $O(n)$ time a realization of a graph given its representation.

A *contig* of a connected proper interval graph G is a straight enumeration of G. The first and the last blocks of a contig are called *end-blocks*, and their vertices are called *end-vertices*. The rest of the blocks are called *inner-blocks*.

As mentioned above, each connected component of the dynamic graph has exactly two contigs (which are reversals of each other) and both are maintained by the algorithm. Each operation involves updating the representation. In the following we concentrate on describing only one of the two contigs for each component. The second contig is updated in a similar way.

3.1. The data structure. We now describe the details of how we keep our representation. The following data is kept and updated by the algorithm:

1. For each vertex we keep the name of the block to which it belongs.
2. For each block we keep the following:
   (a) The *size* of the block.
   (b) Left and right *near pointers*, pointing to nearest neighbor blocks on the left and on the right, respectively.

(c) Left and right *far pointers*, pointing to farthest neighbor blocks on the left and on the right, respectively.
(d) Left and right *self pointers*, pointing to the block itself.
(e) An *end pointer* which is null if the block is not an end-block of its contig, and otherwise, points to the other end-block of that contig.

In the following we shall omit details about the obvious updates to the names of the blocks containing each of the vertices (item 1) and to the block sizes (item 2a).

We introduce self pointers due to the possible need in the course of the algorithm to update many far pointers pointing to a certain block, so that they point to another block. In order to be able to do that in $O(1)$ time we use the technique of *nested pointers*: We make the far pointers point to a *location* whose content is the address of the block to which the far pointers should point. The role of this special location will be served by our self-pointers. The value of the left and right self-pointers of a block B is always the address of B. When we say that a certain left (right) far pointer points to B we mean that it points to a left (right) self-pointer of B. Let A and B be blocks. In order to change all left (right) far pointers pointing to A so that they point to B, we require that no left (right) far pointer points to B. If this is the case, we simply *exchange* the left (right) self-pointer of A with the left (right) self-pointer of B. This means that (1) the previous left (right) self-pointer of A is made to point to B, and the algorithm records it as the new left (right) self-pointer of B; and (2) the previous left (right) self-pointer of B is made to point to A, and the algorithm records it as the new left (right) self-pointer of A.

We shall use the following notation: For a block B we denote its address in the memory by $\&B$. $\&\emptyset$ denotes the null pointer. When we set a far pointer to point to a left or to a right self-pointer of B we shall abbreviate and set it to $\&B$. We denote the left and right near pointers of B by $N_l(B)$ and $N_r(B)$, respectively. We denote the left and right far pointers of B by $F_l(B)$ and $F_r(B)$, respectively. We denote its end pointer by $E(B)$. In the rest of this paper we often refer to blocks by their addresses. For example, if A and B are blocks and $N_r(A) = \&B$, we sometimes refer to B by $N_r(A)$. We define $N_r(\emptyset) = N_l(\emptyset) = F_r(\emptyset) = F_l(\emptyset) = \&\emptyset$. When it is clear from the context, we also use a name of a block to denote any vertex in that block. Given a contig Φ we denote its reversal by Φ^R. In general when performing an operation, we denote the graph before the operation is carried out by G, and the graph after the operation is carried out by G'.

Given this data structure we can compute a realization of a contig C of G as follows: We first rank the blocks of C, starting with the leftmost block. This is done by choosing an arbitrary block of C, and marching up the enumeration of blocks of C using left near pointers, until we reach an end-block. We then set the rank of this block to 1, and march down the enumeration of blocks using right near pointers, until we reach the other end-block. We rank all the blocks of C along the way. Let us denote by $r(B)$ the rank of a block B. Then the out-degree of B is simply $o(B) = r(F_r(B)) - r(B)$, and the interval that we assign to the vertices of B is $[r(B), r(F_r(B)) + 1 - 1/r(B)]$. We conclude with the following theorem.

THEOREM 3.1. *A realization of a proper interval graph, which is represented using the data structure described above, can be computed in time $O(n)$.*

3.2. The impact of a new vertex. In the following we describe the changes made to the representation of the graph in case G' is formed from G by the addition of a new vertex v of degree d. We also give some necessary and some sufficient conditions for deciding whether G' is a proper interval graph.

Let B be a block of G. We say that v is *adjacent* to B if v is adjacent to some vertex in B. We say that v is *fully adjacent* to B if v is adjacent to *every* vertex in B. We say that v is *partially adjacent* to B if v is adjacent to B but not fully adjacent to B.

The following lemmas characterize the adjacencies of the new vertex, assuming that G' is a proper interval graph.

LEMMA 3.2. *If G' is a proper interval graph then v can have neighbors in at most two connected components of G.*

Proof. Suppose to the contrary that x, y, and z are neighbors of v in three distinct components of G. Then v, x, y, and z induce a claw in G', a contradiction. □

LEMMA 3.3 (see [4]). *Let C be a connected component of G containing neighbors of v. Let $B_1 < \cdots < B_k$ be a contig of C. Suppose that G' is a proper interval graph and let $1 \le a < b < c \le k$. Then the following properties are satisfied:*

1. *If v is adjacent to B_a and to B_c, then v is fully adjacent to B_b.*
2. *If v is adjacent to B_b and not fully adjacent to B_a and to B_c, then B_a is not adjacent to B_c.*
3. *If $b = a+1, c = b+1$, and v is adjacent to B_b, then v is fully adjacent to B_a or to B_c.*

One can view a contig Φ of a connected proper interval graph C as a weak linear order $<_\Phi$ on the vertices of C, where $x <_\Phi y$ if and only if the block containing x is ordered in Φ to the left of the block containing y. We say that Φ' is a *refinement* of Φ if either, for every $x, y \in V(C)$, $x <_\Phi y$ implies $x <_{\Phi'} y$ or, for every $x, y \in V(C)$, $x >_\Phi y$ implies $x <_{\Phi'} y$.

LEMMA 3.4. *If H is a connected induced subgraph of a proper interval graph H', Φ is a contig of H, and Φ' is a straight enumeration of H', then Φ' is a refinement of Φ.*

Proof. By induction on the number of additional vertices in H': If $H' = H$ then the claim is obvious. Let $k = |V(H') \setminus V(H)|$. By the induction hypothesis, for a proper interval graph H'' which contains H (as an induced subgraph) and is contained in H', and for which $|V(H'') \setminus V(H)| = k-1$, every straight enumeration is a refinement of Φ. Let C be a connected component of H'' for which $V(C) \supseteq V(H)$, and let Φ''_C be a contig of C. Let C' be a connected component of H' for which $V(C') \supseteq V(H)$ (and therefore $V(C') \supseteq V(C)$), and let Φ'_C be a contig of C'. In [4] it is constructively shown how Φ'_C is obtained as a refinement of Φ''_C (see also section 3.3). Since Φ''_C is a refinement of Φ, the claim follows. □

Note that whenever v is partially adjacent to a block B in G, then the addition of v will cause B to split into two blocks of G', namely, $B \setminus N(v)$ and $B \cap N(v)$. Otherwise, if B is a block of G to which v is either fully adjacent or nonadjacent, then one of B or $B \cup \{v\}$ is a block of G'.

COROLLARY 3.5. *If B is a block of G to which v is partially adjacent, then $B \setminus N(v)$ and $B \cap N(v)$ occur consecutively in a straight enumeration of G'.*

LEMMA 3.6. *Let C be a connected component of G containing neighbors of v. Let $\{B_1, \ldots, B_k\}$ denote the set of blocks in C which are adjacent to v, such that in a contig of C, $B_1 < \cdots < B_k$. If G' is a proper interval graph, then the following properties are satisfied:*

1. *$B_1, \ldots, B_k$ are consecutive in a contig of C.*
2. *If $k \ge 3$ then v is fully adjacent to $B_2, \ldots, B_{k-1}$.*
3. *If v is adjacent to a single block B_1 in C, then B_1 is an end-block.*
4. *If v is adjacent to more than one block in C and has neighbors in another*

component, then B_1 is adjacent to B_k, and one of B_1 or B_k is an end-block to which v is fully adjacent, while the other is an inner-block.

Proof. Claims 1 and 2 follow directly from part 1 of Lemma 3.3. Claim 3 follows from part 3 of Lemma 3.3. To prove the last part of the lemma let us denote the other component containing neighbors of v by D. Examine the induced connected subgraph H of G' whose set of vertices is $V(H) = \{v\} \cup V(C) \cup V(D)$. H is a proper interval graph since it is an induced subgraph of G'. It is composed of three types of blocks: blocks whose vertices are from $V(C)$, which we will henceforth call C-blocks; blocks whose vertices are from $V(D)$, which we will henceforth call D-blocks; and $\{v\}$, which is a block of H, since $H \setminus \{v\}$ is not connected. All blocks of C remain intact in H, except B_1 and B_k, each of which may split into $B_j \setminus N(v)$ and $B_j \cap N(v)$, $j = 1, k$.

Surely, in a contig of H all C-blocks must be ordered completely before or completely after all D-blocks. Let Φ denote a contig of H, in which C-blocks are ordered before D-blocks. Let X denote the rightmost C-block in Φ. By the umbrella property, $X < \{v\}$, and moreover, X is adjacent to v. By Lemma 3.4, Φ is a refinement of a contig of C. Hence, $X \subseteq B_1$ or $X \subseteq B_k$ (more precisely, $X = B_1 \cap N(v)$ or $X = B_k \cap N(v)$). Therefore, one of B_1 or B_k is an end-block.

Without loss of generality, $X \subseteq B_k$. Suppose to the contrary that v is not fully adjacent to B_k. Then by Lemma 3.4 we have $B_{k-1} \cap N(v) < B_k \setminus N(v) < \{v\}$ in Φ (note that these blocks are not consecutive), contradicting the umbrella property. We conclude that v is fully adjacent to B_k. Furthermore, B_1 must be adjacent to B_k, or else G' contains a claw consisting of v and one vertex from each of B_1, B_k, and $V(D) \cap N(v)$. It remains to show that B_1 is an inner-block in C. Suppose it is an end-block. Since B_1 and B_k are adjacent, C consists of a single block, a contradiction. Thus, claim 4 is proved. ∎

3.3. The DHH algorithm. In our algorithm we rely on the incremental algorithm of Deng, Hell, and Huang (DHH) [4]. This algorithm handles the insertion of a new vertex into a connected proper interval graph in $O(d)$ time, changing its straight enumeration appropriately or determining that the new graph is not a proper interval graph. We describe it briefly below. For simplicity, we assume throughout that the modified graph is a proper interval graph.

Let H be a connected proper interval graph, and let v be a vertex to be added, which is adjacent to d vertices in H. Let $\Phi = B_1 < \cdots < B_p$ denote a contig of H. By Lemma 3.6, the blocks to which v is fully adjacent occur consecutively in Φ. Assume that v is fully adjacent to $B_l, \ldots, B_r$, and for clarity we shall consider only the case where $1 < l < r < p$. Let $a = l - 1$ and $c = r + 1$. By Lemma 3.3(2), B_a and B_c are nonadjacent. Let $b > a$ be the largest index such that B_b is adjacent to B_a, and let $d < c$ be the smallest integer such that B_d is adjacent to B_c. It is shown in [4] that $a < b < d < c$.

In order to construct a straight enumeration of the new graph we have to distinguish between two cases:

1. If v is adjacent either to B_a or to B_c, then a straight enumeration of the new graph can be obtained as follows: If v is adjacent to B_a, we split B_a into $B_a \setminus N(v), B_a \cap N(v)$, list them in this order, and add $\{v\}$ as a block just after B_b. If v is adjacent to B_c, we split B_c into $B_c \cap N(v), B_c \setminus N(v)$ in this order, and add $\{v\}$ as a block just before B_d. In case v is adjacent to both B_a and B_c then these two instructions coincide, as shown in [4].
2. If v is adjacent neither to B_a nor to B_c then there are two possibilities: If there exists a block B_j, $b < j < d$, such that B_j is adjacent to both B_l and

B_r, then a straight enumeration is obtained by adding v to B_j. Otherwise, let u be the least integer greater than b such that B_u is adjacent to B_r. Then a straight enumeration is obtained by inserting a new block $\{v\}$ just before B_u.

In section 3.4 we show how to find the sequence of blocks $B_l, \ldots, B_r$ from our data structure in $O(d)$ time. Using near and far pointers we can find, in $O(1)$ time, $B_a = N_l(B_l), B_c = N_r(B_r), B_b = F_r(B_a)$, and $B_d = F_l(B_c)$. If v is adjacent to B_a or to B_c then updating the straight enumeration can be done in $O(1)$ time. Otherwise, finding B_j (if such exists) can be done in $O(d)$ time, and alternatively, finding $B_u = F_l(B_r)$ can be done in $O(1)$ time. Again in this case we can update the straight enumeration in $O(1)$ time. Hence, our data structure supports the insertion of a vertex of degree d in $O(d)$ time, when all its neighbors are in the same connected component.

3.4. Our algorithm. We perform the following upon a request for adding a new vertex v: We make two passes over the neighbors of v. In the first pass we discover all blocks adjacent to v, and for each such block we allocate a *counter* and initialize it to zero. In the second pass, for each neighbor u of v we add one to the counter of the block containing u. We call a block *full* if its counter equals its size, *empty* if its counter equals zero, and *partial* otherwise. In order to find a set of consecutive blocks which contain neighbors of v, we pick arbitrarily a neighbor of v and march up the enumeration of blocks to the left using the left near pointers. We continue till we hit an empty block or till we reach the end of the contig. We do the same to the right and this way we discover a maximal sequence of nonempty blocks in that component which contain neighbors of v. We call this maximal sequence a *segment*. Only the two extreme blocks of the segment are allowed to be partial, or else we fail (by Lemma 3.6(2)).

If the segment we found contains all the neighbors of v then we can use the DHH algorithm in order to insert v into G, updating our internal data structure accordingly. Otherwise, by Lemmas 3.2 and 3.6(1) there could be only one more segment which contains neighbors of v. In that case, exactly one extreme block in each segment is an end-block to which v is fully adjacent (if the segment contains more than one block), and the two extreme blocks in each segment are adjacent, or else we fail (by Lemma 3.6(3, 4)).

We proceed as above to find a second segment containing neighbors of v. We can make sure that the two segments are from two different contigs by checking that their end-blocks do not point to each other. We also check that conditions 3 and 4 in Lemma 3.6 are satisfied for both segments. If the two segments do not cover all neighbors of v, we fail.

If v is adjacent to vertices in two distinct components C and D, then we should merge their contigs. Let $\Phi = B_1 < \cdots < B_k$ and Φ^R be the two contigs of C. Let $\Psi = B'_1 < \cdots < B'_l$ and Ψ^R be the two contigs of D. The way the merge is performed depends on the identity of the end-blocks to which v is adjacent in each segment. If v is adjacent to B_k and B'_1, then by the umbrella property the two new contigs (up to refinements described below) are $\Phi < \{v\} < \Psi$ and $\Psi^R < \{v\} < \Phi^R$. In the following we describe the necessary changes to our internal data structure in case these are the new contigs. The three other cases (e.g., v is adjacent to B_1 and B'_1, etc.) are handled similarly.

- Block enumeration: We merge the two enumerations of blocks and put a new block $\{v\}$ in-between the two contigs. Let the leftmost block which is adjacent to v in the new ordering $\Phi < \{v\} < \Psi$ be B_i, and let the rightmost block

adjacent to v be B'_j. If B_i is partial we split it into two blocks $\hat{B}_i = B_i \setminus N(v)$ and $B_i = B_i \cap N(v)$ in this order. If B'_j is partial we split it into two blocks $B'_j = B'_j \cap N(v)$ and $\hat{B}'_j = B'_j \setminus N(v)$ in this order.

- End pointers: We set $E(B_1) = E(B'_1)$ and $E(B'_l) = E(B_k)$. We then nullify the end pointers of B_k and B'_1.
- Near pointers: We update $N_l(\{v\}) = \&B_k, N_r(\{v\}) = \&B'_1, N_r(B_k) = \&\{v\}$, and $N_l(B'_1) = \&\{v\}$. Let $B_0 = \emptyset$. If B_i was split we set $N_r(\hat{B}_i) = \&B_i, N_l(B_i) = \&\hat{B}_i, N_l(\hat{B}_i) = \&B_{i-1}$, and $N_r(B_{i-1}) = \&\hat{B}_i$. Analogous updates are made to the near pointers of $B'_j, \hat{B}'_j$, and B'_{j+1}, in case B'_j was split.
- Far pointers: If B_i was split we set $F_l(\hat{B}_i) = F_l(B_i), F_r(\hat{B}_i) = \&B_k$, and exchange the left self-pointer of B_i with the left self-pointer of $\hat{B}_i$. If B'_j was split we set $F_r(\hat{B}'_j) = F_r(B'_j), F_l(\hat{B}'_j) = \&B'_1$ and exchange the right self-pointer of B'_j with the right self-pointer of $\hat{B}'_j$. In addition, we set all right far pointers of $B_i, B_{i+1}, \ldots, B_k$ and all left far pointers of $B'_1, \ldots, B'_{j-1}, B'_j$ to $\&\{v\}$ (in $O(d)$ time). Finally, we set $F_l(\{v\}) = \&B_i$ and $F_r(\{v\}) = \&B'_j$.

The algorithm is summarized in Figure 3.1.

Input: A representation of the current graph G and a list of neighbors in G of a new vertex v.
Output: A representation of $G \cup \{v\}$ or a *False* value indicating that $G \cup \{v\}$ is not a proper interval graph.

1. Find the number s of segments of blocks which are adjacent to v.
2. If $s \geq 3$ then return *False*.
3. If $s = 1$ then apply the DHH algorithm.
4. Otherwise, proceed as follows ($s = 2$):
   (a) Check that exactly one extreme block in each segment is an end-block to which v is fully adjacent, and that the two extreme blocks in each segment are adjacent. Otherwise, return *False*.
   (b) Check that the two segments are in distinct contigs. Otherwise, return *False*.
   (c) Update the representation of the graph as described above.

FIG. 3.1. *An incremental algorithm for vertex addition.*

4. An incremental algorithm for edge addition. In this section we show how to handle the addition of a new edge (u, v) in $O(1)$ time. We characterize the cases for which $G' = G \cup \{(u, v)\}$ is a proper interval graph and show how to efficiently detect them and how to update our representation of the graph.

LEMMA 4.1. *If u and v are in distinct connected components in G, then G' is a proper interval graph if and only if u and v are end-vertices in a straight enumeration of G.*

Proof. To prove the "only if" part let us examine the graph $H = G' \setminus \{u\} = G \setminus \{u\}$. H is a proper interval graph as it is an induced subgraph of G. If G' is also a proper interval graph, then by Lemma 3.6(3) v must be an end-vertex in a straight enumeration of G, since u is not adjacent to any other vertex in the component containing v. The same argument applies to u.

To prove the "if" part we give a straight enumeration of the new connected component containing u and v in G'. Denote by C and D the components containing

u and v, respectively.

Let $B_1 < \cdots < B_k$ be a contig of C, such that $u \in B_k$. Let $B'_1 < \cdots < B'_l$ be a contig of D, such that $v \in B'_1$. Then $B_1 < \cdots < B_k \setminus \{u\} < \{u\} < \{v\} < B'_1 \setminus \{v\} < \ldots < B'_l$ is the required straight enumeration. □

By the previous lemma if u and v are in distinct components in G, and G' is a proper interval graph, then they must reside in end-blocks of distinct contigs. We can check that in $O(1)$ time. In case u and v are end-vertices of two distinct contigs, we update our internal data structure as follows:

- Block enumeration: Given in the proof of Lemma 4.1.
- End pointers: We set $E(B_1) = E(B'_1)$ and $E(B'_l) = E(B_k)$. We then nullify the end-pointers of B_k and B'_1.
- Notation: Let $B_0 = \emptyset$ and $B'_{l+1} = \emptyset$. Let $B_k = B_k \setminus \{u\}$ and $B'_1 = B'_1 \setminus \{v\}$. If $B_k \neq \emptyset$, let $x = k$, and otherwise, let $x = k - 1$. If $B'_1 \neq \emptyset$ let $y = 1$, and otherwise, let $y = 2$.
- Near pointers: We set $N_r(\{u\}) = \&\{v\}, N_l(\{u\}) = \&B_x, N_l(\{v\}) = \&\{u\}$, and $N_r(\{v\}) = \&B'_y$. We also update $N_r(B_x) = \&\{u\}$ and $N_l(B'_y) = \&\{v\}$.
- Far pointers: We set $F_l(\{u\}) = F_l(B_k)$ and $F_r(\{v\}) = F_r(B'_1)$. We exchange the right self-pointer of B_k with the right self-pointer of $\{u\}$, and the left self-pointer of B'_1 with the left self-pointer of $\{v\}$. Finally, we set $F_r(\{u\}) = \&\{v\}$ and $F_l(\{v\}) = \&\{u\}$.

It remains to handle the case where u and v are in the same connected component C in G. If $N(u) = N(v)$, then by the umbrella property it follows that C contains only three blocks which are merged into a single block in G'. In this case G' is a proper interval graph and updates to the internal data structure are trivial. The remaining case is analyzed in the following lemma.

LEMMA 4.2. *Let $B_1 < \cdots < B_k$ be a contig of C, such that $u \in B_i$ and $v \in B_j$ for some $1 \leq i < j \leq k$. Assume that $N(u) \neq N(v)$. Then G' is a proper interval graph if and only if $F_r(B_i) = B_{j-1}$ and $F_l(B_j) = B_{i+1}$ in G.*

Proof. Let G' be a proper interval graph. Since B_i and B_j are nonadjacent, $F_r(B_i) \leq B_{j-1}$ and $F_l(B_j) \geq B_{i+1}$. Suppose to the contrary that $F_r(B_i) < B_{j-1}$. Let $z \in B_{j-1}$. If in addition $F_l(B_j) = B_{i+1}$, then by the umbrella property $N[v] \supset N[z]$. (This is a strict containment.) As v and z are in distinct blocks, there exists a vertex $b \in N[v] \setminus N[z]$. But then v, b, z, and u induce a claw in G', a contradiction. Hence, $F_l(B_j) > B_{i+1}$ and therefore $F_r(B_{i+1}) < B_j$. Let $x \in B_{i+1}$ and let $y \in F_r(B_{i+1})$. As u and x are in distinct blocks, we have either $(u, y) \notin E(G)$ or there exists a vertex $a \in N[u] \setminus N[x]$ (or both). In the first case, v, u, x, y, and the vertices on a shortest path from y to v induce a chordless cycle in G'. In the second case u, a, x, and v induce a claw in G'. Hence, in both cases we arrive at a contradiction. By a symmetric argument we deduce that $F_l(B_j) = B_{i+1}$.

To prove the "if" part we provide a straight enumeration of $C \cup \{(u, v)\}$. If $B_i = \{u\}$, $F_r(B_{j-1}) = F_r(B_j)$, and $F_l(B_{j-1}) = B_i$ (i.e., $N[v] = N[B_{j-1}]$ in G'), we move v from B_j to B_{j-1}. Similarly, if B_j contained only v, $F_l(B_{i+1}) = F_l(B_i)$ and $F_r(B_{i+1}) = B_j$ (i.e., $N[u] = N[B_{i+1}]$ in G'), we move u from B_i to B_{i+1}. If u was not moved and B_i contained vertices other than u, we split B_i into $B_i = B_i \setminus \{u\}, \{u\}$ in this order. If v was not moved and B_j contained vertices other than v, we split B_j into $\{v\}, B_j = B_j \setminus \{v\}$ in this order. It is easy to see that the result is a straight enumeration of $C \cup \{(u, v)\}$. □

If u and v are neither end-vertices of distinct contigs nor end-vertices of a three-block contig, then, assuming that G' is a proper interval graph, the condition of

Lemma 4.2 must hold. We can check that in time $O(1)$, and if it is the case, change our data structure so as to reflect the new straight enumeration of blocks given in the proof of Lemma 4.2. We describe below the changes to our data structure.

- Block enumeration: Given in the proof of Lemma 4.2.
- Near pointers: Let $B_{k+1} = \emptyset$. If u was moved into B_{i+1}, then no change is necessary with respect to u. If $B_i \supset \{u\}$, u forms a new block and we set $N_l(\{u\}) = \&B_i, N_r(B_i) = \&\{u\}, N_r(\{u\}) = \&B_{i+1}$, and $N_l(B_{i+1}) = \&\{u\}$. Analogous updates are made with respect to v.
- Far pointers: If u was moved into B_{i+1}, then no change is necessary with respect to u. If $B_i \supset \{u\}$, we exchange the right self-pointer of B_i with the right self-pointer of (the new block) $\{u\}$. Let B denote the block containing v in G'. We also set $F_l(\{u\}) = F_l(B_i)$ and $F_r(\{u\}) = \&B$. Analogous updates are made with respect to v.

The following theorem summarizes the results of sections 3 and 4.

THEOREM 4.3. *The incremental proper interval graph representation problem is solvable in $O(1)$ time per added edge.*

5. The fully dynamic algorithm. In this section we give a fully dynamic algorithm for recognizing and representing proper interval graphs. The algorithm performs an operation involving d edges in $O(d + \log n)$ time. It supports four types of operations: adding a vertex, adding an edge, deleting a vertex, and deleting an edge. It is based on the incremental algorithm. The main difficulty in extending the incremental algorithm to handle all types of operations is updating the end pointers of blocks when both insertions and deletions are allowed. To bypass this problem we (implicitly) keep the identity of each block as an end/inner-block but do not keep end pointers at all. Instead, we maintain the connected components of G and use this information in our algorithm. In the next section we provide a fully dynamic algorithm for maintaining the connected components of a proper interval graph. This algorithm handles a modification request involving d edges in $O(d + \log n)$ time and determines whether two blocks are in the same connected component in $O(\log n)$ time. We describe below how each operation is handled by the fully dynamic proper interval graph representation algorithm.

5.1. The addition of a vertex. This operation is handled in essentially the same way as done by the incremental algorithm. However, in order to check if the end-blocks of two distinct segments are in distinct components, we query our data structure of connected components (in $O(\log n)$ time), rather than checking if the end pointers of these blocks do not point to each other.

5.2. The addition of an edge. Again, handling this operation is similar to its handling by the incremental algorithm, with the exception that in order to check if the endpoints of an edge are in distinct components, we query our data structure of connected components (in $O(\log n)$ time).

5.3. The deletion of a vertex. We show next how to update the contigs of G after deleting a vertex v of degree d. Note, that in this case G' is an induced subgraph of G, and hence, also a proper interval graph.

Denote by X the block containing v. If X contains vertices other than v then the data structure is simply updated by deleting v. Hence, we concentrate on the case that $X = \{v\}$. In time $O(d)$ we can find the segment of blocks which includes X and all its neighbors. Let the contig containing X be $B_1 < \cdots < B_k$, and let the blocks

of the segment be $B_i < \cdots < B_j$, where $X = B_l$ for some $1 \le i \le l \le j \le k$. The following updates should be performed:

- Block enumeration: If $1 < i < l$, we check whether B_i can be merged with B_{i-1}. If $F_l(B_i) = F_l(B_{i-1})$, $F_r(B_i) = B_l$, and $F_r(B_{i-1}) = B_{l-1}$, we merge these blocks by moving all vertices from B_i to B_{i-1} (in $O(d)$ time) and deleting B_i. If $l < j < k$ we deal similarly with B_j and B_{j+1}.
  Finally, we delete B_l. If $1 < l < k$ and B_{l-1}, B_{l+1} are nonadjacent, then by the umbrella property they are no longer in the same connected component, and the contig should be split into two contigs, one ending at B_{l-1} and the other beginning at B_{l+1}.
- Near pointers: Let $B_0 = \emptyset, B_{k+1} = \emptyset$. If B_i and B_{i-1} were merged, we update $N_r(B_{i-1}) = \&B_{i+1}$ and $N_l(B_{i+1}) = \&B_{i-1}$. Similar updates are made with respect to B_{j-1} and B_{j+1} in case B_j and B_{j+1} were merged. If the contig is split, we nullify $N_r(B_{l-1})$ and $N_l(B_{l+1})$. Otherwise, we update $N_r(B_{l-1}) = \&B_{l+1}$ and $N_l(B_{l+1}) = \&B_{l-1}$.
- Far pointers: If B_i and B_{i-1} were merged, we exchange the right self-pointer of B_i with the right self-pointer of B_{i-1}. Similar changes should be made with respect to B_j and B_{j+1}. We also set all right far pointers, previously pointing to B_l, to $\&B_{l-1}$ and all left far pointers, previously pointing to B_l, to $\&B_{l+1}$ (in $O(d)$ time).

Note that these updates take $O(d)$ time and require no knowledge about the connected components of G.

5.4. The deletion of an edge. Let (u, v) be an edge of G to be deleted. Let C be the connected component of G containing u and v. Let B_i and B_j be the blocks containing u and v, respectively, in a contig $B_1 < \cdots < B_k$ of C. If $i = j = k = 1$, then B_1 is split into $\{u\}, B_1 \setminus \{u, v\}$, and $\{v\}$, in this order, resulting in a straight enumeration of G'. Updates are trivial in this case. Henceforth we assume that $k > 1$. We first observe that $i \ne j$, i.e., $N[u] \ne N[v]$.

LEMMA 5.1. *If $N[u] = N[v]$ then G' is a proper interval graph if and only if C is a clique.*

Proof. To prove the "only if" part, we first show that every vertex $x \in C \setminus \{u, v\}$ is adjacent to both u and v. Suppose to the contrary that there exists a vertex $x \in C \setminus \{u, v\}$ which is not adjacent to u. Let $x = x_1, \ldots, x_k = u$ be a path in C from x to u. Let x_i be the first vertex on the path which is adjacent to u (and therefore also to v). Then $\{x_i, x_{i-1}, u, v\}$ induce a claw in G', a contradiction. Finally, if a and b are two nonadjacent vertices in $C \setminus \{u, v\}$, then $\{a, u, b, v\}$ induce a chordless cycle in G', a contradiction.

To prove the "if" part, notice that since C is a clique, it is a block in G, and therefore, $\{u\}, C \setminus \{u, v\}, \{v\}$ is a straight enumeration of $C \setminus \{(u, v)\}$. □

Since by our assumptions $k > 1$, we conclude that $N[u] \ne N[v]$, and therefore, $N(u) \ne N(v)$. Without loss of generality, $i < j$. The updates to the straight enumeration of $C \setminus \{(u, v)\}$ are derived from the following lemma.

LEMMA 5.2. *Let $B_1 < \cdots < B_k$ be a contig of C, such that $u \in B_i$ and $v \in B_j$ for some $1 \le i < j \le k$. Then G' is a proper interval graph if and only if $F_r(B_i) = B_j$ and $F_l(B_j) = B_i$ in G.*

Proof. Suppose that G' is a proper interval graph. We prove that $F_r(B_i) = B_j$. A symmetric argument shows that $F_l(B_j) = B_i$. Since B_i and B_j are adjacent in G, $F_r(B_i) \ge B_j$. Suppose to the contrary that $F_r(B_i) > B_j$. Let $x \in F_r(B_i)$. By the umbrella property $(x, v) \in E(G)$. Since x and v are in distinct blocks in G, either

there exists a vertex $a \in N[v] \setminus N[x]$ or there exists a vertex $b \in N[x] \setminus N[v]$ (or both). In the first case, by the umbrella property $(a, u) \in E(G)$. Therefore, u, x, v, and a induce a chordless cycle in G'. In the second case, x, b, u, and v induce a claw in G'. Hence in both cases we arrive at a contradiction.

To prove the converse implication we give a straight enumeration of $C \setminus \{(u, v)\}$. If $B_i = \{u\}$, $B_j = \{v\}$, and $j = i+1$, we have to split the contig into two contigs, one ending at B_i and the other beginning at B_j. If $B_j = \{v\}$, $F_l(B_{i-1}) = F_l(B_i)$, and $F_r(B_{i-1}) = B_{j-1}$ (i.e., $N[u] = N[B_{i-1}]$ in G'), we move u into B_{i-1}. If B_i contained only u, $F_r(B_{j+1}) = F_r(B_j)$ and $F_l(B_{j+1}) = B_{i+1}$ (i.e., $N[v] = N[B_{j+1}]$ in G'), we move v into B_{j+1}. If u was not moved and B_i contains vertices other than u, then B_i is split into $\{u\}, B_i = B_i \setminus \{u\}$ in this order. If v was not moved and B_j contains vertices other than v, then B_j is split into $B_j = B_j \setminus \{v\}, \{v\}$ in this order. The result is a straight enumeration of $C \setminus \{(u, v)\}$. □

If the conditions of Lemma 5.2 are fulfilled, then the following updates should be made:

- Block enumeration: Given in the proof of Lemma 5.2.
- Near pointers: Let $B_0 = \emptyset, B_{k+1} = \emptyset$. If $B_i = \{u\}$, $B_j = \{v\}$, and $j = i + 1$, we nullify $N_r(u)$. If B_i was split, we set $N_r(\{u\}) = \&B_i$, $N_l(B_i) = \&\{u\}$, $N_l(\{u\}) = \&B_{i-1}$ and $N_r(B_{i-1}) = \&\{u\}$. If B_i contained only u, and u was moved into B_{i-1}, we update $N_r(B_{i-1}) = \&B_{i+1}$ and $N_l(B_{i+1}) = \&B_{i-1}$. Analogous updates are made with respect to v.
- Far pointers: If $B_i = \{u\}$, $B_j = \{v\}$, and $j = i + 1$, we nullify $F_r(u)$. If B_i was split, we exchange the left self-pointer of B_i with the left self-pointer of $\{u\}$. We also set $F_l(\{u\}) = F_l(B_i)$ and $F_r(\{u\}) = \&B_y$, where $y = j$ in case v is no longer in B_j (that is, v was moved into B_{j+1} or B_j was split), and otherwise, $y = j - 1$. If B_i contained only u, and u was moved into B_{i-1}, we exchange the right self-pointer of B_i with the right self-pointers of B_{i-1}, and delete B_i. Analogous updates are made with respect to v.

Note that these updates take $O(1)$ time and require no knowledge about the connected components of G. Hence, from sections 5.3 and 5.4 there follows an optimal algorithm for the decremental proper interval graph representation problem. The following theorem summarizes this result.

THEOREM 5.3. *The decremental proper interval graph representation problem is solvable in $O(1)$ time per removed edge.*

6. Maintaining the connected components. In this section we describe a fully dynamic algorithm for maintaining connectivity in a proper interval graph G in $O(d + \log n)$ time per operation involving d edges. In section 7 we shall establish a lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per edge operation (in the cell probe model of computation with word-size b) for this problem.

The algorithm receives as input a series of operations to be performed on a graph, which can be any of the following: Adding a vertex, adding an edge, deleting a vertex, deleting an edge, or querying if two vertices are in the same connected component. It operates on the blocks of the graph rather than on its vertices. The algorithm depends on a data structure which includes the blocks and the contigs of the graph. It hence interacts with the proper interval graph representation algorithm. In response to an update request, changes are made to the representation of the graph based on the structure of its connected components prior to the update. Only then are the connected components of the graph updated. We provide a data structure of connected components which performs each operation in $O(\log n)$ time.

Let us denote by $B(G)$ the *block graph* of G, that is, a graph in which each vertex corresponds to a block of G and two vertices are adjacent if and only if their corresponding blocks are adjacent in G. The algorithm maintains a spanning forest F of $B(G)$. When a modification in the graph occurs, the spanning forest is updated accordingly. In order to decide if two blocks are in the same connected component, the algorithm checks if they belong to the same tree in F.

The key idea is to design F so that it can be efficiently updated upon a modification in G. We define the edges of F as follows: For every two vertices u and v in $B(G)$, $(u, v) \in E(F)$ if and only if their corresponding blocks are consecutive in a contig of G (or equivalently, if the near pointers of these blocks point to each other in our representation). Consequently, each tree in F is a path representing a contig. The crucial observation about F is that an addition or a deletion of a vertex or an edge in G induces $O(1)$ modifications to the vertices and edges of F. This can be seen by noting that each modification of G induces $O(1)$ updates to near pointers in our representation of G.

It remains to show a data structure for storing F that allows us to query for each vertex to which path it belongs, and that enables splitting a path upon a deletion of an edge in F, and linking two paths upon an addition of an edge to F. If we store the vertices of each path of F in a balanced binary tree, then each of these operations can be supported in $O(\log n)$ time (cf. [2]).

We are now ready to state our main result.

THEOREM 6.1. *The fully dynamic proper interval graph representation problem is solvable in $O(d + \log n)$ time per modification involving d edges.*

We note, that the performance of our representation algorithm depends on the performance of a data structure of connected components of a graph, which is a union of disjoint paths, that supports the following operations: linking two paths, splitting a path, and querying if two vertices belong to the same path. Given such a data structure which supports each operation in $O(f(n))$ time, for some function f, our representation algorithm can be implemented to run in $O(d + f(n))$ time per modification involving d edges.

7. The lower bounds. In this section we prove a lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per edge operation for fully dynamic proper interval graph recognition in the cell probe model of computation with word-size b (see [20] for details about the model). Furthermore, we prove the same lower bound also for the problem of fully dynamic connectivity maintenance of a proper interval graph.

Fredman and Henzinger [9] have shown a lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per operation (in the cell probe model of computation with word-size b) for fully dynamic connectivity, by reduction from the *helpful parity prefix sum* (HPPS) problem, which is defined below. We use similar constructions in our lower bound proofs.

The HPPS problem is a modified *parity prefix sum* problem (see [6] for definition of the latter problem). Its lower bound of $\Omega(\log n/(\log\log n + \log b))$ amortized time per operation follows from the work of Fredman and Saks [6]. It is defined as follows: Given an array $A[0], \ldots, A[n+1]$ of zeros and ones such that initially all $A[i]$ are 0, except $A[0]$ and $A[n+1]$ which are 1, execute an arbitrary sequence of the following operations:

Add(t, i, j): If $0 \le i < t < j \le n+1, A[i] > 0, A[j] > 0$, and $A[k] = 0$ for all $i < k < j$, then $A[t] = A[t] + 1$. Otherwise, do nothing.

Sum(t): Return $(\sum_{i=1}^{t} A[i]) \bmod 2$.

THEOREM 7.1. *Fully dynamic proper interval graph recognition takes amortized time* $\Omega(\log n/(\log\log n + \log b))$ *per edge operation in the cell probe model of computation with word-size* b.

Proof. Given an instance of the HPPS problem (i.e., a sequence of Add and Sum operations) we construct an instance of the dynamic proper interval graph recognition problem, such that each Add operation corresponds to $O(1)$ edge modifications in the dynamic proper interval graph instance, and each Sum operation corresponds to $O(1)$ temporary modifications in the dynamic graph: Depending on whether the modifications generate a proper interval graph we answer the Sum query and then reverse the modification. Thus, the lower bound for the HPPS problem shows that there exists a sequence of m operations for the dynamic proper interval recognition problem that takes $\Omega(m \log n/(\log\log n + \log b))$ time in the cell probe model of computation with word-size b.

Let $S_{-1} = 0$ and let $S_0 = 1$. Given an instance of the HPPS problem, define $S_t \equiv (\sum_{i=1}^{t} A[i]) \bmod 2$ for $1 \le t \le n$. The reduction is as follows: We construct a graph $G = (V, E)$ with $n + 2$ vertices labeled $-1, 0, 1, \ldots, n$, where each vertex v represents S_v. If $S_t = i$ for $i = 0, 1$, and $t' < t$ is the largest index such that $S_{t'} = i$, then G contains the edge (t', t). In other words, vertices t for which $S_t = 1$ are connected in a chain, which we henceforth call the *odd chain*, and all other vertices are connected in a chain, which we henceforth call the *even chain*. Note that the vertex labeled -1 lies on the even chain, and the vertex labeled 0 lies on the odd chain.

To answer a Sum(t) query ($1 \le t \le n$) we do the following:

1. If $(0, t) \in E$ or $(0, t'), (t', t) \in E$ for some vertex $t' \in V$, we output 1.
2. Otherwise, let t' be a vertex such that $t' > t$ and $(t, t') \in E$. If such a vertex exists, define $H \equiv G \setminus \{(t, t')\} \cup \{(0, t)\}$. Otherwise, let $H \equiv G \cup \{(0, t)\}$. If t is on the odd chain then this modification forms a chordless cycle. If t is on the even chain then the new graph is a single path or a union of two disjoint paths. Hence, H is a proper interval graph if and only if Sum(t) = 0. Thus, if H is a proper interval graph we output 0, and otherwise, we output 1. Note, that G is not modified in this case.

To perform an Add(t, i, j) operation we do the following:

1. Let i_{odd} (i_{even}) be the largest vertex on the odd (even) chain with $i_{odd} < t$ ($i_{even} < t$). Let j_{odd} (j_{even}) be the smallest vertex on the odd (even) chain with $j_{odd} \ge t$ ($j_{even} \ge t$), if such a vertex exists.
2. Delete from G the edges (i_{odd}, j_{odd}) and (i_{even}, j_{even}).
3. Add to G the edges (i_{odd}, j_{even}) and (i_{even}, j_{odd}).

By [9, Lemma 3.1] $i_{odd}, j_{odd}, i_{even}$, and j_{even} can be found by querying Sum(t) as follows: If Sum(t) = 1, then $i_{odd} = t - 1$, $j_{odd} = t$, $i_{even} = i - 1$ if $i > 1$, or $i_{even} = -1$ otherwise; and $j_{even} = j$ if $j <= n$, or j_{even} is undefined otherwise. If Sum(t) = 0, then $i_{odd} = i - 1$ if $i > 0$, or $i_{odd} = 0$ otherwise; $j_{odd} = j$ if $j \le n$, or j_{odd} is undefined otherwise; $i_{even} = t - 1$ if $t > 1$, or $i_{even} = -1$ otherwise; and $j_{even} = t$. This completes the reduction. □

Note, that since the key to the reduction above is the ability to detect cycles, similar arguments can be used to show that the same lower bound applies also to other problems, e.g., fully dynamic interval graph recognition and fully dynamic chordal graph recognition.

THEOREM 7.2. *There is a lower bound of* $\Omega(\log n/(\log\log n + \log b))$ *amortized time per edge operation in the cell probe model of computation with word-size* b *for*

fully dynamic connectivity maintenance of a proper interval graph.

Proof. We use the same reduction as in the proof of Theorem 7.1, with the exception that in order to answer a $Sum(t)$ query we check whether vertices 0 and t are connected. If the answer is positive we output 1, and otherwise we output 0. The reduction is valid, since the graph G, which is constructed in the reduction, is a union of two disjoint paths and therefore is a proper interval graph. □

Note added in proof. After the submission of the manuscript, we found out that a simpler reduction was given from the parity prefix sum problem to fully dynamic connectivity by Miltersen et al. [15]. This reduction allows the repeated modification of $A[t]$ for the same argument t and leads to alternative proofs of Theorems 7.1 and 7.2.

REFERENCES

[1] A. V. Carrano, *Establishing the order of human chromosome-specific DNA fragments*, in Biotechnology and the Human Genome, A. D. Woodhead and B. J. Barnhart, eds., Plenum Press, New York, 1988, pp. 37–50.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[3] D. Corneil, H. Kim, S. Natarajan, S. Olariu, and A. P. Sprague, *Simple linear time recognition of unit interval graphs*, Inform. Process. Lett., 55 (1995), pp. 99–104.

[4] X. Deng, P. Hell, and J. Huang, *Recognition and representation of proper circular arc graphs*, in Proceedings of the 2nd Integer Programming and Combinatorial Optimization (IPCO), Carnegie Mellon University, Pittsburgh, PA, 1992, pp. 114–121. Journal version: *Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs,* SIAM J. Comput., 25 (1996), pp. 390–403.

[5] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—A technique for speeding up dynamic graph algorithms*, in Proceedings of the 33rd Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1992, pp. 60–69.

[6] M. Fredman and M. Saks, *The cell probe complexity of dynamic data structures*, in Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 345–354.

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

[8] P. Hell, R. Shamir, and R. Sharan, *A fully dynamic algorithm for recognizing and representing proper interval graphs*, in Proceedings of the Seventh Annual European Symposium on Algorithms (ESA '99), Lecture Notes in Comput. Sci. 1643, Springer-Verlag, New York, 1999, pp. 527–539.

[9] M. Henzinger and M. Fredman, *Lower bounds for fully dynamic connectivity problems in graphs*, Algorithmica, 22 (1998), pp. 351–362.

[10] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree,* 2*-edge and biconnectivity*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98), New York, 1998, ACM Press, New York, pp. 79–89.

[11] W.-L. Hsu, *A simple test for interval graphs*, in Proceedings of the 18th International Workshop (WG '92), Wiesbaden-Naurod, Germany, Graph-Theoretic Concepts in Computer Science, W.-L. Hsu and R. C. T. Lee, eds., Springer-Verlag, Berlin, 1992, pp. 11–16.

[12] L. Ibarra, *Fully dynamic algorithms for chordal graphs*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), SIAM, Philadelphia, 1999, pp. 923–924.

[13] C. G. Lekkerkerker and J. C. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.

[14] P. Looges and S. Olariu, *Optimal greedy algorithms for indifference graphs*, Comput. Math. Appl., 25 (1993), pp. 15–25.

[15] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia, *Complexity models for incremental computation,* Theoret. Comput. Sci., 130 (1994), pp. 203–236.

[16] F. S. Roberts, *Indifference graphs*, in Proof Techniques in Graph Theory, F. Harary, ed., Academic Press, New York, 1969, pp. 139–146.

[17] M. Thorup, *Near-optimal fully-dynamic graph connectivity*, in Proceedings of the 32th Annual ACM Symposium on Theory of Computing (STOC'00), 2000, pp. 343–350.
[18] J. Watson, M. Gilman, J. Witkowski, and M. Zoller, *Recombinant DNA*, 2nd ed., W. H. Freeman, New York, 1992.
[19] G. Wegner, *Eigenschaften der nerven homologische einfacher familien in* R^n, Ph.D. thesis, University of Göttingen, Göttingen, Germany, 1967.
[20] A. Yao, *Should tables be sorted?*, J. ACM, 28 (1981), pp. 615–628.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 306–322

PROVABLY FAST AND ACCURATE RECOVERY OF EVOLUTIONARY TREES THROUGH HARMONIC GREEDY TRIPLETS*

MIKLÓS CSŰRÖS† AND MING-YANG KAO†

Abstract. We give a greedy learning algorithm for reconstructing an evolutionary tree based on a certain harmonic average on triplets of terminal taxa. After the pairwise distances between terminal taxa are estimated from sequence data, the algorithm runs in $\mathcal{O}(n^2)$ time using $\mathcal{O}(n)$ work space, where n is the number of terminal taxa. These time and space complexities are optimal in the sense that the size of an input distance matrix is n^2 and the size of an output tree is n. Moreover, in the Jukes–Cantor model of evolution, the algorithm recovers the correct tree topology with high probability using sample sequences of length polynomial in (1) n, (2) the logarithm of the error probability, and (3) the inverses of two small parameters.

Key words. evolutionary trees, the Jukes–Cantor model of evolution, computational learning, harmonic greedy triplets

AMS subject classifications. 05C05, 05C85, 92D15, 60J85, 92D20

PII. S009753970037905X

1. Introduction.

Algorithms for reconstructing evolutionary trees are useful tools in biology [16, 22]. These algorithms usually compare aligned character sequences for the terminal taxa in question to infer their evolutionary relationships. In the past, such characters were often categorical variables of morphological features; newer studies have taken advantage of available biomolecular sequences. This paper focuses on datasets of the latter type.

We present a new learning algorithm, called *fast harmonic greedy triplets* (Fast-HGT), using a greedy strategy based on a certain harmonic average on triplets of terminal taxa. After the pairwise distances between terminal taxa are estimated from their observed sequences, Fast-HGT runs in $\mathcal{O}(n^2)$ time using $\mathcal{O}(n)$ work space, where n is the number of terminal taxa. These time and space complexities are optimal in the sense that n^2 is the size of an input distance matrix and n is the size of an output tree. An earlier variant of Fast-HGT takes $\mathcal{O}(n^5)$ time [5]. In the Jukes–Cantor model of sequence evolution generalized for an arbitrary alphabet [22], Fast-HGT is proven to recover the correct topology with high probability while requiring sample sequences of length ℓ polynomial in (1) n, (2) the logarithm of the error probability, and (3) the inverses of two small parameters (Theorem 3.8). In subsequent work [6], Fast-HGT and its variants are shown to have similar theoretical performance in more general Markov models of evolution.

Among the related work, there are four other algorithms which have essentially the same guarantee on the length ℓ of sample sequences. These are the dyadic closure method (DCM) [10] and the witness-antiwitness method (WAM) [11] of Erdős et al., the algorithm of Cryan, Goldberg, and Goldberg (CGG) [4], and the DCM-Buneman algorithm of Huson, Nettles, and Warnow [18]. Not all of these results

*Received by the editors March 13, 2000; accepted for publication (in revised form) November 22, 2000; published electronically July 31, 2001. This research was supported in part by NSF grant 9531028.

http://www.siam.org/journals/sicomp/31-1/37905.html

†Department of Computer Science, Yale University, New Haven, CT 06520 (csuros-miklos@cs.yale.edu, kao-ming-yang@cs.yale.edu).

analyzed the space complexity. In terms of time complexity, DCM-Buneman is not a polynomial-time algorithm. CGG runs in polynomial time, whose degree has not been explicitly determined but which appears to be higher than n^2. DCM takes $\mathcal{O}(n^5 \log n)$ time to assemble $\mathcal{O}(n^4)$ quartets using $\mathcal{O}(n^4)$ space. The two versions of WAM take $\mathcal{O}(n^6 \log n)$ and $\mathcal{O}(n^4 \log n \log \ell)$ time, respectively. In the uniform and Yule–Harding models of randomly generating trees, with high probability, these two latter running times are reduced to $\mathcal{O}(n^3 \operatorname{polylog} n)$ and $\mathcal{O}(n^2 \operatorname{polylog} n)$, respectively. Under these two tree distributions, Erdős et al. [10] further showed that with high probability, the required sample size of DCM is polylogarithmic in n; this bound also applies to WAM, CGG, DCM-Buneman, and Fast-HGT.

Among the algorithms with no known comparbale guarantees on ℓ, the neighbor joining method of Saitou and Nei [22] runs in $\mathcal{O}(n^3)$ time and reconstructs many trees highly accurately in practice, although the best known upper bound on its required sample size is exponential in n [3]. Maximum likelihood methods [15, 14] are not known to achieve the optimal required sample size as such methods are usually expected to [20]; moreover, all their known implementations take exponential time to find local optima, and none can find provably global optima. Parsimony methods aim to compute a tree that minimizes the number of mutations leading to the observed sequences [13]; in general, such optimization is NP-hard [8]. Some algorithms strive to find an evolutionary tree among all possible trees to fit the observed distances the best according to some metric [1]; such optimization is NP-hard for L_1 and L_2 norms [7] and for L_∞ [1].

A common goal of the above algorithms is to construct a tree with the same topology as that of the true tree. In contrast, the work on PAC-learning the true tree in the j-state general Markov model [21] aims to construct a tree which is close to the true tree in terms of the leaf distribution in the sense of Kearns et al. [19] but which need not be the same as the true tree. Farach and Kannan [12] gave an $\mathcal{O}(n^2\ell)$-time algorithm (FK) for the symmetric case of the 2-state model provided that all pairs of leaves have a sufficiently high probability of being the same. Ambainis et al. [2] gave a nearly tight lower bound on ℓ for achieving a given variational distance between the true tree and the reconstructed tree. As for obtaining the true tree, the best known upper bound on ℓ required by FK is exponential in n. CGG [4] also improves upon FK to PAC-learn in the general 2-state model without the symmetry and leaf similarity constraints.

The remainder of the paper is organized as follows. Section 2 reviews the generalized Jukes–Cantor model of sequence evolution and discusses distance-based probabilistic techniques. Section 3 gives Fast-HGT. Section 4 concludes the paper with some directions for further research.

2. Model and techniques.

Section 2.1 defines the model of evolution used in the paper. Section 2.2 defines our problem of recovering evolutionary trees from biological sequences. Sections 2.3 through 2.5 develop basic techniques for the problem.

2.1. A model of sequence evolution.

This paper employs the generalized Jukes–Cantor model [22] of sequence evolution defined as follows. Let $m \geq 2$ and $n \geq 3$ be two integers. Let $\mathcal{A} = \{a_1, \ldots, a_m\}$ be a finite alphabet. An *evolutionary tree* T for $\mathcal{A}$ is a rooted binary tree of n leaves with an *edge mutation probability* p_e for each tree edge e. The edge mutation probabilities are bounded away from 0 and $1 - \frac{1}{m}$, i.e., there exist f and g such that for every edge e of T, $0 < f \leq p_e \leq g < 1 - \frac{1}{m}$. Given a sequence $s_1 \cdots s_\ell \in \mathcal{A}^\ell$ associated with the root of T, a set of n *mutated sequences* in $\mathcal{A}^\ell$ is generated by ℓ random labelings of the tree at the nodes. These ℓ labelings are

mutually independent. The labelings at the jth leaf give the jth *mutated sequence* $s_1^{(j)} \cdots s_\ell^{(j)}$, where the ith labeling of the tree gives the ith symbols $s_i^{(1)}, \ldots, s_i^{(n)}$. The ith labeling is carried out from the root towards the leaves along the edges. The root is labeled by s_i. On edge e, the child's label is the same as the parent's with probability $1 - p_e$ or is different with probability $\frac{p_e}{m-1}$ for each different symbol. Such *mutations* of symbols along the edges are mutually independent.

2.2. Problem formulation. The *topology* $\Psi(T)$ of T is the unrooted tree obtained from T by omitting the edge mutation probability and by replacing the two edges e_1 and e_2 between the root and its children with a single edge e_0. Note that the leaves of $\Psi(T)$ are labeled with the same sequences as in T, but $\Psi(T)$ need not be labeled otherwise. The *weighted topology* $\Psi_{\mathrm{w}}(T)$ of T is $\Psi(T)$ where each edge $e \neq e_0$ of $\Psi(T)$ is further weighted by its edge mutation probability p_e in T and for technical reasons, the edge e_0 is weighted by $1 - (1 - p_{e_1})(1 - p_{e_2})$.

For technical convenience, the weight of each edge XY in $\Psi_{\mathrm{w}}(T)$ is often replaced by a certain edge length, such as Δ_{XY} in (2.5), from which the weight of XY can be efficiently determined.

The *weighted evolutionary topology* problem is that of taking n mutated sequences as input and recovering $\Psi_{\mathrm{w}}(T)$ with high accuracy and high probability. Fast-HGT is a learning algorithm for this problem.

Remark. The special treatment for e_1 and e_2 is due to the fact that the root sequence may be entirely arbitrary and thus, in general, no algorithm can place the root accurately. This is consistent with the fact that the root sequence is not directly observable in practice, and locating the root requires considerations beyond those of general modeling [22]. If the root sequence is also given as input, Fast-HGT can be modified to locate the root and the weights of e_1 and e_2 in a straightforward manner.

2.3. Probabilistic closeness. Fast-HGT is based on a notion of probabilistic closeness between nodes. For the ith random labeling of T, we identify each node of T with the random variable X_i that gives the labeling at the node. Note that since $s_1 \cdots s_\ell$ may be arbitrary, the random variables X_i for different i are not necessarily identically distributed. For brevity, we often omit the index i of X_i in a statement if the statement is independent of i.

For nodes X and $Y \in T$, let $p_{XY} = \Pr\{\, X \neq Y \,\}$. The *closeness* of X and Y is

$$\sigma_{XY} = \Pr\{\, X = Y \,\} - \frac{1}{m-1}\Pr\{\, X \neq Y \,\} = 1 - \alpha p_{XY}, \text{where } \alpha = \frac{m}{m-1}. \tag{2.1}$$

LEMMA 2.1 (folklore). *If node Y is on the path between two nodes X and Z in T, then $\sigma_{XZ} = \sigma_{XY}\sigma_{YZ}$.*

If X and Y are leaves, their closeness is estimated from sample sequences as

$$\hat{\sigma}_{XY} = \frac{1}{\ell}\sum_{i=1}^{\ell} I_{\hat{X}_i \hat{Y}_i}, \tag{2.2}$$

where $\hat{X}_1, \ldots, \hat{X}_\ell$ and $\hat{Y}_1, \ldots, \hat{Y}_\ell$ are the symbols at positions $1, \ldots, \ell$ of the observed sample sequences for the two leaves, and

$$I_{xy} = \begin{cases} \frac{-1}{m-1} & \text{if } x \neq y; \\ 1 & \text{if } x = y. \end{cases}$$

The next lemma is useful for analyzing the estimation given by (2.2).

LEMMA 2.2. *For $\epsilon > 0$,*

$$\Pr\left\{ \frac{\hat{\sigma}_{XY}}{\sigma_{XY}} \le 1-\epsilon \right\} \le \exp\left(-\frac{2}{\alpha^2}\ell\sigma_{XY}^2\epsilon^2\right); \tag{2.3}$$

$$\Pr\left\{ \frac{\hat{\sigma}_{XY}}{\sigma_{XY}} \ge 1+\epsilon \right\} \le \exp\left(-\frac{2}{\alpha^2}\ell\sigma_{XY}^2\epsilon^2\right). \tag{2.4}$$

Proof. By (2.2),

$$\Pr\left\{ \frac{\hat{\sigma}_{XY}}{\sigma_{XY}} \le 1-\epsilon \right\} = \Pr\left\{ \sum_{i=1}^{\ell} (I_{X_iY_i} - \sigma_{XY}) \le -\ell\sigma_{XY}\epsilon \right\};$$

$$\Pr\left\{ \frac{\hat{\sigma}_{XY}}{\sigma_{XY}} \ge 1+\epsilon \right\} = \Pr\left\{ \sum_{i=1}^{\ell} (I_{X_iY_i} - \sigma_{XY}) \ge \ell\sigma_{XY}\epsilon \right\}.$$

Since $\frac{-1}{m-1} \le I_{X_iY_i} \le 1$ and $\mathbb{E}[\, I_{X_iY_i} - \sigma_{XY}\,] = 0$, we use Hoeffding's inequality [17] on sums of independent bounded random variables to have (2.3) and (2.4). ∎

2.4. Distance and harmonic mean. The *distance* of nodes X and $Y \in T$ is

$$\Delta_{XY} = -\ln \sigma_{XY}. \tag{2.5}$$

For an edge XY in T, Δ_{XY} is called the *edge length* of XY.

Fast-HGT uses Statement 2.3 of the next corollary to locate internal nodes of T.

COROLLARY 2.3. *Let X, Y, and Z be nodes in T.*

1. *If $X \neq Y$, then $\Delta_{XY} = \Delta_{YX} > 0$. Also, $\Delta_{XX} = 0$.*
2. *If Y is on the path between X and Z in T, then $\Delta_{XZ} = \Delta_{XY} + \Delta_{YZ}$.*
3. *For any σ with $\sigma_{XY} \le \sigma < 1$, there is a node P on the path between X and Y in T such that $\sigma(1-\alpha g)^{1/2} \le \sigma_{XP} \le \sigma(1-\alpha g)^{-1/2}$. Furthermore, if $\sigma_{XY}(1-\alpha g)^{1/2} < \sigma < (1-\alpha g)^{-1/2}$, then P is distinct from X and Y.*

Proof. Statements 1 and 2 follow from (2.1) and Lemma 2.1. Statement 3 becomes straightforward when restated in terms of distance as follows. For any Δ with $\Delta_{XY} \ge \Delta > 0$, there is a node P on the path between X and Y in T such that $\Delta + \frac{-\ln(1-\alpha g)}{2} \ge \Delta_{XP} \ge \Delta - \frac{-\ln(1-\alpha g)}{2}$. Furthermore, if $\Delta_{XY} - \frac{-\ln(1-\alpha g)}{2} > \Delta > \frac{-\ln(1-\alpha g)}{2}$, then P is distinct from X and Y. ∎

If X and Y are leaves, their distance is estimated from sample sequences as

$$\hat{\Delta}_{XY} = \begin{cases} -\ln \hat{\sigma}_{XY} & \text{if } \hat{\sigma}_{XY} > 0; \\ \infty & \text{otherwise.} \end{cases} \tag{2.6}$$

A *triplet* XYZ consists of three distinct leaves X, Y, and Z of T. There is an internal node P in T at which the pairwise paths between the leaves in XYZ intersect; see Figure 2.1. P is the *center* of XYZ, and XYZ *defines* P. Note that a star is formed by the edges on the paths between P and the three leaves in XYZ.

By Corollary 2.3(2), the distance between P and a leaf in XYZ, say, X, can be obtained as $\Delta_{XP} = \frac{\Delta_{XY}+\Delta_{XZ}-\Delta_{YZ}}{2}$, which is estimated by

$$\hat{\Delta}_{XP} = \frac{\hat{\Delta}_{XY} + \hat{\Delta}_{XZ} - \hat{\Delta}_{YZ}}{2}. \tag{2.7}$$

The *closeness* of XYZ is $\sigma_{XYZ} = \frac{3}{\frac{1}{\sigma_{XY}}+\frac{1}{\sigma_{XZ}}+\frac{1}{\sigma_{YZ}}}$, which is estimated by $\hat{\sigma}_{XYZ} = \frac{3}{\frac{1}{\hat{\sigma}_{XY}}+\frac{1}{\hat{\sigma}_{XZ}}+\frac{1}{\hat{\sigma}_{YZ}}}$. XYZ is called *positive* if $\hat{\sigma}_{XY}$, $\hat{\sigma}_{XZ}$, and $\hat{\sigma}_{YZ}$ are all positive.

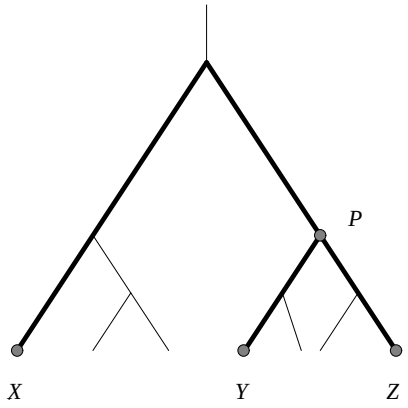


FIG. 2.1. *P is the center of triplet XYZ.*

The next corollary relates σ_{XYZ} and the pairwise closenesses of X, Y, and Z.

COROLLARY 2.4. *If $\sigma_{XP} \leq \sigma_{YP} \leq \sigma_{ZP}$, then $\sigma_{XY} \leq \sigma_{XZ} \leq \sigma_{YZ}$, $\sigma_{XZ} \geq \frac{2}{3}\sigma_{XYZ}$, and $\sigma_{YP}^2 \geq \frac{1}{3}\sigma_{XYZ}$.*

Proof. This corollary follows from Lemma 2.1 and simple algebra. □

The next lemma relates σ_{XYZ} to the probability of overestimating the distance between P and a leaf in XYZ using (2.7).

LEMMA 2.5. *For $0 < \epsilon < 1$,*

$$\Pr\left\{ \hat{\Delta}_{XP} - \Delta_{XP} \geq \frac{-\ln(1-\epsilon)}{2} \right\} \leq 3\exp\left(-\frac{2}{9\alpha^2}\ell\sigma_{XYZ}^2\epsilon^2\right).$$

Proof. See section A.1. □

2.5. Basis of a greedy strategy. Let d_{XY} denote the number of edges in the path between two leaves X and Y in T. By Lemma 2.1, σ_{XY} can be as small as $(1-\alpha g)^{d_{XY}}$. Thus, the larger d_{XY} is, the more difficult it is to estimate σ_{XY} and Δ_{XY}. This intuition leads to a natural greedy strategy outlined below that favors leaf pairs with small d_{XY} and large σ_{XY}.

The *g-depth* of a node in a rooted tree T' is the smallest number of edges in a path from the node to a leaf. Let e be an edge between nodes u_1 and u_2. Let T_1' and T_2' be the subtrees of T' obtained by cutting e which contain u_1 and u_2, respectively. The *g-depth* of e in T' is the larger of the g-depth of u_1 in T_1' and that of u_2 in T_2'. The *g-depth* of a rooted tree is the largest possible g-depth of an edge in the tree. (The prefix g emphasizes that this usage of depth is nonstandard in graph theory.)

Let d be the g-depth of T. Variants of the next lemma have proven very useful and insightful; see, e.g., [9, 11, 10].

LEMMA 2.6.

1. $d \leq 1 + \lfloor \log_2(n-1) \rfloor$.
2. *Every internal node P of T except the root has a defining triplet XYZ such that d_{XP}, d_{YP}, and d_{ZP} are all at most $d+1$ and thus, $\sigma_{XYZ} \geq (1-\alpha g)^{2(d+1)}$. Every leaf of T is in such a triplet.*

Proof. The proof is straightforward. Note that the more unbalanced T is, the smaller its g-depth is. □

In T, the star formed by a defining triplet of an internal node contains the three edges incident to the internal node. Thus, $\Psi(T)$ can be reconstructed from triplets

described in Lemma 2.6(2) or those with similarly large closenesses. This observation motivates the following definitions. Let

$$\sigma_{\mathrm{lg}} = \frac{3\sqrt{2}}{2}\left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^2 (1-\alpha g)^{2d+4};\quad \sigma_{\mathrm{sm}} = \frac{\sigma_{\mathrm{lg}}}{\sqrt{2}};\quad \sigma_{\mathrm{md}} = \frac{\sigma_{\mathrm{lg}}+\sigma_{\mathrm{sm}}}{2}.$$

Remark. The choice of σ_{lg} is obtained by solving (3.2), (3.3), and (3.4).

A triplet XYZ is *large* if $\sigma_{XYZ} \geq \sigma_{\mathrm{lg}}$; it is *small* if $\sigma_{XYZ} \leq \sigma_{\mathrm{sm}}$. Note that by Lemma 2.6(2), each nonroot internal node of T has at least one large defining triplet.

LEMMA 2.7. *The first inequality below holds for all large triplets XYZ, and the second holds for all small triplets.*

$$\Pr\{\,\hat{\sigma}_{XYZ} \leq \sigma_{\mathrm{md}}\,\} \leq \exp\left(-\frac{\left(\sqrt{2}-1\right)^2}{36\alpha^2}\ell\sigma_{\mathrm{lg}}^2\right); \tag{2.8}$$

$$\Pr\{\,\hat{\sigma}_{XYZ} \geq \sigma_{\mathrm{md}}\,\} \leq \exp\left(-\frac{\left(\sqrt{2}-1\right)^2}{36\alpha^2}\ell\sigma_{\mathrm{lg}}^2\right). \tag{2.9}$$

Proof. See section A.2. □

A nonroot internal node of T may have more than one large defining triplet. Consequently, since distance estimates contain errors, we may obtain an erroneous estimate of $\Psi(T)$ by reconstructing the same internal node more than once from its different large defining triplets. To address this issue, Fast-HGT adopts a threshold $0 < \Delta_{\min} < \frac{-\ln(1-\alpha f)}{2}$ based on the fact that the distance between two distinct nodes is at least $-\ln(1-\alpha f)$; also let $c = \frac{\Delta_{\min}}{-\ln(1-\alpha f)}$. Fast-HGT considers the center P of a triplet XYZ and the center Q of another triplet XUV to be separate if and only if

$$|\hat{\Delta}_{XP} - \hat{\Delta}_{XQ}| \geq \Delta_{\min}, \tag{2.10}$$

where $\hat{\Delta}_{XP} = (\hat{\Delta}_{XY} + \hat{\Delta}_{XZ} - \hat{\Delta}_{YZ})/2$ and $\hat{\Delta}_{XQ} = (\hat{\Delta}_{XU} + \hat{\Delta}_{XV} - \hat{\Delta}_{XV})/2$. Notice that two triplet centers can be compared in this manner only if the triplets share at least one leaf. The next lemma shows that a large triplet's center is estimated within a small error with high probability.

LEMMA 2.8. *Let P be the center of a triplet XYZ. If XYZ is not small, then*

$$\Pr\left\{\left|\hat{\Delta}_{XP} - \Delta_{XP}\right| \geq \frac{\Delta_{\min}}{2}\right\} \leq 7\exp\left(-\frac{c^2}{81}\ell\sigma_{\mathrm{lg}}^2 f^2\right). \tag{2.11}$$

Proof. See section A.3. □

We next define and analyze two key events $\mathcal{E}_{\mathrm{c}}$ and $\mathcal{E}_{\mathrm{g}}$ as follows. The subscripts c and g denote the words greedy and center, respectively.

- $\mathcal{E}_{\mathrm{c}}$ is the event that for every triplet XYZ that is not small, $\left|\hat{\Delta}_{XP} - \Delta_{XP}\right| < \frac{\Delta_{\min}}{2}$, $\left|\hat{\Delta}_{YP} - \Delta_{YP}\right| < \frac{\Delta_{\min}}{2}$, and $\left|\hat{\Delta}_{ZP} - \Delta_{ZP}\right| < \frac{\Delta_{\min}}{2}$, where P is the center of XYZ.
- $\mathcal{E}_{\mathrm{g}}$ is the event that $\hat{\sigma}_{XYZ} > \hat{\sigma}_{X'Y'Z'}$ for every large triplet XYZ and every small triplet $X'Y'Z'$.

LEMMA 2.9.

$$\Pr\{\,\overline{\mathcal{E}}_{\mathrm{c}}\,\} \leq 21\binom{n}{3}\exp\left(-\frac{c^2}{81}\ell\sigma_{\mathrm{lg}}^2 f^2\right);\ \Pr\{\,\overline{\mathcal{E}}_{\mathrm{g}}\,\} \leq \binom{n}{3}\exp\left(-\frac{\left(\sqrt{2}-1\right)^2}{36\alpha^2}\ell\sigma_{\mathrm{lg}}^2\right).$$

Proof. The inequalities follow from (2.11) and Lemma 2.7, respectively. ∎

3. Fast-HGT. Section 3.1 details Fast-HGT. Section 3.2 analyzes its running time and work space. Section 3.3 proves technical lemmas for bounding the algorithm's required sample size. Section 3.4 analyzes this sample size.

Algorithm Fast Harmonic Greedy Triplets
Input:
- $\Delta_{\min}$;
- $\hat{\Delta}_{XY}$ for all leaves X and Y of T which are computed via (2.2) and (2.6) from n mutated length-ℓ sequences generated by T.

Output: $\Psi_{\mathrm{w}}(T)$.

F1 Select an arbitrary leaf A and find a triplet ABC with the maximum $\hat{\sigma}_{ABC}$.
F2 **if** ABC is not positive **then** let T^* be the empty tree, **fail**, and **stop**.
F3 Let T^* be the star with three edges formed by ABC and its center D.
F4 Use (2.7) to set $\Delta^*_{AD} \leftarrow \hat{\Delta}_{AD}$, $\Delta^*_{BD} \leftarrow \hat{\Delta}_{BD}$, $\Delta^*_{CD} \leftarrow \hat{\Delta}_{CD}$.
F5 Set $\mathrm{def}(D) \leftarrow \{A, B, C\}$.
F6 First set all $\mathcal{S}[M]$ to null; then for $Q_1Q_2 \in \{AD, BD, CD\}$, Update-$\mathcal{S}(Q_1Q_2)$.
F7 **repeat**
F8 **if** $\mathcal{S}[M] =$ null for all leaves $M \in T$ **then fail** and **stop**.
F9 Find $\mathcal{S}[N] = \langle P_1P_2, NXY, P, \Delta^*_{P_1P}, \Delta^*_{P_2P}, \Delta^*_{NP}\rangle$ with the maximum $\hat{\sigma}_{NXY}$.
F10 Split P_1P_2 into two edges P_1P and P_2P in T^* with lengths $\Delta^*_{P_1P}$ and $\Delta^*_{P_2P}$.
F11 Add to T^* a leaf N and an edge NP with length Δ^*_{NP}.
F12 Set $\mathrm{def}(P) \leftarrow \{N, X, Y\}$.
F13 For every M with $\mathcal{S}[M]$ containing the edge P_1P_2, set $\mathcal{S}[M] \leftarrow$ null.
F14 For each $Q_1Q_2 \in \{P_1P, P_2P, NP\}$, Update-$\mathcal{S}(Q_1Q_2)$.
F15 **until** all leaves of T are inserted to T^*; i.e., this loop has iterated $n-3$ times.
F16 Output T^*.

FIG. 3.1. *The* Fast-HGT *algorithm.*

Algorithm Update-$\mathcal{S}$
Input: an edge $Q_1Q_2 \in T^*$
U1 Find all splitting tuples for $Q_1Q_2 \in T^*$.
U2 For each $\langle Q_1Q_2, MUV, Q, \Delta^*_{Q_1Q}, \Delta^*_{Q_2Q}, \Delta^*_{MQ}\rangle$ at line U1, assign it to $\mathcal{S}[M]$ if $\hat{\sigma}_{MUV}$ is greater than that of $\mathcal{S}[M]$.

FIG. 3.2. *The Update-*$\mathcal{S}$ *subroutine.*

3.1. The description of Fast-HGT. Fast-HGT and its subroutines Update-$\mathcal{S}$ and Split-Edge are detailed in Figures 3.1, 3.2, and 3.3, respectively.

Given $\Delta_{\min}$ and n mutated sequences as input, the task of Fast-HGT is to recover $\Psi_{\mathrm{w}}(T)$. The algorithm first constructs a star T^* formed by a large triplet at lines F1 through F3. It then inserts into T^* a leaf of T and a corresponding internal node per iteration of the repeat at line F7 until T^* has a leaf for each input sequence. The T^* at line F16 is our reconstruction of $\Psi_{\mathrm{w}}(T)$. For $k = 3, \ldots, n$, let T^*_k be the version of T^* with k leaves constructed during a run of Fast-HGT; i.e., T^*_3 is constructed at line F3, and T^*_k with $k \geq 4$ is constructed at line F11 during the $(k-3)$th iteration of the repeat. Note that T^*_n is output at line F16.

Algorithm Split-Edge
Input: an edge P_1P_2 in T^* and a relevant triplet NXY with center P.
Output: If P is strictly between P_1 and P_2 in T and thus can be inserted on P_1P_2, then we return the message "split" and the edge lengths $\Delta^*_{P_1P}$, $\Delta^*_{P_2P}$, and Δ^*_{NP}. Otherwise, we return a reason why P cannot be inserted.

S1 Use (2.7) to compute $\hat{\Delta}_{XP}$, $\hat{\Delta}_{YP}$, $\hat{\Delta}_{NP}$ for NXY.
S2 Let $X_1 \in \{X, Y\} \cap \mathrm{def}(P_1)$ and $X_2 \in \{X, Y\} \cap \mathrm{def}(P_2)$.
S3 For each $i = 1$ or 2, **if** P_i is an internal node of T^*
S4 **then** use (2.7) to compute $\hat{\Delta}_{X_iP_i}$ for the triplet formed by $\mathrm{def}(P_i)$
S5 **else** set $\hat{\Delta}_{X_iP_i} \leftarrow 0$.
S6 Set $\Delta_1 \leftarrow \hat{\Delta}_{X_1P} - \hat{\Delta}_{X_1P_1}$ and $\Delta_2 \leftarrow \hat{\Delta}_{X_2P} - \hat{\Delta}_{X_2P_2}$.
S7 **if** $|\Delta_1| < \Delta_{\min}$ or $|\Delta_2| < \Delta_{\min}$
S8 **then return** "too close"
S9 **else begin**
S10 **if** P_2 (respectively, P_1) is on the path between P_1 and X_1 (P_2 and X_2) in T^*
S11 **then** set $\Delta'_1 \leftarrow -\Delta_1$ ($\Delta'_2 \leftarrow -\Delta_2$)
S12 **else** set $\Delta'_1 \leftarrow \Delta_1$ ($\Delta'_2 \leftarrow \Delta_2$).
(*Remark.* Since X_1 may equal X_2, the tests for P_1 and P_2 are both needed.)
S13 Set $\Delta''_1 \leftarrow (\Delta'_1 + \Delta^*_{P_1P_2} - \Delta'_2)/2$ and $\Delta''_2 \leftarrow (\Delta'_2 + \Delta^*_{P_1P_2} - \Delta'_1)/2$.
(*Remark.* $\Delta''_1 + \Delta''_2 = \Delta^*_{P_1P_2}$, Δ''_1 estimates Δ_{P_1P}, and Δ''_2 estimates Δ_{P_2P}.)
S14 **if** $\Delta''_1 \geq \Delta^*_{P_1P_2}$ or $\Delta''_2 \geq \Delta^*_{P_1P_2}$
S15 **then return** "outside this edge"
S16 **else return** "split", Δ''_1, Δ''_2, $\hat{\Delta}_{NP}$.
S17 **end.**

FIG. 3.3. *The Split-Edge subroutine.*

A node Q is *strictly between* nodes Q_1 and Q_2 in T if Q is on the path between Q_1 and Q_2 in T but $Q \neq Q_1$, $Q \neq Q_2$, and Q is not the root of T. At each iteration of the repeat, Fast-HGT finds an edge P_1P_2 in T^* and a triplet NXY where $X, Y \in T^*$, $N \notin T^*$, and the center P of NXY is strictly between on P_1 and P_2 in T^*. Such P_1P_2 and NXY can be used to insert N and P into T^*. We record an insertion by letting $\mathrm{def}(P) = \{N, X, Y\}$; for notational uniformity, let $\mathrm{def}(X) = \{X\}$ for all leaves X.

At line F6, $\mathcal{S}$ is an array indexed by the leaves M of T^*. At the beginning of each iteration of the repeat, $\mathcal{S}[N]$ stores the most suitable P_1P_2 and NXY for inserting N into T^*. $\mathcal{S}$ is initialized at line F6; it is updated at lines F13 and F14 after a new leaf and a new internal node are inserted into T^*. The precise content of $\mathcal{S}$ is described in Lemma 3.6.

To further specify $\mathcal{S}[N]$, we call NXY *relevant* for $P_1P_2 \in T^*_k$ if it is positive, $N \notin T^*_k$, $X \in \mathrm{def}(P_1)$, $Y \in \mathrm{def}(P_2)$, and P_1P_2 is on the path between X and Y in T^*_k. We use Split-Edge to determine whether the center P of a relevant NXY is strictly between P_1 and P_2 in T. We also use Split-Edge to calculate an estimation $\Delta^*_{P'P''}$ of $\Delta_{P'P''}$ for each edge $P'P'' \in T^*_k$, which is called the *length* of $P'P''$ in T^*_k. Split-Edge has three possible outcomes:

1. At line S8, P is too close to P_1 or P_2 to be a different internal node.
2. At line S15, P is outside the path between P_1 and P_2 in T and thus should not be inserted into T^*_k on P_1P_2.
3. At line S16, P is strictly between P_1 and P_2 in T. Thus, P can be inserted between P_1 and P_2 in T^*_k, and the lengths $\Delta^*_{P_1P}, \Delta^*_{P_2P}, \Delta^*_{NP}$ of the possible

new edges P_1P, P_2P, and NP are returned.

In the case of the third outcome, NXY is called a *splitting triplet* for P_1P_2 in T_k^*, and $\langle P_1P_2, NXY, P, \Delta^*_{P_1P}, \Delta^*_{P_2P}, \Delta^*_{NP}\rangle$ is a *splitting tuple*. Each $\mathcal{S}[N]$ is either a single splitting tuple or null. In the latter case, the estimated closeness of the triplet in $\mathcal{S}[N]$ is regarded as 0 for technical uniformity.

Fast-HGT ensures the accuracy of T^* in several ways. The algorithm uses only positive triplets to recover internal nodes of T at lines F1 and F9. These two lines together form the greedy strategy of Fast-HGT. The maximality of the triplet chosen at these two lines favors large triplets over small ones based on Lemmas 2.6 and 2.7. With a relevant triplet as input, Split-Edge compares P to P_1 and P_2 using the rule of (2.10) and can estimate the distance between P and P_1 or P_2 from the same leaf to avoid accumulating estimation errors in edge lengths.

The next lemma enables Fast-HGT to grow T^* by always using relevant triplets.

LEMMA 3.1. *For each $k = 3, \ldots, n-1$, at the start of the $(k-2)$th iteration of the repeat at line* F7, $\operatorname{def}(P_1) \cap \operatorname{def}(P_2) \neq \emptyset$ *for every edge $P_1P_2 \in T_k^*$.*

Proof. The proof is by induction on k. The base case follows from the fact that the statement holds for T_3^* at line F3. The induction step follows from the use of a relevant triplet at line F9. ∎

Remark. A subsequence work [6] shows that Fast-HGT can run with the same time, space, and sample complexities without knowing f and $\Delta_{\min}$; this is achieved by slightly modifying some parts of Split-Edge.

3.2. The running time and work space of Fast-HGT. Before proving the desired time and space complexities of Fast-HGT in Theorem 3.2 below, we note the following three key techniques used by Fast-HGT to save time and space.

1. At line F1, ABC is selected for a fixed arbitrary A. This limits the number of triplets considered at line F1 to $\mathcal{O}(n^2)$. This technique is supported by the fact that each leaf in T is contained in a large triplet.
2. At lines F6 and F14, $\mathcal{S}$ keeps only splitting tuples. This limits the number of triplets considered for each involved edge to $\mathcal{O}(n)$. This technique is feasible since by Lemma 3.5, $\Psi(T)$ can be recovered using only relevant triplets.
3. At line F14, $\mathcal{S}$ includes no new splitting tuples for the edges Q_1Q_2 that already exist in T^* before N is inserted. This technique is feasible because the insertion of N results in no new relevant triplets for such Q_1Q_2 at all.

THEOREM 3.2. Fast-HGT *runs in $\mathcal{O}(n^2)$ time using $\mathcal{O}(n)$ work space.*

Proof. We analyze the time and space complexities separately as follows.

Time complexity. Line F1 takes $\mathcal{O}(n^2)$ time. Line F6 takes $\mathcal{O}(n)$ total time to examine $2(n-3)$ triplets for each Q_1Q_2. As for the repeat at line F7, lines F8, F9, and F13 take $\mathcal{O}(n)$ time to search through $\mathcal{S}$. For the $(k-3)$th iteration of the repeat where $k = 4, \ldots, n-1$, line F14 takes $\mathcal{O}(n)$ total time to examine at most $9(n-k-1)$ triplets for each of P_1P, P_2P, and NP. Thus, each iteration of the repeat takes $\mathcal{O}(n)$ time. Since the repeat iterates at most $n-3$ times, the time complexity of Fast-HGT is as stated.

Space complexity. T^* and the sets $\operatorname{def}(G)$ for all nodes G in T^* take $\mathcal{O}(n)$ work space. $\mathcal{S}$ takes $\mathcal{O}(n)$ space. Lines F1, F6, and F14 in Fast-HGT and lines U1 and U2 in Update-$\mathcal{S}$ can be implemented to use $\mathcal{O}(1)$ space. The other variables needed by Fast-HGT take $\mathcal{O}(1)$ space. Thus, the space complexity of Fast-HGT is as stated. ∎

3.3. Technical lemmas for bounding the sample size. Let L_k be the set of the leaves of $\Psi(T)$ that are in T_k^*. Let Ψ_k be the subtree of $\Psi(T)$ formed by the edges

on paths between leaves in L_k. A *branchless* path in Ψ_k is one whose internal nodes are all of degree 2 in Ψ_k. We say that T_k^* *matches* T if T_k^* without the edge lengths can be obtained from Ψ_k by replacing every maximal branchless path with an edge between its two endpoints.

For $k = 3, \ldots, n$, we define the following conditions:

- $\mathcal{A}_k$: T_k^* matches T.
- $\mathcal{B}_k$: For every internal node $Q \in T_k^*$, the triplet formed by $\mathrm{def}(Q)$ is not small.
- $\mathcal{C}_k$: For every edge $Q_1Q_2 \in T_k^*$, $|\Delta^*_{Q_1Q_2} - \Delta_{Q_1Q_2}| < 2\Delta_{\min}$.

In this section, Lemmas 3.3, 3.4, and 3.5 analyze under what conditions Split-Edge can help correctly insert a new leaf and a new internal node to T_k^*. Later in section 3.4, we use these lemmas to show by induction in Lemma 3.7 that the events $\mathcal{E}_{\mathrm{g}}$ and $\mathcal{E}_{\mathrm{c}}$, which are defined before Lemma 2.9, imply that $\mathcal{A}_k$, $\mathcal{B}_k$, and $\mathcal{C}_k$ hold for all k. This leads to Theorem 3.8, stating that Fast-HGT solves the weighted evolutionary topology problem with a polynomial-sized sample.

Lemmas 3.3, 3.4, and 3.5 make the following assumptions for some $k < n$:

- The $(k-3)$th iteration of the repeat at line F7 has been completed.
- T_k^* has been constructed, and $\mathcal{A}_k$, $\mathcal{B}_k$, and $\mathcal{C}_k$ hold.
- Fast-HGT is currently in the $(k-2)$th iteration of the repeat.

LEMMA 3.3. *Assume that $\mathcal{E}_{\mathrm{c}}$ holds and the triplet NXY input to Split-Edge is not small. Then, the test of line* S7 *fails if and only if $P \neq P_1$ and $P \neq P_2$ in T.*

Proof. There are two directions, both using the following equation. By line S6,

$$\Delta_1 = (\hat{\Delta}_{X_1P} - \Delta_{X_1P}) - (\hat{\Delta}_{X_1P_1} - \Delta_{X_1P_1}) + (\Delta_{X_1P} - \Delta_{X_1P_1}). \tag{3.1}$$

($\Longrightarrow$) To prove by contradiction, assume $P = P_1$ or $P = P_2$ in T. If $P = P_1$, then $\Delta_{X_1P} = \Delta_{X_1P_1}$, and by $\mathcal{A}_k$, P_1 is an internal node in T_k^*. By $\mathcal{B}_k$, the triplet formed by $\mathrm{def}(P_1)$ is not small. Thus, by $\mathcal{E}_{\mathrm{c}}$ and (3.1), $|\Delta_1| < \Delta_{\min}$. By symmetry, if $P = P_2$, then $|\Delta_2| < \Delta_{\min}$. In either case, the test of line S7 passes.

($\Longleftarrow$) Since $P \neq P_1$, $\Delta_{X_1P} - \Delta_{X_1P_1} \geq -\ln(1-\alpha f) \geq 2\Delta_{\min}$. If P_1 is a leaf in T_k^*, then by $\mathcal{A}_k$, P_1 is leaf X_1 in T, and $\hat{\Delta}_{X_1P_1} = \Delta_{X_1P_1} = 0$. By $\mathcal{E}_{\mathrm{c}}$ and (3.1), $|\Delta_1| > 1.5\Delta_{\min}$. If P_1 is an internal node in T_k^*, then by $\mathcal{B}_k$, $\mathcal{E}_{\mathrm{c}}$, and (3.1), we have $|\Delta_1| > \Delta_{\min}$. In either case, $|\Delta_1| > \Delta_{\min}$. By symmetry, since $P \neq P_2$, $|\Delta_2| > \Delta_{\min}$. Thus, the test of line S7 fails. ∎

LEMMA 3.4. *In addition to the assumption in Lemma* 3.3*, also assume that $P \neq P_1$ and $P \neq P_2$ in T, i.e., the test of line* S7 *has failed. Then, the test of line* S14 *fails if and only if P is on the path between P_1 and P_2 in T.*

Proof. There are two directions.

($\Longleftarrow$) From lines S6, S10, and Corollary 2.3(2),

$$\begin{aligned}(\Delta_1' - \Delta_2') - (\Delta_{P_1P} - \Delta_{P_2P}) = &\pm \left((\hat{\Delta}_{X_1P} - \Delta_{X_1P}) - (\hat{\Delta}_{X_1P_1} - \Delta_{X_1P_1})\right) \\ &\pm \left((\hat{\Delta}_{X_2P} - \Delta_{X_2P}) - (\hat{\Delta}_{X_2P_2} - \Delta_{X_2P_2})\right).\end{aligned}$$

Thus, whether P_1 and P_2 are leaves or internal nodes in T_k^*, by $\mathcal{A}_k$, $\mathcal{B}_k$, and $\mathcal{E}_{\mathrm{c}}$, $|(\Delta_1' - \Delta_2') - (\Delta_{P_1P} - \Delta_{P_2P})| < 2\Delta_{\min}$. By line S13 and Corollary 2.3(2),

$$\begin{aligned}\Delta_1'' &< \frac{2\Delta_{\min} + (\Delta_{P_1P} - \Delta_{P_2P}) + \Delta^*_{P_1P_2}}{2} \\ &= \frac{2(2\Delta_{\min} - \Delta_{P_2P}) + (-2\Delta_{\min} + \Delta_{P_1P_2}) + \Delta^*_{P_1P_2}}{2}.\end{aligned}$$

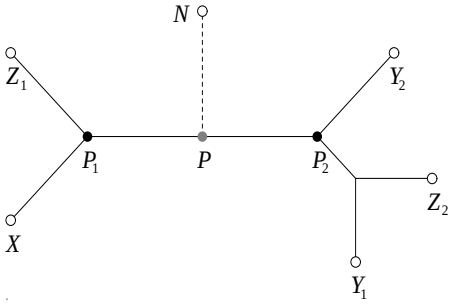


FIG. 3.4. *This subgraph of T fixes some notation used in the proof of Case* 1 *of Lemma* 3.5. *The location of Y_1 relative to Y_2 and Z_2 is nonessential; for instance, Y_1 can even be the same as Y_2. In T_k^*, $\mathrm{def}(P_1) = \{X, Y_1, Z_1\}$ and $\mathrm{def}(P_2) = \{X, Y_2, Z_2\}$. Neither XY_1Z_1 nor XY_2Z_2 is small, and $\Delta_{P_2Y_2} \le \Delta_{P_2Z_2}$. We aim to prove that there is a leaf $N \notin T_k^*$ such that NXY_2 or NZ_1Y_2 is large and defines a node P strictly between P_1 and P_2 in T.*

Then, since $P \neq P_2$ and thus $\Delta_{P_2P} \ge 2\Delta_{\min}$, by $\mathcal{C}_k$, we have $\Delta_1'' < \Delta^*_{P_1P_2}$. By symmetry, $\Delta_2'' < \Delta^*_{P_1P_2}$. Thus, the test of line S14 fails.

($\Longrightarrow$) To prove by contradiction, assume that P is not on the path between P_1 and P_2. By similar arguments, if $\Delta_{P_1P} > \Delta_{P_1P_2}$ (respectively, $\Delta_{P_2P} > \Delta_{P_1P_2}$), then $\Delta_1'' > \Delta^*_{P_1P_2}$ (respectively, $\Delta_2'' > \Delta^*_{P_1P_2}$). Thus, the test of line S14 passes. ∎

LEMMA 3.5. *Assume that P_1P_2 is an edge in T_k^* and some node is strictly between P_1 and P_2 in T. Then there is a large triplet NQ_1Q_2 with center P such that $N \notin T_k^*$, $Q_1 \in \mathrm{def}(P_1)$, $Q_2 \in \mathrm{def}(P_2)$, and P is strictly between P_1 and P_2 in T.*

Proof. By Lemma 2.6(2), for every node P strictly between P_1 and P_2 in T, there exists a leaf $N \notin T_k^*$ with $\sigma_{PN} \ge (1-\alpha g)^{d+1}$. To choose P, there are two cases: (1) both P_1 and P_2 are internal nodes in T_k^*, and (2) P_1 or P_2 is a leaf in T_k^*.

Case 1: By Lemma 3.1, let $\mathrm{def}(P_1) = \{X, Y_1, Z_1\}$ and $\mathrm{def}(P_2) = \{X, Y_2, Z_2\}$. By $\mathcal{B}_k$, neither XY_2Z_2 nor XY_1Z_1 is small. To fix the notation for $\mathrm{def}(P_1)$ and $\mathrm{def}(P_2)$ with respect to their topological layout, we assume without loss of generality that Figure 3.4 or equivalently the following statements hold:

- In T_k^* and thus in T by $\mathcal{A}_k$, P_2 is on the paths between P_1 and Y_2, between P_1 and Z_2, and between P_1 and Y_1, respectively.
- Similarly, P_1 is on the paths between P_2 and Z_1 and between P_2 and X.
- $\Delta_{P_2Y_2} \le \Delta_{P_2Z_2}$.

Both NXY_2 and NZ_1Y_2 define P, and the target triplet is one of these two for some suitable P. To choose P, we further divide Case 1 into three subcases.

Case 1a: $\sigma_{XP_2} < \sigma_{Y_2P_2}(1-\alpha g)$ and $\sigma_{Y_2P_1} < \sigma_{XP_1}(1-\alpha g)$. The target triplet is NXY_2. Since $\sigma_{XY_2} \le \sqrt{\sigma_{XY_2}}$, by Corollary 2.3(3), let P be a node on the path between X and Y_2 in T with $\sqrt{\sigma_{XY_2}(1-\alpha g)} \le \sigma_{XP} \le \sqrt{\sigma_{XY_2}(1-\alpha g)^{-1}}$ and thus by Lemma 2.1 $\sqrt{\sigma_{XY_2}(1-\alpha g)} \le \sigma_{Y_2P} \le \sqrt{\sigma_{XY_2}(1-\alpha g)^{-1}}$. By the condition of Case 1a and Lemma 2.1, P is strictly between P_1 and P_2 in T. Also, by Corollary 2.4, $\sigma_{XY_2} \ge \frac{2}{3}\sigma_{XY_2Z_2}$. Thus, by Lemma 2.1, since XY_2Z_2 is not small,

$$\sigma_{NXY_2} = \frac{3}{\frac{1}{\sigma_{XP}\sigma_{PN}} + \frac{1}{\sigma_{Y_2P}\sigma_{PN}} + \frac{1}{\sigma_{XY_2}}} \ge \frac{1}{\sqrt{\frac{2}{3}}\sigma_{XY_2Z_2}^{-1/2}(1-\alpha g)^{-d-3/2} + \frac{1}{2}\sigma_{XY_2Z_2}^{-1}} > \sigma_{\mathrm{lg}}. \tag{3.2}$$

So NXY_2 is as desired for Case 1a.

Case 1b: $\sigma_{XP_2} \geq \sigma_{Y_2P_2}(1-\alpha g)$. The target triplet is NXY_2. Let P be the first node after P_2 on the path from P_2 toward P_1 in T. Then, $\sigma_{Y_2P} \geq \sigma_{Y_2P_2}(1-\alpha g)$. By Corollary 2.4, $\sigma^2_{Y_2P} \geq \sigma_{XY_2Z_2}(1-\alpha g)^2/3$. Next, since $\sigma_{XY_2} \geq \sigma_{XZ_2}$ and $\sigma_{P_2Y_2} \geq \sigma_{P_2Z_2}$,

$$\sigma_{XY_2Z_2} \leq \frac{3}{2\sigma^{-1}_{XY_2} + \sigma^{-1}_{Y_2P_2}\sigma^{-1}_{P_2Z_2}} \leq \frac{3}{2\sigma^{-1}_{XP_2}\sigma^{-1}_{Y_2P_2} + \sigma^{-2}_{Y_2P_2}} \leq \frac{3\sigma^2_{XP_2}}{2(1-\alpha g)+(1-\alpha g)^2}.$$

So $\sigma^2_{XP} > \sigma^2_{XP_2} > \sigma_{XY_2Z_2}(1-\alpha g)^2$. Since $\sigma_{XY_2} \geq \frac{2}{3}\sigma_{XY_2Z_2}$ and XY_2Z_2 is not small,

$$\sigma_{NXY_2} = \frac{3}{\frac{1}{\sigma_{XP}\sigma_{PN}} + \frac{1}{\sigma_{Y_2P}\sigma_{PN}} + \frac{1}{\sigma_{XY_2}}} > \frac{1}{\left(\frac{1+\sqrt{3}}{3}\right)\sigma^{-1/2}_{XY_2Z_2}(1-\alpha g)^{-d-2} + \frac{1}{2}\sigma^{-1}_{XY_2Z_2}} > \sigma_{\lg}. \tag{3.3}$$

So NXY_2 is as desired for Case 1b.

Case 1c: $\sigma_{Y_2P_1} \geq \sigma_{XP_1}(1-\alpha g)$. If $\sigma_{Z_1P_1} > \sigma_{XP_1}$, the target triplet is NZ_1Y_2; otherwise, it is NXY_2. The two cases are symmetric, and we assume $\sigma_{XP_1} \geq \sigma_{Z_1P_1}$. Let P be the first node after P_1 on the path from P_1 toward P_2 in T. Then, $\sigma_{XP} \geq \sigma_{XP_1}(1-\alpha g)$. By Corollary 2.4, $\sigma^2_{XP} \geq \sigma^2_{XP_1}(1-\alpha g)^2 \geq \sigma_{XY_1Z_1}(1-\alpha g)^2/3$. Since $\sigma_{XY_2} \geq \sigma_{XZ_2}$ and $\sigma_{Y_2Z_2} > 0$,

$$\sigma_{XY_2Z_2} \;<\; \frac{3}{2\sigma^{-1}_{XY_2}} \;\leq\; \frac{3}{2\sigma^{-1}_{Y_2P_1}\sigma^{-1}_{XP_1}} \;\leq\; \frac{3\sigma^2_{Y_2P_1}}{2(1-\alpha g)}.$$

Hence $\sigma^2_{Y_2P} > \sigma^2_{Y_2P_1} > 2\sigma_{XY_2Z_2}(1-\alpha g)/3$. Then, since neither XY_2Z_2 nor XY_1Z_1 is small and $\sigma_{XY_2} \geq \frac{2}{3}\sigma_{XY_2Z_2}$,

$$\sigma_{NXY_2} = \frac{3}{\frac{1}{\sigma_{XP}\sigma_{PN}} + \frac{1}{\sigma_{Y_2P}\sigma_{PN}} + \frac{1}{\sigma_{XY_2}}} > \frac{1}{\frac{1}{\sqrt{3}}\sigma^{-1/2}_{XY_1Z_1}(1-\alpha g)^{-d-2} + \frac{1}{\sqrt{6}}\sigma^{-1/2}_{XY_2Z_2}(1-\alpha g)^{-d-3/2} + \frac{1}{2}\sigma^{-1}_{XY_2Z_2}} > \sigma_{\lg}. \tag{3.4}$$

So NXY_2 is as desired for Case 1c with $\sigma_{XP_1} \geq \sigma_{Z_1P_1}$.

Case 2: By symmetry, assume that $P_2 = X$ is a leaf in T^*_k. Since $k \geq 3$, P_1 is an internal node in T^*_k. Let $\mathrm{def}(P_1) = \{X, Y, Z\}$. By symmetry, further assume $\sigma_{YP_1} \geq \sigma_{ZP_1}$. There are two subcases. If $\sigma_{XP_1} < \sigma_{YP_1}(1-\alpha g)$, the proof is similar to that of Case 1a and the desired P is in the middle of the path between X and Y in T. Otherwise, the proof is similar to that of Case 1b and P is the first node after P_1 on the path from P_1 toward X in T. In both cases, the desired triplet is NXY. ∎

3.4. The sample size required by Fast-HGT. The next lemma analyzes $\mathcal{S}$. For $k = 3, \ldots, n-1$ and each leaf $M \in T$, let $\mathcal{S}_k[M]$ be the version of $\mathcal{S}[M]$ at the start of the $(k-2)$th iteration of the repeat at line F7.

LEMMA 3.6. *Assume that for a given $k \leq n-1$, $\mathcal{E}_{\mathrm{g}}$, $\mathcal{E}_{\mathrm{c}}$, $\mathcal{A}_{k'}$, $\mathcal{B}_{k'}$, and $\mathcal{C}_{k'}$ hold for all $k' \leq k$.*

1. *If $\mathcal{S}_k[M]$ is not null, then it is a splitting tuple for some edge in T^*_k.*

2. *If an edge $Q_1Q_2 \in T_k^*$ and a triplet MR_1R_2 with $M \notin T_k^*$ satisfy Lemma* 3.5*, then $\mathcal{S}_k[M]$ is a splitting tuple for Q_1Q_2 in T_k^* that contains a triplet $MR_1'R_2'$ with $\hat{\sigma}_{MR_1'R_2'} \geq \hat{\sigma}_{MR_1R_2}$.*

Proof. The two statements are proved as follows.

Statement 1. This statement follows directly from the initialization of $\mathcal{S}$ at line F6, the deletions from $\mathcal{S}$ at line F13, and the insertions into $\mathcal{S}$ at lines F6 and F14.

Statement 2. The proof is by induction on k.

Base case: $k = 3$. By $\mathcal{E}_{\mathrm{c}}$, $\mathcal{A}_3$, $\mathcal{B}_3$, $\mathcal{C}_3$, and Lemmas 3.3 and 3.4, MR_1R_2 is a splitting triplet for Q_1Q_2 in T_3^*. By the maximization in Update-$\mathcal{S}$ at line F6, $\mathcal{S}[M]$ is a splitting tuple for some edge $Q_1'Q_2' \in T_3^*$ that contains a triplet $MR_1'R_2'$ with $\hat{\sigma}_{MR_1'R_2'} \geq \hat{\sigma}_{MR_1R_2}$. By $\mathcal{E}_{\mathrm{g}}$, $MR_1'R_2'$ is not small. By Lemmas 3.3 and 3.4, $Q_1'Q_2'$ is Q_1Q_2.

Induction hypothesis. Statement 2 holds for $k < n - 1$.

Induction step. We consider how $\mathcal{S}_{k+1}$ is obtained from $\mathcal{S}_k$ during the $(k-2)$th iteration of the repeat at line F7. There are two cases.

Case 1: Q_1Q_2 also exists in T_k^*. By $\mathcal{A}_k$, Q_1Q_2 and MR_1R_2 also satisfy Lemmas 3.3 and 3.4 for T_k^*. By the induction hypothesis, $\mathcal{S}_k[M]$ is a splitting tuple for Q_1Q_2 in T_k^* that contains a triplet $MR_1'R_2'$ with $\hat{\sigma}_{MR_1'R_2'} \geq \hat{\sigma}_{MR_1R_2}$. Then, since $Q_1Q_2 \neq P_1P_2$ and $M \neq N$ at line F13, $\mathcal{S}_k[M]$ is not reset to null. Thus, it can be changed only through replacement at line F14 by a splitting tuple for some edge $Q_1'Q_2'$ in T_{k+1}^* that contains a triplet $MR_1''R_2''$ with $\hat{\sigma}_{MR_1''R_2''} \geq \hat{\sigma}_{MR_1'R_2'}$. By $\mathcal{E}_{\mathrm{g}}$, $MR_1''R_2''$ is not small. Thus, by $\mathcal{E}_{\mathrm{c}}$, $\mathcal{A}_{k+1}$, $\mathcal{B}_{k+1}$, $\mathcal{C}_{k+1}$, and Lemmas 3.3 and 3.4, $Q_1'Q_2'$ is Q_1Q_2.

Case 2: $Q_1Q_2 \notin T_k^*$. This case is similar to the base case but uses the maximization in Update-$\mathcal{S}$ at line F14. □

LEMMA 3.7. *$\mathcal{E}_{\mathrm{g}}$ and $\mathcal{E}_{\mathrm{c}}$ imply that $\mathcal{A}_k$, $\mathcal{B}_k$, and $\mathcal{C}_k$ hold for all $k = 3, \ldots, n$.*

Proof. The proof is by induction on k.

Base case: $k = 3$. By Lemma 2.6(2), $\mathcal{E}_{\mathrm{c}}$, and the greedy selection of line F1, line F3 constructs T_3^* without edge lengths. Then, $\mathcal{A}_3$ holds trivially. $\mathcal{B}_3$ follows from $\mathcal{E}_{\mathrm{c}}$, $\mathcal{E}_{\mathrm{g}}$, and line F1. $\mathcal{C}_3$ follows from $\mathcal{B}_3$, $\mathcal{E}_{\mathrm{c}}$ and the use of (2.7) at line F4.

Induction hypothesis. $\mathcal{A}_k$, $\mathcal{B}_k$, and $\mathcal{C}_k$ hold for some $k < n$.

Induction step. The induction step is concerned with the $(k-2)$th iteration of the repeat at line F7. Right before this iteration, by the induction hypothesis, since $k < n$, some $N'Q_1Q_2$ satisfies Lemma 3.5. Therefore, during this iteration, by $\mathcal{E}_{\mathrm{c}}$ and Lemmas 3.3, 3.4, and 3.6, $\mathcal{S}$ at line F8 has a splitting tuple for T_k^* that contains a triplet NXY with $\hat{\sigma}_{NXY} \geq \hat{\sigma}_{N'Q_1Q_2}$. Furthermore, line F9 finds such a tuple. By $\mathcal{E}_{\mathrm{g}}$, NXY is not small. Lines F10 and F11 create T_{k+1}^* using this triplet. Thus, $\mathcal{B}_{k+1}$ follows from $\mathcal{B}_k$. By Lemmas 3.3 and 3.4, $\mathcal{A}_{k+1}$ follows from $\mathcal{A}_k$. $\mathcal{C}_{k+1}$ follows from $\mathcal{C}_k$ since the triplets involved at line S13 are not small. □

THEOREM 3.8. *For any $0 < \delta < 1$, using sequence length*

$$\ell = \mathcal{O}\left(\frac{\log\frac{1}{\delta} + \log n}{(1-\alpha g)^{4d+8} f^2 c^2}\right),$$

Fast-HGT *outputs T^* with the properties below with probability at least $1 - \delta$:*

1. *Disregarding the edge lengths, $T^* = \Psi_{\mathrm{w}}(T)$.*
2. *For each edge Q_1Q_2 in T^*, $|\Delta^*_{Q_1Q_2} - \Delta_{Q_1Q_2}| < 2\Delta_{\min}$.*

Proof. By Lemma 2.9, $\Pr\{\,\overline{\mathcal{E}}_{\mathrm{g}}\,\} \leq \frac{\delta}{2}$ if

$$\ell \;\geq\; \ell_{\mathrm{g}} \;\stackrel{\mathrm{def}}{=}\; 210\alpha^2 \frac{3\ln n + \ln\frac{3}{\delta}}{\sigma^2_{\mathrm{lg}}}.$$

Similarly, by Lemma 2.9, $\Pr\{\,\overline{\mathcal{E}}_{\rm c}\,\} \le \frac{\delta}{2}$ if

$$\ell \;\ge\; \ell_{\rm c} \;\stackrel{\rm def}{=}\; 81\frac{3\ln n + \ln\frac{7}{\delta}}{\sigma_{\rm lg}^2 f^2 c^2}.$$

We choose $\ell = \lceil \max\{\ell_{\rm g}, \ell_{\rm c}\}\rceil$. Consequently, $\Pr\{\,\mathcal{E}_{\rm g} \text{ and } \mathcal{E}_{\rm c}\,\} \ge 1-\delta$. By Lemma 3.7, with probability at least $1-\delta$, Fast-HGT outputs T_n^*, and $\mathcal{A}_n$ and $\mathcal{C}_n$ hold, which correspond to the two statements of the theorem. □

4. Further research. We have shown that theoretically, Fast-HGT has the optimal time and space complexity as well as a polynomial sample complexity. It would be important to determine the practical performance of the algorithm by testing it extensively on empirical and simulated trees and sequences. Furthermore, as conjectured by one of the referees and some other researchers, there might be a trade-off between the time complexity and the practical performance. If this is indeed true empirically, it would be significant to quantify the trade-off analytically.

Appendix. Proofs of technical lemmas.

A.1. Proof of Lemma 2.5. Let $h_{XY} = \frac{\hat\sigma_{XY}}{\sigma_{XY}}$; $h_{XZ} = \frac{\hat\sigma_{XZ}}{\sigma_{XZ}}$; $h_{YZ} = \frac{\hat\sigma_{YZ}}{\sigma_{YZ}}$. By (2.6) and (2.7), and by conditioning on the events $\{h_{XZ} \le 1-r\}$ and $\{h_{YZ} \ge 1+s\}$ for some $r, s > 0$,

$$\Pr\left\{\,\hat\Delta_{XP} - \Delta_{XP} \ge \frac{-\ln(1-\epsilon)}{2}\,\right\} = \Pr\{\,h_{XY}h_{XZ} \le h_{YZ}(1-\epsilon)\,\}$$
$$\le \Pr\{\,h_{XZ} \le 1-r\,\} + \Pr\{\,h_{YZ} \ge 1+s\,\} + \Pr\left\{\,h_{XY} \le (1-\epsilon)\frac{1+s}{1-r}\,\right\}.$$

Setting $\frac{1-r}{1+s} > 1-\epsilon$, by (2.3) and (2.4),

$$\Pr\left\{\,\hat\Delta_{XP} - \Delta_{XP} \ge \frac{-\ln(1-\epsilon)}{2}\,\right\} \le$$
$$\exp\left(-\frac{2}{\alpha^2}\ell\sigma_{XZ}^2 r^2\right) + \exp\left(-\frac{2}{\alpha^2}\ell\sigma_{YZ}^2 s^2\right) + \exp\left(-\frac{2}{\alpha^2}\ell\sigma_{XY}^2\left(1-(1-\epsilon)\frac{1+s}{1-r}\right)^2\right).$$

Equating these exponential terms yields equations for r and s. The solution for r is

$$r = \frac{t-\sqrt{t^2-u}}{2\sigma_{XZ}\sigma_{YZ}};\; t = \sigma_{XY}\sigma_{YZ} + \sigma_{XZ}\sigma_{YZ} + (1-\epsilon)\sigma_{XY}\sigma_{XZ};\; u = 4\sigma_{XY}\sigma_{YZ}^2\sigma_{XZ}\epsilon.$$

Using Taylor's expansion, for $u > 0$, $(t-\sqrt{t^2-u})^2 > \frac{u^2}{4t^2}$. Thus,

$$r^2 > \frac{\epsilon^2}{\left(\frac{1}{\sigma_{XZ}} + \frac{1-\epsilon}{\sigma_{YZ}} + \frac{1}{\sigma_{XY}}\right)^2\sigma_{XZ}^2} > \frac{\epsilon^2\sigma_{XYZ}^2}{9\sigma_{XZ}^2}.$$

So $\Pr\left\{\,\hat\Delta_{XP} - \Delta_{XP} \ge \frac{-\ln(1-\epsilon)}{2}\,\right\} \le 3\exp\left(-\frac{2}{\alpha^2}\ell\sigma_{XZ}^2 r^2\right) < 3\exp\left(-\frac{2}{9\alpha^2}\ell\sigma_{XYZ}^2\epsilon^2\right)$.

A.2. Proof of Lemma 2.7. We use the following basic inequalities.

$$\min\left\{\frac{\hat\sigma_{XY}}{\sigma_{XY}}, \frac{\hat\sigma_{XZ}}{\sigma_{XZ}}, \frac{\hat\sigma_{YZ}}{\sigma_{YZ}}\right\} \le \frac{\hat\sigma_{XYZ}}{\sigma_{XYZ}} \le \max\left\{\frac{\hat\sigma_{XY}}{\sigma_{XY}}, \frac{\hat\sigma_{XZ}}{\sigma_{XZ}}, \frac{\hat\sigma_{YZ}}{\sigma_{YZ}}\right\}; \tag{A.1}$$

$$\frac{\sigma_{XYZ}}{3} \leq \min\{\sigma_{XY}, \sigma_{XZ}, \sigma_{YZ}\}. \tag{A.2}$$

The proof of (2.9) is symmetric to that of (2.8). So we only prove the latter. Pick $\lambda \geq 1$ with $\sigma_{XYZ} = \sigma_{\mathrm{lg}}\lambda$. Without loss of generality, we assume $\min\{\frac{\hat{\sigma}_{XY}}{\sigma_{XY}}, \frac{\hat{\sigma}_{XZ}}{\sigma_{XZ}}, \frac{\hat{\sigma}_{YZ}}{\sigma_{YZ}}\} = \frac{\hat{\sigma}_{XY}}{\sigma_{XY}}$. By (2.3), (A.1), and (A.2),

$$\begin{aligned}
&\Pr\{\,\hat{\sigma}_{XYZ} \leq \sigma_{\mathrm{md}}\,\} = \Pr\left\{\,\frac{\hat{\sigma}_{XYZ}}{\sigma_{XYZ}} \leq \frac{\sigma_{\mathrm{md}}}{\sigma_{\mathrm{lg}}\lambda}\,\right\} \leq \Pr\left\{\,\frac{\hat{\sigma}_{XY}}{\sigma_{XY}} \leq \frac{\sigma_{\mathrm{md}}}{\sigma_{\mathrm{lg}}\lambda}\,\right\} \\
&\leq \exp\left(-\frac{2}{\alpha^2}\ell\left(1-\frac{\sigma_{\mathrm{md}}}{\sigma_{\mathrm{lg}}\lambda}\right)^2\sigma_{XY}^2\right) \leq \exp\left(-\frac{2\left(1-\frac{\sigma_{\mathrm{md}}}{\sigma_{\mathrm{lg}}}\right)^2}{9\alpha^2}\ell\sigma_{\mathrm{lg}}^2\right).
\end{aligned}$$

Then, (2.8) follows from the fact that by the choice of σ_{md},

$$\frac{2\left(1-\frac{\sigma_{\mathrm{md}}}{\sigma_{\mathrm{lg}}}\right)^2}{9\alpha^2} = \frac{(\sqrt{2}-1)^2}{36\alpha^2}.$$

A.3. Proof of Lemma 2.8. Since Lemma 2.5 can help establish only one half of the desired inequality, we split the probability on the left-hand side of (2.11):

$$\begin{aligned}
&\Pr\left\{\,\left|\hat{\Delta}_{XP} - \Delta_{XP}\right| \geq \frac{\Delta_{\min}}{2}\,\right\} \\
&\leq \Pr\left\{\,\hat{\Delta}_{XP} - \Delta_{XP} \geq \frac{\Delta_{\min}}{6}\,\right\} + \Pr\left\{\,\hat{\Delta}_{YP} - \Delta_{YP} \geq \frac{\Delta_{\min}}{6}\,\right\} \\
&\quad + \Pr\left\{\,\hat{\Delta}_{XP} - \Delta_{XP} \leq -\frac{\Delta_{\min}}{2} \;\middle|\; \hat{\Delta}_{YP} - \Delta_{YP} < \frac{\Delta_{\min}}{6}\,\right\}.
\end{aligned}$$

Then, since $\hat{\Delta}_{XY} - \Delta_{XY} = (\hat{\Delta}_{XP} - \Delta_{XP}) + (\hat{\Delta}_{YP} - \Delta_{YP})$, we have

$$\begin{aligned}
&\Pr\left\{\,\hat{\Delta}_{XP} - \Delta_{XP} \leq -\frac{\Delta_{\min}}{2} \;\middle|\; \hat{\Delta}_{YP} - \Delta_{YP} < \frac{\Delta_{\min}}{6}\,\right\} \\
&\leq \Pr\left\{\,\hat{\Delta}_{XY} - \Delta_{XY} \leq -\frac{\Delta_{\min}}{3}\,\right\}.
\end{aligned}$$

Consequently,

$$\begin{aligned}
\Pr\left\{\,\left|\hat{\Delta}_{XP} - \Delta_{XP}\right| \geq \frac{\Delta_{\min}}{2}\,\right\} &\leq \Pr\left\{\,\hat{\Delta}_{XP} - \Delta_{XP} \geq \frac{\Delta_{\min}}{6}\,\right\} \\
&\quad + \Pr\left\{\,\hat{\Delta}_{YP} - \Delta_{YP} \geq \frac{\Delta_{\min}}{6}\,\right\} \\
&\quad + \Pr\left\{\,\hat{\Delta}_{XY} - \Delta_{XY} \leq -\frac{\Delta_{\min}}{3}\,\right\}.
\end{aligned} \tag{A.3}$$

By Lemma 2.5,

$$\Pr\left\{\,\hat{\Delta}_{XP} - \Delta_{XP} \geq \frac{\Delta_{\min}}{6}\,\right\} \leq 3\exp\left(-\frac{2}{9\alpha^2}\ell\sigma_{XYZ}^2\left(1-e^{-\frac{\Delta_{\min}}{3}}\right)^2\right).$$

By Taylor's expansion, $(1-e^{-\frac{\Delta_{\min}}{3}})^2 \geq \left(1-(1-\alpha f)^{\frac{c}{3}}\right)^2 > \frac{c^2}{9}\alpha^2 f^2$, and thus

$$\Pr\left\{ \hat{\Delta}_{XP} - \Delta_{XP} \geq \frac{\Delta_{\min}}{6} \right\} \leq 3\exp\left(-\frac{c^2}{81}\ell\sigma_{\lg}^2 f^2\right). \tag{A.4}$$

By symmetry,

$$\Pr\left\{ \hat{\Delta}_{YP} - \Delta_{YP} \geq \frac{\Delta_{\min}}{6} \right\} \leq 3\exp\left(-\frac{c^2}{81}\ell\sigma_{\lg}^2 f^2\right). \tag{A.5}$$

By (2.4), $\Pr\{\hat{\Delta}_{XY} - \Delta_{XY} \leq -\frac{\Delta_{\min}}{3}\} \leq \exp(-\frac{2}{\alpha^2}\ell\sigma_{XY}^2(e^{\frac{\Delta_{\min}}{3}} - 1)^2)$. From (A.2), $\sigma_{XY} \geq \frac{(1-\alpha g)^{2d+2}}{3\sqrt{2}}$. By Taylor's expansion, $(e^{\frac{\Delta_{\min}}{3}} - 1)^2 \geq ((1-\alpha f)^{-\frac{c}{3}} - 1)^2 > \frac{c^2}{9}\alpha^2 f^2$. Therefore,

$$\Pr\left\{ \hat{\Delta}_{XY} - \Delta_{XY} \leq -\frac{\Delta_{\min}}{3} \right\} \leq \exp\left(-\frac{c^2}{81}\ell\sigma_{\lg}^2 f^2\right). \tag{A.6}$$

Lemma 2.8 follows from the fact that putting (A.3) through (A.6) together, we have $\Pr\{|\hat{\Delta}_{XY} - \Delta_{XY}| \geq \frac{\Delta_{\min}}{2}\} \leq 7\exp(-\frac{c^2}{81}\ell\sigma_{\lg}^2 f^2)$.

Acknowledgments. We thank Dana Angluin, Kevin Atteson, Joe Chang, Junhyong Kim, Stan Eisenstat, Tandy Warnow, and the anonymous referees for extremely helpful discussions and comments.

REFERENCES

[1] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup, *On the approximability of numerical taxonomy* (*fitting distances by tree metrics*), SIAM J. Comput., 28 (1999), pp. 1073–1085.

[2] A. Ambainis, R. Desper, M. Farach, and S. Kannan, *Nearly tight bounds on the learnability of evolution*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Santa Fe, NM, 1997, pp. 524–533.

[3] K. Atteson, *The performance of neighbor-joining algorithms of phylogeny reconstruction*, Algorithmica, 25 (1999), pp. 251–278.

[4] M. Cryan, L. A. Goldberg, and P. W. Goldberg, *Evolutionary trees can be learned in polynomial time in the two-state general Markov-model*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Palo Alto, CA, 1998, pp. 436–445.

[5] M. Csűrös and M. Y. Kao, *Recovering evolutionary trees through harmonic greedy triplets*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 261–270.

[6] M. Csuros, *Reconstructing Phylogenies in Markov Models of Evolution*, Ph.D. thesis, Yale University, 2000.

[7] W. H. E. Day, *Computational complexity of inferring phylogenies from dissimilarity matrices*, Bulletin of Mathematical Biology, 49 (1997), pp. 461–467.

[8] W. H. E. Day, D. S. Johnson, and D. Sankoff, *The computational complexity of inferring rooted phylogenies by parsimony*, Math. Biosci., 81 (1986), pp. 33–42.

[9] D.-Z. Du, Y.-J. Zhang, and Q. Feng, *On better heuristic for Euclidean Steiner minimum trees* (*extended abstract*), in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, San Juan, Puerto Rico, 1991, pp. 431–439.

[10] P. L. Erdős, M. A. Steel, L. A. Székely, and T. J. Warnow, *A few logs suffice to build (almost) all trees.* I, Random Structures Algorithms, 14 (1999), pp. 153–184.

[11] P. L. Erdős, M. A. Steel, L. A. Székely, and T. J. Warnow, *A few logs suffice to build (almost) all trees.* II, Theoret. Comput. Sci., 221 (1999), pp. 77–118.

[12] M. Farach and S. Kannan, *Efficient algorithms for inverting evolution*, J. ACM, 46 (1999), pp. 437–449.

[13] J. Felsenstein, *Numerical methods for inferring evolutionary trees*, The Quarterly Review of Biology, 57 (1982), pp. 379–404.
[14] J. Felsenstein, *Inferring evolutionary trees from DNA sequences*, in Statistical Analysis of DNA Sequence Data, B. Weir, ed., Dekker, New York, 1983, pp. 133–150.
[15] J. Felsenstein, *Statistical inference of phylogenies*, J. Roy. Statist. Soc. Ser. A, 146 (1983), pp. 246–272.
[16] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, New York, 1997.
[17] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.
[18] D. Huson, S. Nettles, and T. Warnow, *Disk-covering, a fast converging method for phylogenetic tree reconstruction*, Journal of Computational Biology, 6 (1999), pp. 369–386.
[19] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie, *On the learnability of discrete distributions (extended abstract)*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, ACM, Montreal, Quebec, Canada, 1994, pp. 273–282.
[20] M. E. Siddall, *Success of parsimony in the four-taxon case: Long-branch repulsion by likelihood in the Farris zone*, Cladistics, 14 (1998), pp. 209–220.
[21] M. Steel, *Recovering a tree from the leaf colourations it generates under a Markov model*, Appl. Math. Lett., 7 (1994), pp. 19–23.
[22] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis, *Phylogenetic inference*, in Molecular Systematics, 2nd ed., D. M. Hillis, C. Moritz, and B. K. Mable, eds., Sinauer Associates, Sunderland, MA, 1996, pp. 407–514.

SIAM J. COMPUT.
Vol. 31, No. 1, pp. 323–330

PROPORTION EXTEND SORT*

JING-CHAO CHEN†

Abstract. PROPORTION EXTEND SORT is a new sorting algorithm, the basic principle of which is similar to PROPORTION SPLIT SORT. This algorithm sorts a sequence by constructing three subproblems, using a QuickSort-like pivot technique and solving recursively each subproblem. The original problem and three subproblems all are of such a structure: a sorted subsequence followed by an unsorted subsequence. The size of the original problem always equals the size of the third subproblem, but in general, the sorted subsequence of the third subproblem is $p+1$ times as much as the sorted subsequence of the original, where p is a fixed positive constant. The worst case number of comparisons required by this algorithm is less than $1/\log(1+1/(2p^2+2p-1))n\log n$ for $p \geq 1$. Empirical results show that the average number of comparisons is close to $n\log n - O(n)$ for some p. From our experiments for sorting integers, when $p = 16$, this algorithm is yet faster, on average, than PROPORTION SPLIT SORT which is faster than CLEVER QUICKSORT.

Key words. algorithm, partition, sort, quick sort, insertion sort

AMS subject classifications. 68P10, 68W40

PII. S0097539798342903

1. Introduction.

It has been studied for a long time whether there exists a simple, practical, and efficient sorting algorithm that sorts n elements using constant extra space and making $O(n\log n)$ comparisons in the worst case, and has an expected number of comparisons approaching $\log(n!) = n\log n - 1.442695n$. (All logarithms throughout the paper are base two.) Notice, $\log(n!)$ is the lower bound on the worst and average case number of comparisons for comparison based sequential sorting algorithms.

MERGESORT, BINARY INSERTION SORT, and WEAK-HEAP SORT [6], etc. are sorting algorithms approaching this lower bound in terms of the number of comparisons. MERGESORT requires $n\log n - 1.2645n$ comparisons [1], [2] on average, but uses extra storage of length n. BINARY INSERTION SORT is not efficient in practice, primarily because of its $O(n^2)$ data movements. WEAK-HEAP SORT introduced by Dutton(1993) is the fastest variant of HEAPSORT [3], [5], and the average number is conjectured to be approximately $(n-0.5)\log n - 0.413n$. Unfortunately, WEAK-HEAP SORT uses n extra bits, and by the empirical results of [8], the comparison plus exchange total required exceeds that required by the best-of-three version of QUICKSORT called CLEVER QUICKSORT.

Most versions of QUICKSORT [4] require $O(n^2)$ comparisons in the worst case but $O(n\log n)$ comparisons in the average case. CLEVER QUICKSORT is a practical version of QUICKSORT and runs in approximately $1.188(n+1)\log(n-1) - 2.255n + 2.507$ (see [7]) comparisons on average.

PROPORTION SPLIT SORT introduced by Chen [8] splits a sequence into two blocks in the ratio of $1 : p-1$, then divides them into four blocks in a QuickSort-

*Received by the editors August 4, 1998; accepted for publication (in revised form) November 27, 2000; published electronically July 31, 2001. This work was partially supported by the National Natural Science Foundation of China grant 69873033.

http://www.siam.org/journals/sicomp/31-1/34290.html

†Department of Computer Science, Shanghai Jiaotong University, 1954 Huashan-Road, Shanghai 200030, People's Republic of China. Current address: Bell Labs Research China, Lucent Technologies, 15/F, Aero Space Great Wall Building, No. 30, Hai Dian Nan Lu, Beijing, 100080, People's Republic of China (jchen33@lucent.com).

like pivot step, finally, sorts recursively the left two blocks and the right two blocks separately. The worst case number of comparisons of PROPORTION SPLIT SORT is bounded by $1/\log(2p/(2p-1))n\log n$ for $p > 1$ [8]. The simulation results of [8] revealed that for some p (e.g., $p = 16$), the average number of comparisons and exchanges required by PROPORTION SPLIT SORT is fewer than that required by CLEVER QUICKSORT.

This paper introduces a new algorithm called PROPORTION EXTEND SORT. The basic principle of this algorithm is similar to PROPORTION SPLIT SORT. PROPORTION EXTEND SORT assumes that a sequence to be sorted has such a structure: a sorted subsequence followed by an unsorted subsequence, then constructs three subproblems which have the same structure as the original, and solves recursively each subproblems. In general, the sorted subsequence of the third subproblem is $p+1$ times as much as the sorted subsequence of the original, where p is a positive constant given. The worst case number of comparisons of this algorithm is bounded by $1/\log(1+1/(2p^2+2p-1))n\log n$ for $p \geq 1$. By experimental results, the performance of PROPORTION EXTEND SORT is better than that of PROPORTION SPLIT SORT.

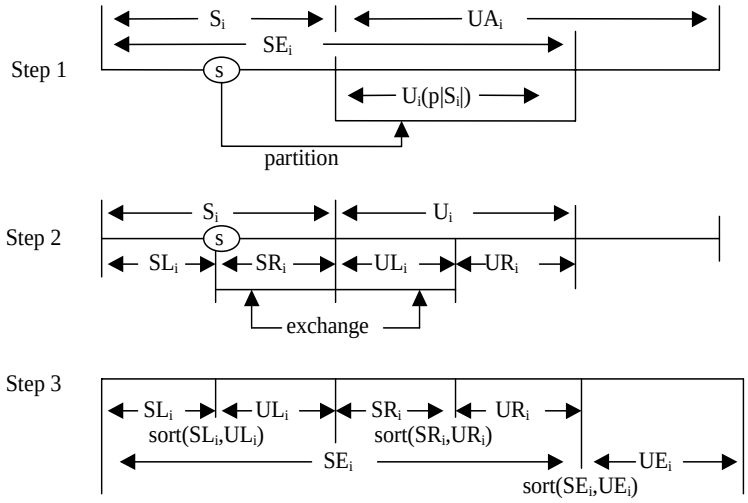


FIG. 2.1. *The basic principle of the sorting algorithm.*

2. The algorithm. The algorithm assumes that an initial array of arbitrary values $A[1..n]$ is given; it then sorts the array A in ascending order. We will design and analyze the algorithm by the following subarrays: $S_i = A[s1..s2]$, $SE_i = A[se1..se2]$, $U_i = A[u1..u2]$, $UA_i = A[ua1..ua2]$, $UL_i = A[ul1..ul2]$, $UR_i = A[ur1..ur2]$, $SL_i = A[sl1..sl2]$, $SR_i = A[sr1..sr2]$, and $UE_i = A[ue1..ue2]$. $|S_i|$ will denote the length of S_i. `Swap`(i, j), which appears below, is a procedure that interchanges the values in $A[i]$ and $A[j]$. The basic idea of PROPORTION EXTEND SORT is to regard the array A as a sequence which has such a structure: a sorted subsequence S_i followed by an unsorted subsequence UA_i, create three subproblems: (SL_i, UL_i), (SR_i, UR_i), and (SE_i, UE_i), using a QuickSort-like pivot technique, and solve recursively each subproblem. The basic idea of the algorithm is shown in Figure 2.1.

Step 1. Suppose that the first $|S_i|$ elements are sorted, and we want to extend that to $|SE_i|(=(1+p)|S_i|)$ elements, where p is a positive constant. That is, $se1 := s1$, $se2 := s1+(1+p)|S_i|-1$. Let the median of S_i be s and the unsorted subsequence in the SE_i be U_i. Use s to partition U_i into two subsequences UL_i and UR_i in a QuickSort-like pivot step, such that $\max(UL_i) \le s \le \min(UR_i)$.

Step 2. Refer to the subsequences to the left and right of s as SL_i and SR_i, respectively. Exchange the subsequences SR_i and UL_i by shifting $A[sr2]$, $A[sr2-1]$,..., $A[sr1]$ to $A[ul2], A[ul2-1], \ldots, A[ul2-(sr2-sr1)]$. This step means the following code:
for $i := 0$ **to** $sr2 - sr1$ **do** `Swap` $(sr2-i, ul2-i)$.

Step 3. After Step 2 is done, the two subproblems (SL_i, UL_i) and (SR_i, UR_i) have the same structure as the original(a sorted subsequence followed by an unsorted subsequence). Let the right subsequence contiguous to SE_i be UE_i. After the two subproblems are solved, the subproblem (SE_i, UE_i) has also the same structure as the original. Hence, the algorithm is called recursively to sort $(SL_i, UL_i), (SR_i, UR_i)$, and (SE_i, UE_i).

We implement the algorithm by way of a recursive routine. It is easy to replace this recursive routine by way of an iterative routine. Next we describe the main algorithm ProportionExtendSort with Pascal-like procedures.

ProportionExtendSort(S_i, UA_i) denotes that it sorts the two contiguous subsequences S_i and UA_i by using the parameters $s1, s2$, and $ua2$ in S_i and UA_i, where S_i is a sorted subsequence and UA_i is an unsorted subsequence. Notice, in this procedure, the parameter $ua1$ in UA_i is not used, since $ua1 = s2+1$. Initially $S_i = A[1..1]$, $UA_i = A[2..n]$.

```
ProportionExtendSort(S_i, UA_i)
      if s2 < s1 then s2 := s1                          { S_i = A[s1..s2] is sorted }
      if UA_i =empty then return                        { UA_i = A[ua1..ua2] is unsorted }
      se2 := s1 + (1 + p)|S_i| - 1                      { Extend S_i to SE_i = A[se1..se2] }
(2.1) if (1 + p)p|S_i| > |S_i| + |UA_i| then  se2 := ua2
      mid := ⌊(s1 + s2)/2⌋
      u1 := s2 + 1, u2 := se2                           { U_i = A[u1..u2] }
      ur1 := Partition(mid, U_i)                        { Split U_i into UL_i and UR_i }
(2.2) for i = 0 to s2 - mid do                          { shift A[s2..mid] to A[ur1-1...] }
            Swap (s2 - i, ur1 - 1 - i)
      sr1 := ur1 - (s2 - mid), sr2 := ur1 - 1           { SR_i = A[sr1..sr2] is sorted }
      ur2 := u2                                         { UR_i = A[ur1..ur2] is unsorted }
      sl1 := s1, sl2 := mid - 1                         { SL_i = A[sl1..sl2] is sorted }
      ul2 := sr1 - 2                                    { UL_i = A[ul1..ul2] is unsorted }
(2.3) ProportionExtendSort(SL_i, UL_i)
(2.4) ProportionExtendSort(SR_i, UR_i)
      se1 := s1
      ue2 := ua2                                        { UE_i = A[ue1..ue2] is unsorted }
      ProportionExtendSort (SE_i, UE_i)
end ProportionExtendSort
```

The routine Partition divides U_i into two subsequences and returns their boundary. The first subsequence consists of all $A[i] \le A[mid]$, and the second subsequence

consists of all $A[j] \geq A[mid]$. Partition is described as follows.

```
Partition (mid, U_i)
    { Suppose U_i = A[u1..u2] }
    i := u1, j := u2
    while i ≤ j do begin
      while i ≤ j and A[i] ≤ A[mid] do i := i + 1
      while i < j and A[j] ≥ A[mid] do j := j − 1
      if  i ≥ j then return i
      Swap(i, j)
      i := i + 1, j := j − 1
    end while
    return i
end Partition
```

The routine Partition first searches from left to right (by increasing i) until the element $> A[mid]$ is found, then searches from right to left (by decreasing j) until the element $< A[mid]$ is found. When both searches have paused, if $i < j$, then we exchange $A[i]$ and $A[j]$ and resume the process, otherwise, the algorithm terminates and returns i.

p in the algorithm is a fixed positive constant. Depending on p, we obtain various algorithms with different complexities.

Different algorithms can be obtained from this algorithm by replacing line (2.1). QUICKSORT is obtained by $se2 := ua2$, and BINARY INSERTIONSORT by $se2 := s2 + 1$. That is, QUICKSORT and BINARY INSERTIONSORT are two extreme cases of this algorithm.

ProportionExtendSort requires space for a pushdown stack which stores the arguments of pending recursive calls. We infer easily that maximal stack depth is $O(n)$. Maximal stack depth can be kept to $2 \log n$ if we make a slight modification of ProportionExtendSort: always solve the smaller subproblem first, i.e., replace lines (2.3) and (2.4) with

```
if  |UL_i| < |UR_i|
then ProportionExtendSort(SL_i, UL_i)
     ProportionExtendSort(SR_i, UR_i)
else ProportionExtendSort(SR_i, UR_i)
     ProportionExtendSort(SL_i, UL_i)
end if
```

3. Analysis and simulation results.

THEOREM 3.1. *Let $W(n)$ denote the worst case number of comparisons required by the algorithm for sorting n elements, then*

$$W(n) \leq 1/\log(1 + 1/(2p^2 + 2p - 1))n \log n, \text{for } p \geq 1.$$

Proof. In PROPORTION EXTEND SORT, comparisons occur only in the routine Partition. Partition(median of S_i, U_i) takes exactly $|U_i|$ comparisons to split U_i, where S_i and U_i denote the sorted set and the unsorted set upon entering the Partition for the ith time, respectively.

Let q denote the total number of the call to Partition, then we have

$$W(n) = \sum_{i=1}^{q} |U_i| = \sum_{i=1}^{q}\sum_{j=1}^{n} b(A[j] \in U_i) = \sum_{j=1}^{n}\sum_{i=1}^{q} b(A[j] \in U_i)$$
$$\leq n \times \max_{j=1}^{n}\left(\sum_{i=1}^{q} b(A[j] \in U_i)\right),$$

where $b(A[j] \in U_i)$ is one when $A[j]$ is in U_i, and zero otherwise.

Suppose $\max_{j=1}^{n}(\sum_{i=1}^{q} b(A[j] \in U_i)) = \sum_{i=1}^{q} b(y \in U_i) = k$. For the sake of simplicity, it will be assumed that y is in U_i for $1 \leq i \leq k$, and is not in U_t for $t > k$. By lines (2.1) in the algorithm, we have

$$|U_i| = \begin{cases} |UA_i| & \text{when } (p+1)p|S_i| > |S_i| + |UA_i|, \\ p|S_i| & \text{otherwise.} \end{cases} \tag{3.1}$$

$(p+1)p|S_i| > |S_i| + |UA_i|$ implies

$$|UA_i| < (p^2 + p - 1)|S_i| \qquad \text{for } p \geq 1. \tag{3.2}$$

By (3.1) and (3.2), we can imply

$$|U_i| \leq (p^2 + p - 1)|S_i| \qquad \text{for } p \geq 1.$$

Thus, we have

$$|U_i| + (p^2 + p - 1)|U_i| \leq (p^2 + p - 1)(|S_i| + |U_i|) \quad \text{for } p \geq 1.$$

That is,

$$|U_i| \leq \frac{(p^2 + p - 1)}{p^2 + p}(|S_i| + |U_i|) \qquad \text{for } p \geq 1. \tag{3.3}$$

By the relation of UA_{i+1} and U_{i+1}, we have

$$|U_{i+1}| \leq |UA_{i+1}|. \tag{3.4}$$

From lines (2.3) and (2.4) in the algorithm, we have

$$\begin{array}{l} S_{i+1} = SL_i \quad \text{and} \quad UA_{i+1} = UL_i, \text{ or} \\ S_{i+1} = SR_i \quad \text{and} \quad UA_{i+1} = UR_i. \end{array} \tag{3.5}$$

By the relation of UL_i, UR_i, and U_i, clearly

$$|UL_i| \leq |U_i| \quad \text{and} \quad |UR_i| \leq |U_i|. \tag{3.6}$$

Since SL_i and SR_i are the left and right half of S_i, we infer easily

$$|SL_i| \leq \frac{|S_i|}{2} \quad \text{and} \quad |SR_i| \leq \frac{|S_i|}{2}. \tag{3.7}$$

By (3.4)−(3.7) and (3.3), we obtain

$$\begin{aligned}
|S_{i+1}| + |U_{i+1}| &\le |S_{i+1}| + |UA_{i+1}| \\
&\le \max(|SL_i| + |UL_i|, |SR_i| + |UR_i|) \\
&\le \frac{|S_i|}{2} + |U_i| \\
&\le \frac{|S_i| + |U_i|}{2} + \frac{|U_i|}{2} \\
&\le \frac{|S_i| + |U_i|}{2} + \frac{(p^2+p-1)}{2(p^2+p)}(|S_i| + |U_i|) \\
&\le \frac{(2p^2+2p-1)}{2(p^2+p)}(|S_i| + |U_i|). \qquad (3.8)
\end{aligned}$$

Clearly, we have

$$|S_1| + |U_1| \le n \quad \text{and} \quad 2 \le |S_k| + |U_k|. \qquad (3.9)$$

by (3.8) and (3.9), we have

$$\begin{aligned}
2 &\le |S_k| + |U_k| \\
&\le \frac{(2p^2+2p-1)}{2(p^2+p)}(|S_{k-1}| + |U_{k-1}|) \\
&\le (\frac{(2p^2+2p-1)}{2(p^2+p)})^{k-1}(|S_1| + |U_1|) \\
&\le (\frac{(2p^2+2p-1)}{2(p^2+p)})^{k-1} \times n \qquad \text{for } p \ge 1.
\end{aligned}$$

We solve this inequality to obtain

$$k \le \frac{\log n}{\log(1 + 1/(2p^2+2p-1))}.$$

Hence,

$$W(n) \le k \times n \le \frac{n \log n}{\log(1 + 1/(2p^2+2p-1))} \quad \text{for } p \ge 1. \qquad \square$$

The worst case behavior of PROPORTION EXTEND SORT seems to occur on the completely sorted sequence as QUICKSORT. Even so, it takes only $O(n \log n)$ comparisons, and by empirical results, the actual number of comparisons required is far fewer than the formula given in the above theorem.

Since the "*ripple swap*"[8] technique can reduce the overall number of moves, the experimental results given below for PROPORTION EXTEND SORT are obtained by replacing line (2.2) with "*ripple swap*."

Table 3.1 summarizes experimental results on the average case performance we observed. For each input size n, we provided 20 distinct sets, each of which consists of randomly generated distinct values. The columns Compares and Moves reflect the average number of compares and exchanges for each instance, respectively. Columns EXSORT($p = 2$) and EXSORT($p = 6$) are the results of PROPORTION EXTEND SORT with $p = 2$ and $p = 6$, respectively. Columns SPSORT and QSORT are

TABLE 3.1
The average number of compares and moves.

Size	EXSORT(p=2)		EXSORT(p=6)		SPSORT		QSORT	
n	Compares	Moves	Compares	Moves	Compares	Moves	Compares	Moves
10	23	9	22	9	23	9	23	13
50	227	89	243	68	247	64	229	94
100	551	316	592	157	605	155	582	212
500	3895	2536	4154	1102	4201	1091	4202	1332
1000	8815	5925	9416	2452	9449	2474	9595	2899
5000	55538	25065	58835	15432	59053	15744	62240	17207
10000	121391	45971	127064	33536	127830	34432	135178	36874
50000	727550	314952	756987	196526	758823	208778	814311	209780

TABLE 3.2
The average-case results of PROPORTION SPLIT SORT when $n = 50000$.

p	COMPARES	MOVES	p	COMPARES	MOVES
3/2	717525	938807	8	743257	246753
7/4	719502	741052	16	758823	208778
2	720214	619809	24	767910	197504
3	725147	413725	32	775507	192418
4	730272	340538	64	793430	184692

the results reported for PROPORTION SPLIT SORT with $p = 16$ and CLEVER QUICKSORT, respectively, from [8].

Among the four algorithms given in Table 3.1, the average number of compares observed for EXSORT($p = 2$) is the smallest, and is less than the values of $n \log n - n$ consistently. SPSORT($p = 16$) is one of the best results of PROPORTION SPLIT SORT. Nevertheless, the average number of compares observed for EXSORT($p = 6$) is fewer than that observed for SPSORT($p = 16$); furthermore, when $n \geq 1000$, EXSORT($p = 6$) used fewer moves than SPSORT($p = 16$). For Table 3.1, exclusive of the number of compares with $n = 50$ and $n = 100$, data observed for EXSORT($p = 6$) are all lower than those observed for QSORT.

To compare PROPORTION EXTEND SORT with PROPORTION SPLIT SORT in further details, we present empirical results of these two sorting algorithms for some p in Tables 3.2 and 3.3, respectively. For each p, we employed 20 distinct sets, each of which consists of randomly generated distinct values as the simulation above. In Tables 3.2 and 3.3, Columns COMPARES and MOVES are the average number of comparisons and exchanges required to sort 50000 elements, respectively. Table 3.2 shows the results reported from [8].

As can be seen from Tables 3.2 and 3.3, the performance of PROPORTION EXTEND SORT is better than the performance of PROPORTION SPLIT SORT, since in the case of the same number of comparisons, in general, PROPORTION EXTEND SORT uses fewer moves than PROPORTION SPLIT SORT. Like PROPORTION SPLIT SORT, one can choose also the best version of PROPORTION EXTEND SORT by comparing the practical overall cost of comparisons and movements for each p based on the property of the input sequence given.

Table 3.4 shows the average execution time required by the algorithms running on Pentium II/350. All the algorithms are written in C and run under the MS-DOS operating system. Both QsortN and QsortR are the best-of-three versions of Quicksort, but when selecting three values to decide a pivot, QsortR adopts a random strategy while QsortN does not. SPSORT($p = 24$) and EXSORT($p = 16$) are the fastest

TABLE 3.3
The average-case results of PROPORTION EXTEND SORT when $n = 50000$.

p	COMPARES	MOVES	p	COMPARES	MOVES
1	717413	893189	7	757438	199929
2	727550	314952	8	762434	195925
3	732150	276479	9	768442	189230
4	741935	226547	10	778984	186906
5	746513	220086	12	777409	188825
6	756987	196526	16	786726	184675

TABLE 3.4
Average execution time to sort n integers (in milliseconds, Pentium II*/*350*MHz).*

n	SPSORT($p = 24$)	QsortN	QsortR	EXSORT($p = 16$)
50	0.233	0.261	0.346	0.227
100	0.556	0.610	0.783	0.549
500	3.775	4.083	4.963	3.734
1000	8.445	9.025	10.790	8.385
5000	50.40	55.80	64.60	50.20
10000	113.3	120.1	137.7	112.4
50000	674	703	794	658

versions of PROPORTION SPLIT SORT and PROPORTION EXTEND SORT, respectively, for sorting integers. From the empirical results given in Table 3.4, our algorithm is the fastest.

4. Comments. PROPORTION EXTEND SORT is a simple, competitive, and efficient sorting algorithm that sorts n elements, using $O(\log n)$ extra space and making $O(n \log n)$ comparisons in the worst case. Since an average case theoretic analysis seems to be quite sophisticated, it is left as an open problem. However, in our simulation, this sorting algorithm is better than PROPORTION SPLIT SORT, and used fewer data moves and fewer comparisons, and its number of comparisons is close to $\log(n!)$. Therefore, we believe that it is possible to replace CLEVER QUICKSORT by PROPORTION EXTEND SORT.

REFERENCES

[1] J.H. KINGSTON, *Algorithms and Data Structures: Design, Correctness, Analysis*, Addison-Wesley, Reading, MA, 1990, pp. 175–194.
[2] D.E. KNUTH, *The Art of Computer Programming,* Vol. 3*: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[3] R.W. FLOYD, *Algorithm* 245*, treesort* 3, Comm. ACM (1964), p. 701.
[4] C.A.R. HOARE, *Algorithm* 63*,* 64*, and* 65, Comm. ACM, 4 (1961), pp. 321–322.
[5] J.W.J. WILLIAM, *Algorithm* 232*, heapsort*, Comm. ACM, 7 (1964), pp. 347–348.
[6] R.D. DUTTON, *Weak-heap sort*, BIT, 33 (1993), pp. 372–381.
[7] I. WEGENER, *Bottom-up heap sort, a new variant of heap sort beating on average quicksort (if n is not very small)*, in Proceedings of Mathematical Foundations of Computer Science, Banska Bystrica, Czechoslovakia, 1990, pp. 516–522.
[8] J.C. CHEN, *Proportion split sort*, Nordic J. Comput., 3 (1996), pp. 271–279.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 331–352

APPROXIMATING THE THROUGHPUT OF MULTIPLE MACHINES IN REAL-TIME SCHEDULING*

AMOTZ BAR-NOY†, SUDIPTO GUHA†, JOSEPH (SEFFI) NAOR‡, AND BARUCH SCHIEBER§

Abstract. We consider the following fundamental scheduling problem. The input to the problem consists of n jobs and k machines. Each of the jobs is associated with a release time, a deadline, a weight, and a processing time on each of the machines. The goal is to find a nonpreemptive schedule that maximizes the weight of jobs that meet their respective deadlines. We give constant factor approximation algorithms for four variants of the problem, depending on the type of the machines (identical vs. unrelated) and the weight of the jobs (identical vs. arbitrary). All these variants are known to be NP-hard, and the two variants involving unrelated machines are also MAX-SNP hard. The specific results obtained are as follows:

- For identical job weights and unrelated machines: a greedy 2-approximation algorithm.
- For identical job weights and k identical machines: the same greedy algorithm achieves a tight $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor.
- For arbitrary job weights and a single machine: an LP formulation achieves a 2-approximation for polynomially bounded integral input and a 3-approximation for arbitrary input. For unrelated machines, the factors are 3 and 4, respectively.
- For arbitrary job weights and k identical machines: the LP-based algorithm applied repeatedly achieves a $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor for polynomially bounded integral input and a $\frac{(1+1/2k)^k}{(1+1/2k)^k-1}$ approximation factor for arbitrary input.
- For arbitrary job weights and unrelated machines: a combinatorial $(3+2\sqrt{2} \approx 5.828)$-approximation algorithm.

Key words. approximation algorithms, scheduling, real-time scheduling, multiple machines scheduling, parallel machines scheduling, throughput

AMS subject classifications. 68Q25, 90B35, 68W25, 68W40, 90B18

PII. S0097539799354138

1. Introduction.

We consider the following fundamental scheduling problem. The input to the problem consists of n jobs and k machines. Each of the jobs is associated with a release time, a deadline, a weight, and a processing time on each of the machines. The goal is to find a nonpreemptive schedule that maximizes the weight of the jobs that meet their deadline. Such scheduling problems are frequently referred

*Received by the editors April 14, 1999; accepted for publication (in revised form) November 10, 2000; published electronically August 8, 2001. An extended abstract of this paper appeared in the *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999. Part of this work was done while the first three authors visited IBM T.J. Watson Research Center.

http://www.siam.org/journals/sicomp/31-2/35413.html

†AT&T Shannon Lab, 180 Park Ave., P.O. Box 971, Florham Park, NJ 07932 (amotz@research.att.com, sudipto@research.att.com). The first author was on leave from the Electrical Engineering Department, Tel Aviv University, Tel Aviv 69978, Israel. Most of the work of the second author was done while the author was with the Computer Science Department, Stanford University, Stanford, CA 94305, and was supported by an IBM Cooperative fellowship, NSF grant IIS-9811904, and NSF award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡Computer Science Department, Technion, Haifa 32000, Israel (naor@cs.technion.ac.il). Most of this author's work was done while the author was with Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

§IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

to as *real-time* scheduling problems, and the objective of maximizing the value of completed jobs is frequently referred to as *throughput*. We consider four variants of the problem depending on the type of the machines (identical vs. unrelated) and the weight of the jobs (identical vs. arbitrary). In the standard notation for scheduling problems, the four problems we consider are $P|r_i|\sum(1-U_i)$, $P|r_i|\sum w_i(1-U_i)$, $R|r_i|\sum(1-U_i)$, and $R|r_i|\sum w_i(1-U_i)$.

Garey and Johnson [17] (see also [18]) show that the simplest decision problem corresponding to this problem is already NP-hard in the strong sense. In this decision problem the input consists of a set of n jobs with release time, deadline, and processing time. The goal is to decide whether all the jobs can be scheduled on a single machine, each within its time window. We show that the two variants involving unrelated machines are also MAX-SNP hard.

In this paper we give constant factor approximation algorithms for all four variants of the problem. To the best of our knowledge, this is the first paper that gives approximation algorithms with guaranteed performance (approximation factor) for these problems. We say that an algorithm has an approximation factor ρ for a maximization problem if the weight of its solution is at least $1/\rho \cdot \mathsf{OPT}$, where OPT is the weight of an optimal solution. (Note that we defined the approximation factor so that it would always be at least 1.)

The specific results obtained are listed below and summarized in Table 1.1.

- For identical job weights and unrelated machines, we give a greedy 2-approximation algorithm.
- For identical job weights and k identical machines, we show that the same greedy algorithm achieves a tight $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor.
- For arbitrary job weights, we round a fractional solution obtained from a linear programming relaxation of the problem. We distinguish between the case where the release times, deadlines, and processing times are integral and polynomially bounded, and the case where they are arbitrary. In the former case, we achieve a 2-approximation factor for a single machine and a 3-approximation factor for unrelated machines. In the latter case, we get a 3-approximation factor for a single machine and a 4-approximation factor for unrelated machines.
- For arbitrary job weights and k identical machines, we achieve a $\frac{(1+1/k)^k}{(1+1/k)^k-1}$ approximation factor for polynomially bounded integral input and a $\frac{(1+1/2k)^k}{(1+1/2k)^k-1}$ approximation factor for arbitrary input. Note that as k tends to infinity these factors tend to $\frac{e}{e-1} \approx 1.58198$ and $\frac{\sqrt{e}}{\sqrt{e}-1} \approx 2.54149$, respectively.
- For arbitrary job weights and unrelated machines we also present a *combinatorial* $(3+2\sqrt{2})$-approximation factor $(3+2\sqrt{2} \approx 5.828)$.

The computational difficulty of the problems considered here is due to the "slack time" available for scheduling the jobs. In general, the time window in which a job can be scheduled may be (much) larger than its processing time. Interestingly, the special case where there is no slack time can be solved optimally in polynomial time even for multiple machines [3] using dynamic programming. Moreover, the problem can be solved optimally on a single machine with the execution window strictly less than twice the length of the job, since it reduces to the case of no slack time.

Another special case that was considered earlier in the literature is the case in which all jobs are released at the same time (or equivalently, the case in which all deadlines are the same). This special case remains NP-hard even for a single machine.

TABLE 1.1

Each entry contains the approximation factors as a function of the number of machines (k) in the form $(1, 2, 3, \ldots, k, \ldots, k \to \infty)$.

Weight function	Identical machines	Unrelated machines
Identical job weights	$\left(2, 1.8, 1.73, \ldots, \frac{(1+1/k)^k}{(1+1/k)^k-1}, \ldots, 1.58\right)$	$(2, 2, 2, \ldots, 2)$
Arbitrary job weights Integral, poly-size, input	$\left(2, 1.8, 1.73, \ldots, \frac{(1+1/k)^k}{(1+1/k)^k-1}, \ldots, 1.58\right)$	$(2, 3, 3, \ldots, 3)$
Arbitrary job weights Arbitrary input	$\left(3, 2.78, 2.7, \ldots, \frac{(1+1/2k)^k}{(1+1/2k)^k-1}, \ldots, 2.54\right)$	$(3, 4, 4, \ldots, 4)$

However, Sahni [30] gave a fully polynomial approximation scheme for this special case.

The problems considered here have several applications. Hall and Magazine [21] considered the single machine version of our problem in the context of maximizing the scientific, military, or commercial value of a space mission. This means selecting and scheduling in advance a set of projects to be undertaken during the space mission, where an individual project is typically executable during only part of the mission. It is indicated in [21] that up to 25% of the budget of a space mission may be spent in making these decisions. Hall and Magazine [21] present eight heuristic procedures for finding a near optimal solution together with computational experiments. However, they do not provide any approximation guarantees on the solutions produced by their heuristics. They also mention the applicability of such problems to patient scheduling in hospitals. For more applications and related work in the scheduling literature see [11, 15] and the survey of [27].

The preemptive version of our problem for a single machine was studied by Lawler [26]. For identical job weights, Lawler showed how to apply dynamic programming techniques to solve the problem in polynomial time. He used the same techniques to obtain a pseudopolynomial algorithm for the NP-hard variant $1|r_i, pmtn|\sum w_i(1-U_i)$ in which the weights are arbitrary [26]. Lawler [25] also obtained polynomial time algorithms that solve the problem in two special cases: (i) the time windows in which jobs can be scheduled are nested, and (ii) the weights and processing times are in opposite order. Kise, Ibaraki, and Mine [23] showed how to solve the special case where the release times and deadlines are similarly ordered. For multiple machines, we note that $P|r_i, pmtn|\sum w_i(1-U_i)$ is NP-hard [27], yet there is a pseudopolynomial algorithm for this problem [26]. (However, it does not imply a fully polynomial approximation scheme [32].)

A closely related problem is considered by Adler et al. [1] in the context of communication in linear networks. In this problem, messages with release times and deadlines have to be transmitted over a bus that has a unit bandwidth, and the goal is to maximize the number of messages delivered within their deadline. It turns out that our approximation algorithms for the case of arbitrary weights can be applied to the weighted version of the unbuffered case considered in [1], yielding a constant factor approximation algorithm. No approximation algorithm is given in [1] for this version.

Spieksma [31] considered the *interval scheduling* problem on a single machine. In this problem, the possible instances of a job are given explicitly as a set of time intervals. The goal is to pick a set of maximum cardinality (or weight) of nonintersecting time intervals such that at most one interval from each set of job instances is picked. This problem can be viewed as the discrete version of our problem. Spieksma [31] considered the unweighted version of the interval scheduling problem. He proved that it is MAX-SNP hard, gave a 2-approximation algorithm which is similar to our greedy algorithm, and showed that the integrality gap of a linear programming formulation for this problem approaches 2 as well. We note that our results imply a 2-approximation algorithm for the weighted interval scheduling problem.

In the on-line version of our problems, the jobs appear one by one, and are not known in advance. Lipton and Tomkins [28] considered the nonpreemptive version of the on-line problem, while Koren and Shasha [24] and Baruah et al. [8] considered the preemptive version. The special cases where the weight of a job is proportional to the processing time were considered in the on-line setting in several papers [5, 14, 16, 19, 20, 7]. Our combinatorial algorithm for arbitrary weights borrows some of the techniques used in the on-line case.

Some of our algorithms are based on rounding a fractional solution obtained from a linear programming (LP) relaxation of the problem. In the LP formulation for a single machine we have a variable for every feasible schedule of each of the jobs, a constraint for each job, and a constraint for each time point. A naive implementation of this approach would require an unbounded number of variables and constraints. To overcome this difficulty, we first assume that all release times, deadlines, and processing times are (polynomially bounded) integers. This yields a polynomial number of variables and constraints, allowing for the LP to be solved in polynomial time. For the case of arbitrary input, we show that we need not consider more than $O(n^2)$ variables and constraints for each of the n jobs. This yields a strongly polynomial running time at the expense of a minor degradation in the approximation factor. The rounding of the fractional solution obtained from the LP relaxation is done by decomposing it into a convex sum of integral solutions, and then choosing the best one among them. We show that the bounds obtained by rounding a fractional solution are the best possible bounds that can be obtained, since they match the integrality gap of the LP relaxation.

We extend our algorithms from a single machine to multiple machines by applying a single machine algorithm repeatedly, machine-by-machine, and provide a rather general analysis for such a paradigm. Interestingly, it turns out that the approximation factor for the case of identical machines is superior to the approximation factor of the single machine algorithm which served as our starting point. A similar phenomenon (in a different context) has been observed by Cornuejols, Fisher, and Nemhauser [12]. In the unrelated machines case, our analysis is similar to the one described (in a different context) by Awerbuch et al. [4]. It is also similar to the $O(1)$-reduction described by Kalyasundaram and Pruhs [22] (in the preemptive case) from $P|r_i, pmtn| \sum w_i(1 - U_i)$ to $1|r_i, pmtn| \sum w_i(1 - U_i)$. Unlike the identical machines case, in the unrelated machines case the extension to multiple machines degrades the performance relative to a single machine.

2. Definitions and notations. Let the job system contain n jobs $\mathcal{J} = \langle J_1, \ldots, J_n \rangle$ and k machines $\mathcal{M} = \langle M_1, \ldots, M_k \rangle$. Each job J_i is characterized by the quadruple (r_i, d_i, L_i, w_i), where $L_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$. The interpretation is that job J_i is available at time r_i, the *release time*; it must be executed by time

d_i, the *deadline*; its *processing time* on machine M_j is $\ell_{i,j}$; and w_i is the *weight* (or *profit*) associated with the job. We note that our techniques can also be extended to the more general case where the release time, deadline, and weight of each job differ on different machines. However, for simplicity, we consider only the case where the release time, deadline, and weight of each job are the same on all machines. The hardness results are also proved under the same assumption.

We refer to the case in which all job weights are the same as the *unweighted* model, and the case in which job weights are arbitrary as the *weighted* model. (In the unweighted case our goal is to maximize the cardinality of the set of scheduled jobs.) We refer to the case in which the processing times of the jobs on all the machines are the same as the *identical machines* model, and the case in which processing times differ as the *unrelated machines* model. In the identical machines model with unweighted jobs, job J_i is characterized by a triplet (r_i, d_i, ℓ_i) where $d_i \geq r_i + \ell_i$. Without loss of generality, we assume that the earliest release time is at time $t = 0$.

A feasible scheduling of job J_i on machine M_j at time t, $r_i \leq t \leq d_i - \ell_{i,j}$, is referred to as a *job instance*, denoted by $J_{i,j}(t)$. A job instance can also be represented by an *interval* on the time line $[0, \infty)$. We say that the interval $J_{i,j}(t) = [t, t + \ell_{i,j})$ *belongs* to job J_i. In general, many intervals may belong to a job. A set of job instances $J_{1,j}(t_1), \ldots, J_{h,j}(t_h)$ is a feasible schedule on machine M_j, if the corresponding intervals are independent, i.e., they do not overlap, and they belong to distinct jobs. The *weight* of a schedule is the sum of the weights of the jobs to which the intervals (job instances) belong. In the case of multiple machines, we need to find a feasible schedule of distinct jobs on each of the machines. The objective is to maximize the sum of the weights of all schedules.

We distinguish between the case where the release times, processing times, and deadlines are integers bounded by a polynomial in the number of jobs, and between the case of arbitrary inputs. The former case is referred to as *polynomially bounded integral* input and the latter case is referred to as *arbitrary* input.

3. Unweighted jobs. In this section we consider the unweighted model. We define a greedy algorithm and analyze its performance in both the unrelated and identical models. In the former model, we show that it is a 2-approximation algorithm, and in the latter model, we show that it is a $\rho(k)$-approximation algorithm, where

$$\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k} = \frac{(1+1/k)^k}{(1+1/k)^k - 1} \ .$$

For $k = 1, 2$ we get $\rho(1) = 2$ and $\rho(2) = 9/5$, and for $k \longrightarrow \infty$ we have $\rho(k) \longrightarrow e/(e-1) \approx 1.58198$.

3.1. The greedy algorithm. The greedy strategy for a single machine is as follows. At each time step t (starting at $t = 0$), the algorithm schedules the job instance that finishes first among all jobs that can be scheduled at t or later. Note that the greedy algorithm does not take into consideration the deadlines of the jobs, except for determining whether jobs are eligible for scheduling. The greedy algorithm for multiple machines executes the greedy algorithm (for a single machine) machine by machine, updating the set of jobs to be scheduled on each machine to include only jobs that have not been scheduled on previous machines.

We now give a more formal definition of our strategy and introduce some notations. Define the procedure $\mathsf{NEXT}(t, j, \mathcal{J})$. The procedure determines the job instance $J_{i,j}(t')$, $t' \geq t$, such that $t' + \ell_{i,j}$ is the earliest among all instances of jobs in $\mathcal{J}$ that

start at time t or later on machine M_j. If no such interval exists, the procedure returns *null*.

Algorithm 1-GREEDY$(j, \mathcal{J})$ finds a feasible schedule on machine M_j among the jobs in $\mathcal{J}$ by calling procedure NEXT$(t, j, \mathcal{J})$, repeatedly.

1. The first call is for $J_{i_1,j}(t_1) = \mathsf{NEXT}(0, j, \mathcal{J})$.
2. Assume the algorithm has already computed $J_{i_1,j}(t_1), \ldots, J_{i_h,j}(t_h)$. Let the current time be $t = t_h + \ell_{i_h,j}$ and let the current set of jobs be $\mathcal{J} := \mathcal{J} \setminus \{J_{i_1,j}, \ldots, J_{i_h,j}\}$.
3. The algorithm calls NEXT$(t, j, \mathcal{J})$ that returns either $J_{i_{h+1},j}(t_{h+1})$ or *null*.
4. The algorithm terminates in round $r+1$ when procedure NEXT returns *null*. It returns the set $\{J_{i_1,j}(t_1), \ldots, J_{i_r,j}(t_r)\}$.

Algorithm k-GREEDY$(\mathcal{J})$ finds k schedules such that a job appears at most once in the schedules. It calls Algorithm 1-GREEDY machine by machine, each time updating the set $\mathcal{J}$ of jobs to be scheduled. Assume that the output of 1-GREEDY$(j, \mathcal{J})$ in the first $i-1$ calls is $G_1, \ldots, G_{i-1}$, where G_j is a feasible schedule on machine M_j, for $1 \le j \le i-1$. Then, the algorithm calls 1-GREEDY$(i, \mathcal{J} \setminus \cup_{j=1,\ldots,i-1} G_j)$ to get schedule G_i.

The following property of Algorithm 1-GREEDY is used in the analysis of the approximation factors of our algorithms.

PROPOSITION 3.1. *Let the set of jobs found by* 1-GREEDY$(j, \mathcal{J})$ *for a job system* $\mathcal{J}$ *be* G. *Let* H *be any feasible schedule on machine* M_j *among the jobs in* $\mathcal{J} \setminus G$. *Then,* $|H| \le |G|$.

Proof. For each interval (job instance) in H there exists an interval in G that overlaps with it and terminates earlier. Otherwise, 1-GREEDY would have chosen this interval. The proposition follows from the feasibility of H, since at most one interval in H can overlap with the endpoint of any interval in G. ∎

3.2. Unrelated machines. Based on Proposition 3.1, the following theorem states the performance of the k-GREEDY algorithm in the unweighted jobs and unrelated machines model.

THEOREM 3.2. *Algorithm* k-GREEDY *achieves an approximation factor of* 2 *in the unweighted jobs and unrelated machines model.*

Proof. Let $G(k) = G_1 \cup \cdots \cup G_k$ be the output of k-GREEDY and let $OPT(k) = O_1 \cup \cdots \cup O_k$ be the sets of intervals scheduled on the k machines by an optimal solution OPT. (We note that these sets will be considered as jobs and job instances interchangeably.) Let $H_j = O_j \setminus G(k)$ be the set of all jobs scheduled by OPT on machine M_j that k-GREEDY did not schedule on any machine, and let $H = H_1 \cup \cdots \cup H_k$. Let $OG = OPT(k) \cap G(k)$ be the set of jobs taken by both k-GREEDY and OPT. It follows that $OPT(k) = OG \cup H$.

Proposition 3.1 implies that $|H_j| \le |G_j|$. This is true since H_j is a feasible schedule on machine M_j among the jobs that were not picked by k-GREEDY while constructing the schedule for machine M_j. Since the sets H_j are mutually disjoint and the same holds for the sets G_j, $|H| \le |G(k)|$. Since $|OG| \le |G(k)|$, we get that $|OPT(k)| \le 2|G(k)|$ and the theorem follows. ∎

3.3. Identical machines. In this section we analyze the k-GREEDY algorithm for the unweighted jobs and identical machines model. We show that the approximation factor in this case is

$$\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k} \ .$$

Recall that for $k \longrightarrow \infty$ we have $\rho(k) \longrightarrow e/(e-1) \approx 1.58198$.

The analysis below is quite general and it only uses the fact that the algorithm is applied sequentially, machine by machine, and that the machines are identical (and that no job migration is allowed). Let $OPT(k)$ be an optimal schedule for k identical machines. Let $\mathcal{A}$ be any algorithm for one machine. Define by $\alpha_{\mathcal{A}}(k)$ (or by $\alpha(k)$ when $\mathcal{A}$ is known) the approximation factor of $\mathcal{A}$ compared with $OPT(k)$. That is, if A is the set of jobs chosen by $\mathcal{A}$, then $|A| \geq (1/\alpha(k))|OPT(k)|$. Note that the comparison is done between an algorithm that uses one machine and an optimal schedule that uses k machines.

Let $\mathcal{A}(k)$ be the algorithm that applies algorithm $\mathcal{A}$, machine by machine, k times. In the next theorem we bound the performance of $\mathcal{A}(k)$ using $\alpha(k)$.

THEOREM 3.3. *Algorithm $\mathcal{A}(k)$ achieves an $\frac{\alpha(k)^k}{\alpha(k)^k-(\alpha(k)-1)^k}$ approximation factor for k identical machines.*

Proof. Let A_i be the set of jobs chosen by $\mathcal{A}(k)$ for the ith machine. Suppose that the algorithm has already determined $A_1, \ldots, A_{i-1}$. Consider the schedule given by removing from $OPT(k)$ all the jobs in $A_1, \ldots, A_{i-1}$. Clearly, this is still a feasible schedule of cardinality at least $|OPT(k)| - \sum_{j=1}^{i-1} |A_j|$. Therefore, by the definition of $\alpha(k)$, the set A_i satisfies $|A_i| \geq (1/\alpha(k))(|OPT(k)| - \sum_{j=1}^{i-1} |A_j|)$. Adding $\sum_{j=1}^{i-1} |A_j|$ to both sides gives us

$$\sum_{j=1}^{i} |A_j| \geq \frac{|OPT(k)|}{\alpha(k)} + \frac{\alpha(k)-1}{\alpha(k)} \sum_{j=1}^{i-1} |A_j| \ . \tag{3.1}$$

We prove by induction on i that

$$\sum_{j=1}^{i} |A_j| \geq \frac{\alpha(k)^i - (\alpha(k)-1)^i}{\alpha(k)^i} |OPT(k)| \ .$$

When $i = 1$, by definition, $|A_1| \geq \frac{|OPT(k)|}{\alpha(k)}$. Assume the claim holds for $i-1$. Applying the induction hypothesis to (3.1) we get

$$\sum_{j=1}^{i} |A_j| \geq \frac{|OPT(k)|}{\alpha(k)} + \frac{\alpha(k)-1}{\alpha(k)} \cdot \frac{\alpha(k)^{i-1} - (\alpha(k)-1)^{i-1}}{\alpha(k)^{i-1}} |OPT(k)| \ .$$

Rearranging terms yields the inductive claim. Setting $i = k$ proves the theorem, namely,

$$\sum_{j=1}^{k} |A_j| \geq \frac{\alpha(k)^k - (\alpha(k)-1)^k}{\alpha(k)^k} |OPT(k)| \ . \qquad \square$$

We now apply the above theorem to Algorithm k-GREEDY. We compute the value of $\alpha(k)$ for Algorithm 1-GREEDY, and observe that Algorithm k-GREEDY indeed applies Algorithm 1-GREEDY k times, as assumed by Theorem 3.3.

THEOREM 3.4. *The approximation factor of k-GREEDY is $\rho(k) = \frac{(k+1)^k}{(k+1)^k-k^k}$ in the unweighted jobs and identical machines model.*

Proof. Recall that Algorithm 1-GREEDY scans all the intervals ordered by their endpoints and picks the first possible interval belonging to a job that was not picked before. Let G be the set picked by the greedy algorithm, and consider the schedule

of $H = OPT(k) - G$ on machines $M_1, \ldots, M_k$. Similar to the arguments of Proposition 3.1, in each of the machines, if a particular job of H was not chosen, then there must be a job in progress in G. Also this job must finish before the particular job in H finishes. Thus, the number of jobs in H executed on any single machine by the optimal schedule has to be at most $|G|$. Since the jobs executed by the optimal schedule on different machines are disjoint, we get $|H| \leq k|G|$. Consequently, $|OPT(k)| \leq (k+1)|G|$ and $\alpha(k) = k+1$. The theorem follows by setting this value for $\alpha(k)$ in Theorem 3.3. □

Remark. It is not difficult to see that $\alpha(k) \leq k\alpha(1)$; that is, if the approximation factor of $\mathcal{A}$ compared with $OPT(1)$ is $\alpha(1)$, then the approximation factor of $\mathcal{A}$ compared with $OPT(k)$ is no more than $k\alpha(1)$. However, applying Theorem 3.3 using this bound for $\alpha(k)$ would yield an approximation factor for k identical machines which is inferior to the $\alpha(1)$ bound on this approximation ratio that can be achieved directly. We note that for this reason Theorem 3.3 cannot be applied to improve the result in [22].

3.4. Tight bounds for GREEDY. In this subsection we construct instances for which our bounds in the unweighted model for algorithm GREEDY are tight. We first show that for one machine (where the unrelated and identical models coincide), the 2-approximation is tight. Next, we generalize this construction for the unrelated model and prove the tight bound of 2 for $k > 1$ machines. Finally, we generalize the construction for one machine to $k > 1$ identical machines and prove the tight bound of $\rho(k)$.

Recall that in the unweighted model each job is characterized by a triplet (r_i, d_i, ℓ_i) in the identical machines model and by a triplet (r_i, d_i, L_i), where $L_i = \{\ell_{i,1}, \ldots, \ell_{i,k}\}$, in the unrelated machines model.

3.4.1. A single machine. For a single machine the system contains two jobs: $G_1 = (0, 3, 1)$ and $H_1 = (0, 2, 2)$. Algorithm 1-GREEDY schedules the instance $G_1(0)$ of job G_1 and cannot schedule any instance of H_1. An optimal solution schedules the instances $H_1(0)$ and $G_1(2)$. Clearly, the ratio is 2. We could repeat this pattern on the time axis to obtain this ratio for any number of jobs.

This construction demonstrates the limitation of the approach of Algorithm 1-GREEDY. This approach ignores the deadlines and therefore does not capitalize on the urgency in scheduling job H_1 in order not to miss its deadline. We generalize this idea further for k machines.

Note that an algorithm that does not consider job lengths and schedules solely according to deadlines may produce even worse results. Consider the following $n+1$ jobs: n jobs $H_i = (0, 2n+1, 2)$, $1 \leq i \leq n$, and one job $G_1 = (0, 2n, 2n)$. An optimal solution schedules all the H-type jobs whereas an algorithm that schedules jobs with earliest deadline first schedules the G-type job first and then is unable to schedule any of the H-type jobs.

3.4.2. Unrelated machines. For $k \geq 1$ machines the job system contains $2k$ jobs: $G_1, \ldots, G_k$ and $H_1, \ldots, H_k$. The release time of all jobs is 0. The deadline of all the G-type jobs is 3 and the deadline of all the H-type jobs is 2. The length of job G_i on machine M_i is 1 and it is 4 on all other machines. The length of job H_i on machine M_i is 2 and it is 3 on all other machines.

Note that only jobs G_i and H_i can be scheduled on machine M_i, since all other jobs are too long to meet their deadline. Hence, Algorithm k-GREEDY considers only these two jobs while constructing the schedule for machine M_i. As a result,

k-GREEDY selects the instance $G_i(0)$ of job G_i to be scheduled on machine M_i and cannot schedule any of the H-type jobs. On the other hand, an optimal solution schedules the instances $H_i(0)$ and $G_i(2)$ on machine M_i.

Overall, Algorithm k-GREEDY schedules k jobs while an optimal algorithm schedules all $2k$ jobs. This yields a tight approximation factor of 2 in the unweighted jobs and unrelated machines model.

3.4.3. Identical machines. We define job systems $\mathcal{J}(k)$ for any given $k \geq 1$. We show that on $\mathcal{J}(k)$ the performance of k-GREEDY$(\mathcal{J}(k))$ is no more than $(1/\rho(k)) \cdot \mathsf{OPT}(\mathcal{J}(k))$. The $\mathcal{J}(1)$ system is the one defined in subsection 3.4.1. The $\mathcal{J}(2)$ job system contains $2 \cdot 3^2 = 18$ jobs: $2 \cdot 3 = 6$ jobs of type $G_1 = (0, d_1, \ell)$, $2^2 = 4$ jobs of type $G_2 = (0, d_2, \ell + 1)$, and $2^3 = 8$ jobs of type $H = (0, d, \ell + 2)$. If we set $\ell = 10$, $d_1 = 100$, $d_2 = 70$, and $d = 48$, we force Algorithm 2-GREEDY to make the following selections:

- On the first machine, 2-GREEDY schedules all the 6 jobs of type G_1. This is true since the length of these jobs is less than the lengths of the jobs of type G_2 and the jobs of type H. The last G_1-type interval terminates at time 60. Hence, there is no room for a G_2-type (H-type) interval, the deadline of which is 70 (48), and the length of which is 11 (12).
- On the second machine, 2-GREEDY schedules all 4 jobs of type G_2 since they are shorter than the jobs of type H. The last G_2-type job terminates at time 44 which leaves no room for another job of type H.

Overall, 2-GREEDY schedules only 10 jobs. We show now an optimal solution that schedules all 18 jobs. It schedules 9 jobs on each machine as follows:

$$H(0), H(12), H(24), H(36), G_2(48), G_2(59), G_1(70), G_1(80), G_1(90) \ .$$

Note that all the instances terminate before their deadlines. As a result we get a ratio $\rho(2) = (2 \cdot 3^2)/(2 \cdot 3^2 - 2^3) = 9/5$.

We are ready to define $\mathcal{J}(k)$ for any $k \geq 1$. The job system contains $k(k+1)^k$ jobs. Algorithm k-GREEDY is able to schedule only $k(k+1)^k - k^{k+1}$ out of them and there exists an optimal solution that schedules all of them. As a result we get the ratio

$$\frac{k(k+1)^k}{k(k+1)^k - k^{k+1}} = \rho(k) \ .$$

The $\mathcal{J}(k)$ system is composed of $k+1$ types of jobs: $G_1, G_2, \ldots, G_k$ and H. There are $k^i(k+1)^{k-i}$ jobs $(0, d_i, \ell + i - 1)$ in G_i and k^{k+1} jobs $(0, d, \ell + k)$ in H. Indeed, $k^{k+1} + \sum_{i=1}^{k} k^i(k+1)^{k-i} = k(k+1)^k$. (To see this, divide the equation by k^{k+1} to get $1 + (1/k) \cdot \sum_{i=0}^{k-1}(1+1/k)^i = (1+1/k)^k$.) Note also that the length of the G_i-type jobs is monotonically increasing in i and that the H-type jobs are the longest.

We show how by fixing $d_1, \ldots, d_k$ and d and by setting ℓ as a large enough number, we force Algorithm k-GREEDY to select for machine i all the jobs of type G_i but no other jobs. Thus, k-GREEDY does not schedule any of the H-type jobs. On the other hand, an optimal solution is able to construct the same schedule for all the k machines. It starts by scheduling $1/k$ of the H-type jobs on each machine, and then it schedules in turn $1/k$ of the jobs from $G_k, G_{k-1}, \ldots, G_1$ in this order.

We fix the values of $d, d_k, \ldots, d_1$ to allow for such an optimal schedule. The optimal solution starts by scheduling on each machine $(1/k) \cdot k^{k+1} = k^k$ of the H-type jobs each of length $\ell + k$. Thus, we set $d = k^k \cdot (\ell + k) = k^k\ell + k^{k+1}$. Next,

$(1/k) \cdot k^k = k^{k-1}$ of the G_k-type jobs each of length $\ell + k - 1$ are scheduled. Thus, we set $d_k = d + k^{k-1} \cdot (\ell + k - 1)$. Similarly, for $i = k-1, \ldots, 1$,

$$\begin{aligned} d_i &= d_{i+1} + (1/k) \cdot k^i (1+k)^{k-i} \cdot (\ell + i - 1) \\ &= d_{i+1} + k^{i-1}(1+k)^{k-i} \cdot (\ell + i - 1) \\ &= d_{i+1} + k^{i-1}(1+k)^{k-i} \cdot \ell + k^{i-1}(1+k)^{k-i} \cdot (i-1) \ . \end{aligned}$$

Observe that $d < d_k < \cdots < d_1$.

We now have to show that with the values fixed above Algorithm k-GREEDY schedules all the jobs of type G_i but no other jobs on machine i. Note that we have not set yet the value of ℓ; we set it to be large enough to force such a behavior of k-GREEDY. First, we find the general solution to the recurrence for $d_{(k+1)-i}$, $1 \le i \le k$. The coefficient of ℓ in $d_{(k+1)-i}$ is

$$\sum_{j=1}^{i} k^{k-j}(k+1)^{j-1} + k^k = k^{k-i}(k+1)^i \ .$$

It follows that

$$d_{k+1-i} = k^{k-i}(k+1)^i \ell + k^{k+1} + \sum_{j=1}^{i} k^{k-j}(k+1)^{j-1}(k-j) \ .$$

For the analysis below we need to break the expression for d_i into two components: one is the term that depends (linearly) on ℓ, and the other that depends only on i and k. For convenience denote $d_{k+1} = d$. It follows that for $1 \le i \le k+1$,

$$d_i = k^{i-1}(k+1)^{k-i+1}\ell + f(i,k)$$

for some function $f(i,k)$ independent of ℓ.

Algorithm k-GREEDY starts by scheduling all the jobs of type G_1 on machine 1, since these are the shortest length jobs. The time taken is $k(k+1)^{k-1}\ell$. Since our goal is not to have any other jobs scheduled on machine 1, we must make the deadline of the other jobs early enough so that they cannot be scheduled after this time. In particular, to prevent scheduling G_2-type jobs on machine 1 we must have $k(k+1)^{k-1}\ell + (\ell + 1) > d_2$. Note that since the deadlines d_i are monotonically decreasing and the lengths of the G_i-type jobs are monotonically increasing, if a job of type G_i cannot be scheduled, then a job of type G_{i+1} cannot be scheduled as well. The same is true for jobs of type H. It follows that if $k(k+1)^{k-1}\ell + (\ell + 1) > d_2$, then after all the jobs of type G_1 are scheduled on machine 1, no other jobs can be scheduled on machine 1.

In general, assume that Algorithm k-GREEDY starts the scheduling on machine i after all jobs of type $G_1, \ldots, G_{i-1}$ have already been scheduled on machines $1, \ldots, i-1$, respectively. In this case Algorithm k-GREEDY schedules all the jobs of type G_i on machine i, since these are the shortest length jobs that have not been scheduled yet. The time taken is $k^i(k+1)^{k-i}(\ell + i - 1)$. Thus, if

$$k^i(k+1)^{k-i}(\ell + i - 1) + (\ell + i) > d_{i+1},$$

then the jobs of type G_{i+1} cannot be scheduled on machine i. This implies that also

all jobs of types $G_{i+2}, \ldots, G_k$ and H cannot be scheduled at this point on machine i.

We conclude that in order to force the desired behavior of Algorithm k-GREEDY we must have for all $1 \leq i \leq k$

$$k^i(k+1)^{k-i}(\ell+i-1)+(\ell+i) > d_{i+1} \ .$$

By the recurrence relation for d_{i+1} it follows that the above inequality holds if and only if the following holds:

$$k^i(k+1)^{k-i}(\ell+i-1)+(\ell+i) > k^i(k+1)^{k-i}\ell + f(i+1,k) \ .$$

This is equivalent to

$$\ell > f(i+1,k) - i - (i-1)k^i(k+1)^{k-i} \ .$$

Since $f(i+1,k)$ does not depend on ℓ, it follows that the above inequality holds for a sufficiently large ℓ. This provides a tight bound for the k-GREEDY schedule.

4. Weighted jobs. In this section we present approximation algorithms for weighted jobs. We first present algorithms for a single machine and for unrelated machines that are based on rounding an LP relaxation of the problem. Then, we reapply the analysis of Theorem 3.3 to get better approximation factors for the identical machines model. We conclude with a combinatorial algorithm for unrelated machines which is efficient and easy to implement. However, it achieves a weaker approximation guarantee than the bound obtained by rounding a fractional solution obtained from the LP relaxation.

4.1. Approximation via linear programming. In this subsection we describe an LP based approximation algorithm. We first describe the algorithm for the case of a single machine, and then generalize it to the case of multiple machines. Our LP formulation is based on discretizing time. Suppose that the time axis is divided into N time slots. The complexity of our algorithms depends on N. However, we assume for now that N is part of the input, and that the discretization of time is fine enough so as to represent any feasible schedule, up to small shifts that do not change the value of the objective function. Later, we show how to get rid of these assumptions at the expense of a slight increase in the approximation factor.

4.1.1. A single machine. In this subsection we describe our linear program assuming that the number of slots N is part of the input.

The linear program relaxes the scheduling problem in the following way. A fractional feasible solution is one which distributes the processing of a job among the job instances or intervals belonging to it with the restriction that at any given point of time t, the sum of the fractions assigned to all the intervals at t (belonging to all jobs) does not exceed 1. To this end, for each job $J_i \in \mathcal{J}$, define a variable x_{it} for each interval $[t, t+\ell_i)$ belonging to it, i.e., for which $t \geq r_i$ and $t+\ell_i \leq d_i$. It would be convenient to assume that $x_{it} = 0$ for any other value of t between 1 and N. The linear program is as follows.

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{t=r_i}^{d_i-\ell_i} w_i \cdot x_{it}$$

subject to

$$\begin{aligned}
&\text{for each time slot } t,\ 1 \le t \le N, && \sum_{i=1}^{n} \sum_{t'=t-\ell_i+1}^{t} x_{it'} \le 1,\\
&\text{for each job } i,\ 1 \le i \le n, && \sum_{t=r_i}^{d_i-\ell_i} x_{it} \le 1,\\
&\text{for all } 1 \le i \le n \text{ and } 1 \le t \le N, && 0 \le x_{it} \le 1.
\end{aligned}$$

It is easy to see that any feasible schedule defines a feasible integral solution to the linear program and vice versa. Therefore, the value of an optimal (fractional) solution to the linear program is an upper bound on the value of an optimal integral solution.

We compute an optimal solution to the linear program and denote the value of variable x_{it} in this solution by q_{it}. Denote the value of the objective function in an optimal solution by OPT. We now show how to round an optimal solution to the linear program to an integral solution.

We define the following coloring of intervals. The collection of all intervals belonging to a set of jobs $\mathcal{J}$ can be regarded as an interval representation of an interval graph $\mathcal{I}$. We define a set of intervals in $\mathcal{I}$ to be *independent* if (i) no two intervals in the set overlap and (ii) no two intervals in the set belong to the same job. (Note that this definition is more restrictive than the regular independence relation in interval graphs.) Clearly, an independent set of intervals defines a feasible schedule. The weight of an independent set P, $w(P)$, is defined to be the sum of the weights of the jobs to which the intervals belong.

Our goal is to color intervals in $\mathcal{I}$ such that each color class induces an independent set. We note that not all intervals are required to be colored and that an interval may receive more than one color. Suppose that a collection of color classes (independent sets) $P_1, \ldots, P_m$ with nonnegative coefficients $\alpha_1, \ldots, \alpha_m$ satisfies (i) $\sum_{i=1}^{m} \alpha_i \le 2$ and (ii) $\sum_{i=1}^{m} w(P_i) \cdot \alpha_i \ge \mathsf{OPT}$. By convexity, there exists a color class P_i, $1 \le i \le m$, for which $w(P_i) \ge \mathsf{OPT}/2$. This color class is defined to be our approximate solution, and the approximation factor is 2. It remains to show how to obtain the desired coloring.

We now take a short detour and define the *group constrained interval coloring problem*. Let $Q = \langle Q_1, \ldots, Q_p \rangle$ be an interval representation in which the maximum number of mutually overlapping intervals is t_1. Suppose that the intervals are partitioned into disjoint groups $g_1, \ldots, g_r$, where a group contains at most t_2 intervals. A legal group constrained coloring of the intervals in Q is a coloring in which (i) overlapping intervals are not allowed to get the same color and (ii) intervals belonging to the same group are not allowed to get the same color.

THEOREM 4.1. *There exists a legal group constrained coloring of the intervals in Q that uses at most $t_1 + t_2 - 1$ colors.*

Proof. We use a greedy algorithm to obtain a legal coloring using at most t_1+t_2-1 colors. Sort the intervals in Q by their left endpoint and color the intervals from left to right with respect to this ordering. When an interval is considered by the algorithm it is colored by any one of the free colors available at that time. We show by induction that when the algorithm considers an interval, there is always a free color.

This is true initially. When the algorithm considers interval Q_i, the colors that cannot be used for Q_i are occupied either by intervals that overlap with Q_i or by intervals that belong to the same group as Q_i. Since we are considering the intervals sorted by their left endpoint, all intervals overlapping with Q_i also overlap with each other, and hence there are at most $t_1 - 1$ such intervals. There can be at most $t_2 - 1$ intervals that belong to the same group as Q_i. Since the number of available colors is $t_1 + t_2 - 1$, there is always a free color. □

We are now back to the problem of coloring the intervals in $\mathcal{I}$. Let $N' = N^2 \cdot n^2$. We can round down each fraction q_{it} in the optimal solution to the closest fraction of the form a/N', where $1 \leq a \leq N'$. Each q_{it} decreases by at most $1/(N^2n^2)$ by this rounding, and hence the rounding decreases the objective function by at most $\max_i w_i/(Nn)$. Since the optimal solution value is at least $\max_i w_i$, it follows that the rounding incurs a negligible error (of at most $1/(Nn)$ factor) in the value of the objective function. We now generate an interval graph $\mathcal{I}'$ from $\mathcal{I}$ by replacing each interval $J_i(t) \in \mathcal{I}$ by $q_{it} \cdot N'$ "parallel" intervals. Define a group constrained coloring problem on $\mathcal{I}'$, where group g_i, $1 \leq i \leq n$, contains all instances of job J_i. Note that in $\mathcal{I}'$, the maximum number of mutually overlapping intervals is bounded by N', and the maximum number of intervals belonging to a group is also N'.

By Theorem 4.1, there exists a group constrained coloring of $\mathcal{I}'$ that uses at most $2N' - 1$ colors. Attach a coefficient of $1/N'$ to each color class. Clearly, the sum of the coefficients is less than 2. Also, by our construction, the sum of the weights of the intervals in all the color classes, multiplied by the coefficient $1/N'$, is at least OPT (up to a factor of $1 - 1/(Nn)$, due to the rounding). We conclude with the following theorem.

THEOREM 4.2. *The approximation factor of the algorithm that rounds an optimal fractional solution is* 2.

We note that the technique of rounding a fractional solution by decomposing it into a convex combination of integral solutions was used in [2, 10].

The 2-approximation factor obtained by the rounding algorithm is the best possible bound that can be obtained through our linear programming relaxation. The following example shows that the integrality gap between the fractional LP solution and the integral solution can approach 2. There are two jobs: $G_1 = (0, d, 1)$ and $H_1 = (0, d, d)$, both of unit weight. Any integral solution can process only a single job. A fractional solution may assign a fraction of $1 - 1/d$ to job H_1, and then assign a fraction of $1/d$ to all job instances of G_1 of the form $[i, i+1)$, $0 \leq i \leq d-1$. Thus, the integrality gap is $2 - 1/d$. Note that the integrality gap depends on the discretization of time and it approaches 2 as d goes to infinity.

4.1.2. A strongly polynomial bound for a single machine. The difficulty with the linear programming formulation and the rounding algorithm is that the complexity of the algorithm depends on N, the number of time slots. We now show how we choose N to be a polynomial in the number of jobs, n, at the expense of losing an additive term of one in the approximation factor.

First, we note that in case the release times, deadlines, and processing times are integral, we may assume without loss of generality that each job is scheduled at an integral point of time. If, in addition, they are restricted to integers of polynomial size, then the number of variables and constraints is bounded by a polynomial.

We now turn our attention to the case of arbitrary inputs. Partition the jobs in $\mathcal{J}$ into two classes:

- big slack jobs: $J_i \in \mathcal{J}$ for which $d_i - r_i \geq n^2 \cdot \ell_i$,

- small slack jobs: $J_i \in \mathcal{J}$ for which $d_i - r_i < n^2 \cdot \ell_i$.

We obtain a fractional solution separately for the big slack jobs and small slack jobs. We first explain how to obtain a fractional solution for the big slack jobs. For each big slack job $J_i \in \mathcal{J}$, find n^2 nonoverlapping job instances and assign a value of $1/n^2$ to each such interval. Note that this many nonoverlapping intervals can be found since $d_i - r_i$ is large enough. We claim that this assignment can be ignored when computing the solution (via LP) for the small slack jobs. This is true because at any point of time t, the sum of the fractions assigned to intervals at t belonging to big slack jobs can be at most $1/n$, and thus their effect on any fractional solution is negligible. (In the worst case, scale down all fractions corresponding to small slack jobs by a factor of $(1-1/n)$.) Nevertheless, a big slack job contributes all of its weight to the fractional objective function because it has n^2 nonoverlapping copies.

We now restrict our attention to the set of small slack jobs and explain how to compute a fractional solution for them. To bound the number of variables and constraints in the LP we partition the time axis into at most $n \cdot (n^2 + 1)$ time slots. Instead of having a variable for each job instance we consider at most $n^2 \cdot (n^2 + 1)$ variables, where for each job $J_i \in \mathcal{J}$, there are at most $n \cdot (n^2 + 1)$ variables, and the jth variable "represents" all the job instances of J_i that start during the jth time slot. Similarly, we consider at most $n \cdot (n^2 + 1)$ constraints, where the jth constraint "covers" the jth time slot. For each small slack job $J_i \in \mathcal{J}$, define $n^2 + 1$ "dividers" along the time axis at points $r_i + j\frac{d_i - r_i}{n^2}$ for $j = 0, \ldots, n^2$. After defining all the $n \cdot (n^2+1)$ dividers, the time slots are determined by the adjacent dividers. The main observation is that for each small slack job J_i, no interval can be fully contained in a time slot, i.e., between two consecutive dividers.

The LP formulation for the modified variables and constraints is slightly different from the original formulation. To see why, consider a feasible schedule. As mentioned above, a job instance cannot be fully contained in a time slot t. However, the schedule we are considering may consist of two instances of jobs such that one terminates within time slot t and the other starts within t. If we keep the constraints that stipulate that the sum of the variables corresponding to intervals that intersect a time slot is bounded by 1, then we would not be able to represent such a schedule in our formulation. To overcome this problem, we relax the linear program and allow that at every time slot t, the sum of the fractions assigned to the intervals that intersect t can be at most 2. The relaxed linear program is the following.

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{r_i \le t \le d_i - \ell_i} w_i \cdot x_{it}$$

subject to

$$\begin{aligned}
&\text{for each time slot } t, && \sum_{i=1}^{n} \sum_{t - \ell_i + 1 \le t' \le t} x_{it'} \le 2,\\
&\text{for each job } i,\ 1 \le i \le n, && \sum_{r_i \le t \le d_i - \ell_i} x_{it} \le 1,\\
&\text{for all } i \text{ and } t, && 0 \le x_{it} \le 1.
\end{aligned}$$

It is easy to see that our relaxation guarantees that the value of the objective function in the above linear program is at least as big as the value of an optimal

schedule. We round an optimal fractional solution in the same way as in the previous section. Since we relaxed our constraints, we note that when we run the group constrained interval coloring algorithm, the number of mutually overlapping intervals can be at most twice the number of intervals in each group. Therefore, when we generate the color classes $P_1, \ldots, P_m$ with coefficients $\alpha_1, \ldots, \alpha_m$, we can guarantee only that (i) $\sum_{i=1}^{m} \alpha_i < 3$ and (ii) $\sum_{i=1}^{m} w(P_i) \cdot \alpha_i = \mathsf{OPT}$, yielding an approximation factor of 3. We conclude the following.

THEOREM 4.3. *The approximation factor of the strongly polynomial algorithm that rounds a fractional solution is* 3.

4.1.3. Unrelated machines. In this section we consider the case of k unrelated machines. We first present an LP formulation. For clarity, we give the LP formulation for polynomially bounded integral inputs. However, the construction given in the previous section that achieves a strongly polynomial algorithm for arbitrary inputs can be applied here as well. Assume that there are N time slots. For each job $J_i \in \mathcal{J}$ and for each machine $M_j \in \mathcal{M}$, define a variable x_{itj} for each instance $[t, t + \ell_{i,j}]$ of J_i.

$$\text{maximize} \quad \sum_{j=1}^{k} \sum_{i=1}^{n} \sum_{t=r_i}^{d_i - \ell_{i,j}} w_i \cdot x_{itj}$$

subject to

$$\begin{aligned}
&\text{for each time slot } t,\ 1 \le t \le N,\\
&\quad \text{and machine } j,\ 1 \le j \le k, && \sum_{i=1}^{n} \sum_{t'=t-\ell_{i,j}+1}^{t} x_{it'j} \le 1,\\
&\text{for each job } i,\ 1 \le i \le n, && \sum_{j=1}^{k} \sum_{t=r_i}^{d_i-\ell_{i,j}} x_{itj} \le 1,\\
&\text{for all } 1 \le i \le n,\ 1 \le j \le k,\\
&\quad \text{and } 1 \le t \le N, && 0 \le x_{itj} \le 1.
\end{aligned}$$

The algorithm rounds the fractional solution machine by machine. Let $S = \{S_1, \ldots, S_k\}$ denote the rounded solution. When rounding machine i, we first discard from its fractional solution all intervals belonging to jobs assigned to $S_1, \ldots, S_{i-1}$. Let c denote the approximation factor that can be achieved when rounding a single machine. Namely, $c = 2$ for integral polynomial size inputs and $c = 3$ for arbitrary inputs.

THEOREM 4.4. *The approximation factor of the algorithm that rounds a k-machine solution is $(c + 1)$.*

Proof. Let F_j, $1 \le j \le k$, denote the fractional solution of machine j, and let $w(F_j)$ denote its value. Denote by F'_j the fractional solution of machine j after discarding all intervals belonging to jobs chosen to $S_1, \ldots, S_{j-1}$.

We know that for all j, $1 \le j \le k$,

$$w(S_j) \ge \frac{1}{c} \cdot w(F'_j) \ .$$

Adding up all the inequalities, since the sets S_j (F'_j) are mutually disjoint, we get that

$$w(S) \geq \frac{1}{c} \cdot \sum_{j=1}^{k} w(F'_j).$$

Recall that for each job i, the sum of the values of the fractional solution assigned to the intervals belonging to it in all the machines does not exceed 1. Therefore,

$$\sum_{j=1}^{k} w(F'_j) \geq \sum_{j=1}^{k} w(F_j) - w(S) \ .$$

Yielding that

$$w(S) \geq \frac{\sum_{j=1}^{k} w(F_j)}{c+1}. \qquad \square$$

4.1.4. Identical machines. In this subsection we apply Theorem 3.3 for the case of weighted jobs and identical machines. We distinguish between the cases of polynomially bounded integral input and arbitrary input.

THEOREM 4.5. *There exists an algorithm for the weighted jobs and identical machines case that achieves an approximation factor of* $\rho(k) = \frac{(k+1)^k}{(k+1)^k - k^k}$ *for polynomially bounded integral input and* $\rho'(k) = \frac{(2k+1)^k}{(2k+1)^k - (2k)^k}$ *for arbitrary input.*

Proof. As shown above, a linear program can be formulated such that the value of its optimal solution is at least as big as the value of an optimal schedule. Let N' be chosen in the same way as in the discussion preceding Theorem 4.2. We claim that using our rounding scheme, this feasible solution defines an interval graph that can be colored by $(k+1)N' - 1$ colors for integral polynomial size inputs and by $(2k+1)N' - 1$ colors for arbitrary inputs.

Consider first the case of integral polynomial size input. In the interval graph that is induced by the solution of the LP, there are at most N' intervals (that correspond to the same job) in the same group, and at most kN' intervals mutually overlap at any point of time. Applying our group constrained interval coloring, we get a valid coloring with $(k+1)N' - 1$ colors. Similarly, for arbitrary inputs, in the interval graph which is induced by the solution of the LP, there are at most N' intervals (that correspond to the same job) in the same group, and at most $2kN'$ intervals mutually overlap. Applying our group constrained interval coloring, we get a valid coloring with $(2k+1)N' - 1$ colors.

This implies that $\alpha(k) = k+1$ for integral polynomial size input and $\alpha(k) = 2k+1$ for arbitrary input. In other words, this is the approximation factor that can be achieved with a single machine when compared to an optimal algorithm that uses k identical machines. Setting these values of $\alpha(k)$ in our paradigm for transforming an algorithm for a single machine to an algorithm for k identical machines yields the claimed approximation factors. $\square$

Remark. Note that as k tends to infinity, the approximation factor is $\frac{e}{e-1} \approx 1.58192$ for both unweighted jobs and for weighted jobs with integral polynomial size inputs. For arbitrary input, the approximation factor is $\frac{\sqrt{e}}{\sqrt{e}-1} \approx 2.54149$. Setting $k = 1$ we get that these bounds coincide with the bounds for a single machine. For every $k \geq 2$ and for both cases these bounds improve upon the bounds for unrelated machines (of 3 and 4).

4.2. A combinatorial algorithm. In this section we present a combinatorial algorithm for the weighted jobs model. We first present an algorithm for the single-machine version and then we show how to extend it to the case where there are $k > 1$ machines, even in the unrelated machines model.

4.2.1. A single machine. The algorithm is inspired by on-line call admission algorithms (see [16, 8]). We scan the job instances (or intervals) one by one. For each job instance, we either accept it or reject it. We note that rejection is an irrevocable decision, where as acceptance can be temporary, i.e., an accepted job may still be rejected at a later point of time. We remark that in the case of nonpreemptive on-line call admission, a constant competitive factor cannot be achieved by such an algorithm. The reason is that due to the on-line nature of the problem jobs must be considered in the order of their release time. Our algorithm has the freedom to order the jobs in a different way, yielding a constant approximation factor.

We now outline the algorithm. All feasible intervals of all jobs are scanned from left to right (on the time axis) sorted by their endpoints. The algorithm maintains a set $\mathcal{A}$ of currently accepted intervals. When a new interval, I, is considered according to the sorted order, it is immediately rejected if it belongs to a job that already has an instance in $\mathcal{A}$, and immediately accepted if it does not overlap with any other interval in $\mathcal{A}$. In case of acceptance, interval I is added to $\mathcal{A}$. If I overlaps with one or more intervals in $\mathcal{A}$, it is accepted only if its weight is more than β ($\beta > 1$, to be determined later) times the sum of the weights of all overlapping intervals. In this case, we say that I "preempts" these overlapping intervals. We add I to $\mathcal{A}$ and discard all the overlapping intervals from $\mathcal{A}$. The process ends when there are no more intervals to scan.

A more formal description of our algorithm, called Algorithm ADMISSION, is given in Figure 4.1.

Algorithm ADMISSION**:**

1. Let $\mathcal{A}$ be the set of accepted job instances.
   Initially, $\mathcal{A} = \emptyset$.
2. Let $\mathcal{I}$ be the set of the yet unprocessed job instances.
   Initially, $\mathcal{I}$ is the set of all feasible job instances.
3. While $\mathcal{I}$ is not empty repeat the following procedure:
   Let $I \in J_i$ be the job instance that terminates earliest among all instances in $\mathcal{I}$ and let w be its weight.
   Let W be the sum of the weights of all instances $I_1, \ldots, I_h$ in $\mathcal{A}$ that overlap I.
   (a) $\mathcal{I} := \mathcal{I} \setminus \{I\}$.
   (b) If $J_i \cap \mathcal{A} \neq \emptyset$ then **reject** I.
   (c) Else if $W = 0$ then **accept** I;
   $\mathcal{A} := \mathcal{A} \cup \{I\}$.
   (d) Else if $\frac{w}{W} > \beta$ then **accept** I and **preempt** $I_1, \ldots, I_h$;
   $\mathcal{A} := \mathcal{A} \cup \{I\} \setminus \{I_1, \ldots, I_h\}$.
   (e) Otherwise (i.e., $\frac{w}{W} \leq \beta$) then **reject** I.

end ADMISSION

FIG. 4.1. *Algorithm* ADMISSION.

We relegate the details of implementing the above algorithm (keeping track of intervals) after we show the approximation guarantee.

We say that an interval I "caused" the rejection or preemption of another interval J, either if interval I directly rejected or preempted interval J or if, for some $h \geq 2$, there exists a sequence of intervals $I = I_0, I_1, \ldots, I_h = J$ such that I_i preempted I_{i+1} for $0 \leq i \leq h-2$ and I_{h-1} directly rejected or preempted interval I_h. Fix an interval I that was accepted by the algorithm, and consider all the intervals chosen by the optimal solution, the rejection, or the preemption of which was caused by interval I. We prove that the total weight of these intervals is at most $f(\beta)$ times the weight of the accepted interval I, for some function f. Optimizing β, we get the $3+2\sqrt{2} \approx 5.828$ bound.

THEOREM 4.6. *The approximation factor of Algorithm* ADMISSION *is* $3 + 2\sqrt{2}$.

Proof. Let $\mathcal{O}$ be the set of intervals chosen by an optimal algorithm OPT. Let the set of intervals accepted by Algorithm ADMISSION be denoted by $\mathcal{A}$. For each interval $I \in \mathcal{A}$ we define a set $R(I)$ of all the intervals in $\mathcal{O}$ that are "accounted for" by I. This set consists of I in case $I \in \mathcal{O}$, and of all the intervals in $\mathcal{O}$ the rejection or preemption of which was caused by I. More formally, we have the following.

- Assume I is accepted by rule 3(c). Then, the set $R(I)$ is initialized to be I in case $I \in \mathcal{O}$ and the empty set $\emptyset$, otherwise.
- Assume I is accepted by rule 3(d). Then $R(I)$ is initialized to contain all those intervals from $\mathcal{O}$ that were directly preempted by I and the union of the sets $R(I')$ of all the intervals I' that were preempted by I. In addition, $R(I)$ contains I in case $I \in \mathcal{O}$.
- Assume $J \in \mathcal{O}$ is rejected by rule 3(b). Let $I \in \mathcal{A}$ be the interval that caused the rejection of J. Note that both I and J belong to the same job. In this case add J to $R(I)$.
- Assume $J \in \mathcal{O}$ was rejected by rule 3(e) and let $I_1, \ldots, I_h$ be the intervals in $\mathcal{A}$ that overlapped with J at the time of rejection. Let w be the weight of J and let w_j be the weight of I_j for $1 \leq j \leq h$. We view J as h imaginary intervals $J_1, \ldots, J_h$, where the weight of J_j is $\frac{w_j \cdot w}{\sum_{i=1}^{h} w_i}$ for $1 \leq j \leq h$. Set $R(I_j) := R(I_j) \cup \{J_j\}$. Note that due to the rejection rule it follows that the weight of J_j is no more than β times the weight of I_j.

It is not hard to see that each interval from $\mathcal{O}$, or a portion of it if we use rule 3(e), belongs exactly to one set $R(I)$ for some $I \in \mathcal{A}$. Thus, the union of all sets $R(I)$ for $I \in \mathcal{A}$ covers $\mathcal{O}$.

We now fix an interval $I \in \mathcal{A}$. Let w be the weight of I and let W be the sum of weights of all intervals in $R(I)$. Define $\rho = \frac{W}{w}$. Our goal is to bound ρ from above.

Interval I may directly reject at most one interval from $\mathcal{O}$. Let w_r be the weight of (the portion of) the interval $I_r \in \mathcal{O} \cap R(I)$ that was directly rejected by I, if such exists. Otherwise, let $w_r = 0$. Observe that $w_r \leq \beta w$, since, otherwise, I_r would not have been rejected. Let $I' \in \mathcal{O}$ be the interval that belongs to the same job as the one to which I belongs (it may be I itself), if such exists. By definition, the weight of I' is w. Let $W' \geq W - w - w_r$ be the sum of the weights of the rest of the intervals in $R(I)$. Define $\alpha = \frac{W'}{w}$. It follows that $\rho \leq \alpha + \beta + 1$.

We now assume inductively that the ρ bound is valid for intervals with earlier endpoint than the endpoint of I. Since the overall weight of the jobs that I directly preempted is at most w/β, we get that $\frac{w}{\beta} \cdot \rho \geq \alpha \cdot w$. This implies that $\frac{\alpha+\beta+1}{\beta} \geq \alpha$. Or equivalently, $\alpha \leq \frac{\beta+1}{\beta-1} = 1 + \frac{2}{\beta-1}$. Therefore, $\rho \leq 2 + \beta + \frac{2}{\beta-1}$. This equation is minimized for $\beta = 1 + \sqrt{2}$ which implies that $\rho \leq 3 + 2\sqrt{2}$. Finally, since the ρ bound holds for all the intervals in $\mathcal{A}$ and since the union of all $R(I)$ sets covers all the intervals taken by OPT, we get that the value of OPT is at most ρ times the value

of $\mathcal{A}$. Hence, the approximation factor is $3 + 2\sqrt{2}$. □

Implementation. Observe that step 3 of the algorithm has to be invoked only when there is a "status" change, i.e., either a new job becomes available (n times) or a job in the schedule ends (n times). Each time step 3 is invoked the total number of jobs instances that have to be examined is at most n (at most one for each job). To implement the algorithm we employ a priority queue that holds intervals according to their endpoint. At any point of time it is enough to hold at most one job instance for each job in the priority queue. It turns out that the total number of operations for retrieving the next instance is $O(n \log n)$, totaling to $O(n^2 \log n)$ operations.

4.2.2. Unrelated machines. If the number of unrelated machines is $k > 1$, we call Algorithm ADMISSION k times, machine by machine, in an arbitrary order, where the set of jobs considered in the ith call does not contain the jobs already scheduled on machines $M_1, \ldots, M_{i-1}$. The analysis that shows how the $3 + 2\sqrt{2} \approx 5.828$ bound carries over to the case of unrelated machines is very similar to the analysis presented in the proof of Theorem 4.6. The main difference is in the definition of $R(I)$. For each interval $I \in \mathcal{A}$ that was executed on machine M_i, we define the set $R(I)$ to consist of I in case $I \in \mathcal{O}$, and of all the intervals that (i) were executed on machine M_i in the optimal schedule, and (ii) the rejection or preemption of these jobs was caused by I.

5. The MAX-SNP hardness. We show that the problem of scheduling unweighted jobs on unrelated machines is MAX-SNP hard. This is done by reducing a variant of Max-2SAT, in which each variable occurs at most three times, to this problem. In this variant of Max-2SAT, we are given a collection of clauses, each consisting of two (Boolean) variables, with the additional constraint that each variable occurs at most three times, and the goal is to find an assignment of values to these variables that would maximize the number of clauses that are satisfied (i.e., contain at least one literal that has a "true" value). This problem is known to be MAX-SNP hard (cf. [33]).

Given an instance of the Max-2SAT problem we show how to construct an instance of the problem of unweighted jobs, unrelated machines, such that the value of the Max-2SAT problem is equal to the value of the scheduling problem. Each variable x_i is associated with a machine M_i. Each clause C_j is associated with a job. The release time and deadline of every job is 0 and 3, respectively. A job can be executed only on the two machines corresponding to the variables the clause C_j contains. (The processing time of this job in the rest of the machines is set to be infinite.)

Suppose that clause C_j contains a variable x_i as a positive (negative) literal. The processing time of the job corresponding to C_j on machine M_i is $3/k$, where $k \in \{1, 2, 3\}$ is the number of occurrences of variable x_i as a positive (negative) literal. Note that in case variable x_i occurs in both positive and negative forms, it occurs exactly once in one of the forms, since a variable x_i occurs at most three times overall. It follows that in any feasible schedule, machine M_i cannot execute both a job that corresponds to a positive literal occurrence and a job that corresponds to a negative literal occurrence.

We conclude that if m jobs can be scheduled, then m clauses can be satisfied. In the other direction, it is not hard to verify that if m clauses can be satisfied, then m jobs can be scheduled. Since Max-2SAT with the restriction that each variable occurs at most three times is MAX-SNP hard, the unweighted jobs and unrelated machines case is MAX-SNP hard as well.

6. Discussion and open problems. In this paper we considered the problem of finding a schedule that maximizes the weight of jobs completed before their deadline on either a single machine or multiple machines. We presented constant approximation algorithms for four variants of the problem depending on the type of the machines (identical vs. unrelated) and the weight of the jobs (identical vs. arbitrary). Most of our algorithms are based on LP rounding. We also presented a combinatorial algorithm for arbitrary job weights and unrelated machines.

Many open problems remain. In what follows we discuss some of them.

- The computational difficulty of the problems considered here is due to the "slack time" available for scheduling the jobs. Interestingly, we do not know if the simple case in which the slack equals the length of the job is as hard as the general case, or is as easy as the case with less slack.
- We showed that the problem of scheduling unweighted jobs on unrelated machines is MAX-SNP hard. We do not know whether this holds in the case of identical machines.
- We did not consider the preemptive version of our problems. Lawler [25, 26] presented some results for the preemptive case (see discussion in the introduction), yet open problems remain. In particular, the migration issue is still open. We note that it does not seem that our LP-based rounding algorithms are of use in the preemptive case.
- Several recent papers [29, 6, 9] addressed the following generalization of our problems. A job has a width that could be an integral number (i.e., the job requires more than one machine) or a real fraction (as is the case in bandwidth allocation). Constant factor approximation algorithms for this problem were obtained by [29] by generalizing our LP rounding technique, and by [6, 9] using the local ratio technique (and deriving combinatorial algorithms as well).
- Phillips, Uma, and Wein [29] used our LP-based algorithms to achieve approximation algorithms for other scheduling problems. For example, consider a problem in which an estimate of the completion time of the jobs can be computed by solving a fractional relaxation of the problem. Then, using these estimated completion times as deadlines, our algorithms can be applied so as to get a schedule where a constant fraction of the jobs indeed finish by these completion times. This observation yields new approximation algorithms for various problems, among them the minimum flow-time problem.
- Erlebach and Jansen [13] generalized our LP rounding technique and developed a general procedure for converting a coloring algorithm into an (approximate) maximum weight independent set algorithm. Using this, they obtain improved approximation factors for several problems.

Acknowledgments. We are indebted to Joel Wein for many helpful discussions, and especially for his suggestion to consider the general case of maximizing the throughput of jobs with release times and deadlines. We thank David Shmoys for helpful comments.

REFERENCES

[1] M. Adler, A. L. Rosenberg, R. K. Sitaraman, and W. Unger, *Scheduling time-constrained communication in linear networks*, in Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 1998, pp. 269–278.

[2] S. Albers, N. Garg, and S. Leonardi, *Minimizing stall time in single and parallel disk systems*, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1998, pp. 454–462.

[3] E. M. Arkin and E. B. Silverberg, *Scheduling jobs with fixed start and end times*, Discrete Appl. Math., 18 (1987), pp. 1–8.

[4] B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén, *On-line competitive algorithms for call admission in optical networks*, in Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, New York, 1996, pp. 431–444.

[5] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, *Competitive non-preemptive call control*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 312–320.

[6] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, *A unified approach to approximating resource allocation and scheduling*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2000, pp. 735–744.

[7] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, SIAM J. Comput., 28 (1999), pp. 1806–1828.

[8] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of on-line real-time task scheduling*, Real-Time Systems, 4 (1992), pp. 125–144.

[9] P. Berman and B. DasGupta, *Improvements in throughput maximization for real-time scheduling*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2000, pp. 680–687.

[10] S. Boyd and R. Carr, *A new bound for the ratio between the* 2*–matching problem and its linear programming relaxation*, Math. Program., Ser. A, 86 (1999), pp. 499–514.

[11] J. Blazewicz, K. H. Ecker, G. Schmidt, and J. Weglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, New York, 1994.

[12] G. Cornuejols, M. Fisher, and G. Nemhauser, *Location of bank accounts to optimize float*, Management Science, 23 (1977), pp. 789–810.

[13] T. Erlebach and K. Jansen, *Conversion of coloring algorithms into maximum weight independent set algorithms*, in Proceedings of the Workshop "Approximation and Randomization Algorithms in Communication Networks" (ARACNE 2000), Proceedings on Informatics 8, Carleton Scientific, Waterloo, ON, Canada, 2000, pp. 135–145.

[14] U. Faigle and W. M. Nawijn, *Note on scheduling intervals on-line*, Discrete Appl. Math., 58 (1995), pp. 13–17.

[15] M. Fischetti, S. Martello, and P. Toth, *The fixed job schedule problem with spread-time constraints*, Oper. Res., 35 (1987), pp. 849–858.

[16] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, *Efficient on-line call control algorithms*, J. Algorithms, 23 (1997), pp. 180–194.

[17] M. R. Garey and D. S. Johnson, *Two-processor scheduling with start-times and deadlines*, SIAM J. Comput., 6 (1977), pp. 416–426.

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

[19] S. A. Goldman, J. Parwatikar, and S. Suri, *On-line scheduling with hard deadlines*, in Proceedings of the 5th International Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 1272, Springer-Verlag, New York, 1997, pp. 258–271.

[20] M. H. Goldwasser, *Patience is a virtue: The effect of slack on competitiveness for admission control*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 396–405.

[21] N. G. Hall and M. J. Magazine, *Maximizing the value of a space mission*, European J. Oper. Res., 78 (1994), pp. 224–241.

[22] B. Kalyanasundaram and K. Pruhs, *Eliminating migration in multi-processor scheduling*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 499–506.

[23] H. Kise, T. Ibaraki, and H. Mine, *A solvable case of one machine scheduling problem with ready and due dates*, Oper. Res., 26 (1978), pp. 121–126.

[24] G. Koren and D. Shasha, *An optimal on-line scheduling algorithm for overloaded real-time systems*, SIAM J. Comput., 24 (1995), pp. 318–339.

[25] E. L. Lawler, *Sequencing to minimize the weighted number of tardy jobs*, Recherche Operationnel, 10 (1976), pp. 27–33.

[26] E. L. Lawler, *A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs*, Ann. Oper. Res., 26 (1990), pp. 125–133.

[27] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *Sequencing and Scheduling: Algorithms and Complexity*, in Handbooks in Operations Research and Management Science, Vol. 4: Logistics for Production and Inventory, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, eds., North Holland, Amsterdam, 1993, pp. 445–522.

[28] R. J. Lipton and A. Tomkins, *Online interval scheduling*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 302–311.

[29] C. Phillips, R. N. Uma, and J. Wein, *Off-line admission control for general scheduling problems*, in Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, 2000, pp. 879–888.

[30] S. Sahni, *Algorithms for scheduling independent tasks*, J. ACM, 23 (1976), pp. 116–127.

[31] F. C. R. Spieksma, *On the approximability of an interval scheduling problem*, J. Sched., 2 (1999), pp. 215–227.

[32] G. Woeginger, *When does a dynamic programming formulation guarantee the existence of an FPTAS?*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1999, pp. 820–829.

[33] M. Yannakakis, *On the approximation of maximum satisfiability*, in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1992, pp. 1–9.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 353–363

LOW REDUNDANCY IN STATIC DICTIONARIES WITH CONSTANT QUERY TIME*

RASMUS PAGH†

Abstract. A *static dictionary* is a data structure storing subsets of a finite universe U, answering membership queries. We show that on a unit cost RAM with word size $\Theta(\log |U|)$, a static dictionary for n-element sets with constant worst case query time can be obtained using $B+O(\log\log|U|)+o(n)$ bits of storage, where $B = \lceil \log_2 \binom{|U|}{n} \rceil$ is the minimum number of bits needed to represent all n-element subsets of U.

Key words. information retrieval, dictionary, hashing, redundancy, compression

AMS subject classifications. 68P10, 68P20, 68P30

PII. S0097539700369909

1. Introduction.

Consider the problem of storing a subset S of a finite set U, such that membership queries, "$u \in S$?", can be answered in worst case constant time on a unit cost RAM. We are interested only in membership queries, so we assume that $U = \{0, \ldots, m-1\}$. Also, we restrict attention to the case where the RAM has word size $\Theta(\log m)$. In particular, elements of U can be represented within a constant number of machine words, and the usual RAM operations (including multiplication) on numbers of size $m^{O(1)}$ can be done in constant time.

Our goal will be to solve this, the *static dictionary problem*, using little memory, measured in consecutive bits. We express the complexity in terms of $m = |U|$ and $n = |S|$, and often consider the asymptotics when n is a function of m. Since the queries can distinguish any two subsets of U, we need at least $\binom{m}{n}$ different memory configurations, that is, at least $B = \lceil \log \binom{m}{n} \rceil$ bits. (Log is base 2 throughout this paper.) We will focus on the case $n \le m/2$ and leave the symmetry implications to the reader. Using Stirling's approximation to the factorial function, one can derive the following (where $e = 2.718\ldots$ denotes the base of the natural logarithm):

$$B = n\log(e\,m/n) - \Theta(n^2/m) - O(\log n). \tag{1.1}$$

It should be noted that using space very close to B is only possible if elements of S are stored *implicitly*, since explicitly representing all elements requires $n\log m = B + \Omega(n\log n)$ bits.

Previous work. The static dictionary is a very fundamental data structure and has been much studied. We focus on the development in space consumption for schemes with worst case constant query time. A bit vector is the simplest possible solution to the problem, but the space complexity of m bits is poor compared to B unless $n \approx m/2$. By the late 1970s, known dictionaries with a space complexity of $O(n)$ words (i.e., $O(n\log m)$ bits) either had nonconstant query time or worked only for restricted universe sizes [4, 16, 17].

*Received by the editors April 3, 2000; accepted for publication (in revised form) November 27, 2000; published electronically August 8, 2001. A preliminary version of this paper appeared at ICALP 1999 [12].

http://www.siam.org/journals/sicomp/31-2/36990.html

†BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation), University of Aarhus, Aarhus, Denmark (pagh@brics.dk).

The breakthrough paper of Fredman, Komlós, and Szemerédi [7] described a general constant time hashing scheme, from now on referred to as the FKS dictionary, using $O(n)$ words. A refined solution in the paper uses $B+O(n\log n+\log\log m)$ bits, which is $O(B)$ when $n=m^{1-\Omega(1)}$. Brodnik and Munro [2] constructed the first static dictionary using $O(B)$ bits with no restrictions on m and n. They later improved the bound to $B+O(B/\log\log\log m)$ bits [3].

No nontrivial space lower bound is known in a general model of computation. However, various restrictions on the data structure and the query algorithm have been successfully studied. Yao [17] showed that if words of the data structure must contain elements of S, the number of words necessary for $o(\log n)$ time queries cannot be bounded by a function of n. Fich and Miltersen [6] studied a RAM with standard unit cost arithmetic operations but *without division and bit operations*, and showed that query time $o(\log n)$ requires $\Omega(m/n^\epsilon)$ words of memory for any $\epsilon>0$. Miltersen [11] showed that on a RAM with bit operations but *without multiplication*, one needs m^ϵ words, for some $\epsilon>0$, when $n=m^{o(1)}$.

This paper. In this paper we show that it is possible to achieve space usage very close to the information theoretical minimum of B bits. The additional term of the space complexity, which we will call the *redundancy*, will be $o(n)+O(\log\log m)$ bits. More precisely we show the following.

THEOREM 1.1. *The static dictionary problem with worst case constant query time can be solved using $B+O(n\,(\log\log n)^2/\log n+\log\log m)$ bits of storage.*

Theorem 1.1 improves the redundancy of $\Omega(\min(n\log\log m, m/(\log n)^{o(1)}))$ obtained by Brodnik and Munro [3] by a factor of $\Omega(\min(n, \log\log m\,(\log n)^{1-o(1)}))$. For example, when $n=\Omega(m)$ we obtain space $B+B/(\log B)^{1-o(1)}$ as compared to $B+B/(\log B)^{o(1)}$. For $n=\Theta(\log m)$ our space usage is $B+n/(\log n)^{1-o(1)}$ rather than $B+\Omega(n\log n)$.

We will also show how to associate *satellite data* from a finite domain to each element of S, with nearly the same redundancy as above.

Our main observation is that one can save space by "compressing" the hash table part of data structures based on (perfect) hashing, storing in each cell not an element of S but only a *quotient* — information that distinguishes the element from the part of U that hashes to the cell.[1] This technique, referred to as quotienting, is presented in section 2, where a $B+O(n+\log\log m)$ bit dictionary is exhibited. Section 3 outlines how to improve the dependency on n to that of theorem 1.1. The construction uses a dictionary supporting *rank* and *predecessor* queries, described in section 4. Section 5 gives the details of the construction and an analysis of the redundancy. The sizes of the data structures described are not computed explicitly. Rather, indirect means are employed to determine the number of redundant bits. While direct summation and comparison with (1.1) would be possible, it is believed that the proofs given here contain more information about the "nature" of the redundancy.

Without loss of generality, we will assume that n is greater than some sufficiently large constant. This is to avoid worrying about special cases for small values of n.

2. First solution. This section presents a static dictionary with a space consumption of $B+O(n+\log\log m)$ bits. Consider a *minimal perfect hash function* for S, i.e., $h_{\text{perfect}}: U\to\{0,\ldots,n-1\}$ which is 1-1 on S. Defining an n-cell *hash table* T

[1] The term "quotient" is inspired by the use of modulo functions for hashing, in which case the integer quotient is exactly what we want in the cell.

such that $T[i] = s_i$ for the unique $s_i \in S$ with $h_{\text{perfect}}(s_i) = i$, the following program implements membership queries for S:

function lookup(x)
  return ($T[h_{\text{perfect}}(x)] = x$);
end.

A more compact data structure results from the observation that $T[i]$ does not need to contain s_i itself ($\lceil \log m \rceil$ bits), but only enough information to identify s_i within $U_i = \{u \in U \mid h_{\text{perfect}}(u) = i\}$ ($\lceil \log |U_i| \rceil$ bits). We will be slightly less ambitious, though, and not necessarily go for the minimal number of bits in each hash table cell. In particular, to allow efficient indexing we want the number of bits to be the same for each table cell. Note that $\log(m/n)$ bits is a lower bound on the size of a cell, since the average size of the U_i is m/n.

To compute the information needed in the hash table, we define a *quotient function* (for h_{perfect}) as a function $q : U \to \{0, \ldots, r-1\}$, $r \in \mathbf{N}$, which is 1-1 on each set U_i. Given such a function, let $T'[i] = q(T[i])$, and the following program is equivalent to the above:

function lookup'(x)
  return ($T'[h_{\text{perfect}}(x)] = q(x)$);
end.

Thus it suffices to use the hash table T' of $\lceil \log r \rceil$-bit entries. By the above discussion, we ideally have that r is close to m/n.

Although the FKS dictionary [7] is not precisely of the form "minimal perfect hash function + hash table," it is easy to modify it to be of this type. We will thus speak of the FKS minimal perfect hash function, h_{FKS}. It has a quotient function which is evaluable in constant time, and costs no extra space in that its parameters k, p, and a are part of the data structure already:

$$q : u \mapsto (u \operatorname{div} p)\, \lceil p/a \rceil + (k\, u \bmod p) \operatorname{div} a. \tag{2.1}$$

Intuitively, this function gives the information that is thrown away by the modulo applications of the scheme's top-level hash function:

$$h : u \mapsto (k\, u \bmod p) \bmod a, \tag{2.2}$$

where k and a are positive integers and $p > a$ is prime. We will not give a full proof that q is a quotient function of h_{FKS}, since our final result does not depend on this. However, the main part of the proof is a lemma that will be used later, showing that q is a quotient function for h.

LEMMA 2.1. *For $U_i = \{u \in U \mid h(u) = i\}$ where $i \in \{0, \ldots, a-1\}$, q is* 1-1 *on U_i. Further, $q[U] \subseteq \{0, \ldots, r-1\}$, where $r = \lceil m/p \rceil \lceil p/a \rceil$.*

Proof. Let $u_1, u_2 \in U_i$. If $q(u_1) = q(u_2)$ we have that $u_1 \operatorname{div} p = u_2 \operatorname{div} p$ and $(k\, u_1 \bmod p) \operatorname{div} a = (k\, u_2 \bmod p) \operatorname{div} a$. From the latter equation and $h(u_1) = h(u_2)$, it follows that $k\, u_1 \bmod p = k\, u_2 \bmod p$. Since p is prime and $k \neq 0$ this implies $u_1 \bmod p = u_2 \bmod p$. Since also $u_1 \operatorname{div} p = u_2 \operatorname{div} p$ it must be the case that $u_1 = u_2$, so q is indeed 1-1 on U_i. The bound on the range of q is straightforward. □

In the FKS scheme, $p = \Theta(m)$ and $a = n$, so the range of q has size $O(m/n)$ and $\log(m/n) + O(1)$ bits suffice to store each hash table element. The space needed to store h_{FKS}, as described in [7], is not good enough to show the result claimed at the beginning of this section. However, Schmidt and Siegel [15] have shown how to implement (essentially) h_{FKS} using $O(n + \log\log m)$ bits of storage (which is optimal up to a constant factor; see, e.g., [10, Theorem III.2.3.6]). The time needed to evaluate the hash function remains constant. Their top-level hash function is not of the form (2.2), but the composition of two functions of this kind, h_1 and h_2. Call the corresponding quotient functions q_1 and q_2. A quotient function for $h_2 \circ h_1$ is $u \mapsto (q_1(u), q_2(h_1(u)))$, which has a range of size $O(m/n)$. One can thus get a space usage of $n\log(m/n) + O(n)$ bits for the hash table, and $O(n + \log\log m)$ bits for the hash function, so by (1.1) we have the following proposition.

PROPOSITION 2.2. *The static dictionary problem with worst case constant query time can be solved using $B + O(n + \log\log m)$ bits of storage.*

As a by-product we get the following corollary.

COROLLARY 2.3. *When $n > c \log\log m/\log\log\log m$, for a suitable constant $c > 0$, the static dictionary problem with worst case constant query time can be solved using n words of $\lceil \log m \rceil$ bits.*

Proof. The dictionary of Proposition 2.2 uses $n\log m - n\log n + O(n + \log\log m)$ bits. For suitable constants c and N, the $O(n + \log\log m)$ term is less than $n\log n$ when $n > N$. If $n \le N$ we can simply list the elements of S. □

The previously best result of this kind needed $n \ge (\log m)^c$ for some constant $c > 0$ [5]. (An interesting feature of this nonconstructive scheme is that it is *implicit*, i.e., the n words contain a permutation of the elements in S.) The question whether n words suffice in all cases was posed in [6].

3. Overview of final construction. This section describes the ideas which allow us to improve the $O(n)$ term of Proposition 2.2 to $o(n)$. There are two redundancy bottlenecks in the construction of the previous section:

- The Schmidt–Siegel hash function is $\Omega(n)$ bit redundant.
- The hash table is $\Omega(n)$ bit redundant.

The first bottleneck seems inherent to the Schmidt–Siegel scheme: it appears there is no easy way of improving the space usage to $1 + o(1)$ times the minimum, at least if constant evaluation time is to be preserved. The second bottleneck is due to the fact that m/n may not be close to a power of two, and hence the space consumption of $n\lceil \log r \rceil$ bits, where $r \ge m/n$, may be $\Omega(n)$ bits larger than the ideal of $n\log(m/n)$ bits. Our way around these bottlenecks starts with the following observations:

- We need only to solve the dictionary problem for some "large" subset $S_1 \subseteq S$.
- We can look at some universe $U_1 \subseteq U$, where $S_1 \subseteq U_1$ and $|U_1|/|S_1|$ is "close to" a power of 2.

The first observation helps by allowing "less than perfect" hash functions which occupy much less memory. The remaining elements, $S_2 = S \backslash S_1$, can be put in a more redundant dictionary, namely the refined FKS dictionary [7]. The second observation gives a way of minimizing redundancy in the hash table.

We will use a hash function of the form (2.2). The following result from [7] shows that (unless a is not much larger than n) it is possible to choose k such that h is 1-1 on a "large" set S_1.

LEMMA 3.1. *If $u \mapsto u \bmod p$ is 1-1 on S, then for at least half the values of $k \in \{1, \ldots, p-1\}$, there exists a set $S_1 \subseteq S$ of size $|S_1| \ge (1 - O(n/a))\,|S|$, on which h is 1-1.*

Without loss of generality we will assume that $|S_1| = n_1$, where n_1 only depends on n and a, and $n_2 = n - n_1 = O(n/a)$. The hash function h is not immediately useful, since it has a range of size $a \gg n_1$. To obtain a minimal perfect hash function for S_1, we compose with a function $g : \{0, \ldots, a-1\} \to \{0, \ldots, n_1 - 1\} \cup \{\perp\}$ which has $g[h[S_1]] = \{0, \ldots, n_1 - 1\}$. (In particular, it is 1-1 on $h[S_1]$.) The extra value of g allows us to look at $U_1 = \{u \in U \mid g(h(u)) \neq \perp\}$, which clearly contains S_1. We require that $g(v) = \perp$ when $v \notin h[S_1]$, since this makes q a quotient function for $g \circ h$ (restricted to inputs in U_1).

The implementation of the function g has the form of a dictionary for $h[S_1]$ within $\{0, \ldots, a-1\}$, which apart from membership queries answers *rank queries*. (The result of a rank query on input v is $|\{w \in h[S_1] \mid w < v\}|$.) So we may take g as the function that returns $\perp$ if its input v is not in the set, and otherwise returns the rank of v. The details on how to implement the required dictionary are given in section 4.

The dictionary of section 4 will also be our dictionary of choice when $m \leq n \log^3 n$ (the "dense" case). Only when $m > n \log^3 n$ do we use the scheme described in this section. This allows us to choose suitable values of hash function parameters p and a (where $p = O(n^2 \log m)$ and $a = \Theta(n (\log n)^2) \leq m$), such that the range of the quotient function, $r = \lceil m/p \rceil \lceil p/a \rceil$, is close to m/a. The details of this, along with an analysis of the redundancy of the resulting dictionary, can be found in section 5.

4. Dictionaries for dense subsets. In this section we describe a dictionary which has the redundancy stated in Theorem 1.1 when $m = n (\log n)^{O(1)}$. Apart from membership queries, it will support queries on the ranks of elements (the rank of u is $|\{v \in S | v < u\}|$), as well as queries on predecessors (the predecessor of u is $\max\{v \in S \mid v < u\}$).

As a first step, we describe a dictionary which has redundancy dependent on m, namely $O(m \log\log m / \log m)$ bits. The final dictionary uses the first one as a substructure.

4.1. Block compression. The initial idea is to split the universe into blocks $U_i = \{b\,i, \ldots, b\,(i+1) - 1\}$ of size $b = \lceil \frac{1}{2} \log m \rceil$, and store each block in a compressed form. (This is similar to the ideas of range reduction and "a table of small ranges" used in [3].) To simplify things we may assume that b divides m (otherwise consider a universe at most $b - 1$ elements larger, increasing the space usage by $O(b)$ bits). If a block contains j elements from S, the compressed representation is the number j ($\lfloor \log\log m \rfloor$ bits) followed by a number in $\{1, \ldots, \binom{b}{j}\}$ corresponding to the particular subset with i elements ($\lceil \log \binom{b}{j} \rceil$ bits). Extraction of information from a compressed block is easy, since any function of the block representations can be computed by table lookup. (The crucial thing being that, since representations have size at most $\frac{1}{2} \log m + \log\log m$ bits, the number of entries in such a table makes its space consumption negligible compared to $O(m \log\log m / \log m)$ bits.)[2]

Let $n_i = |S \cap U_i|$ and $B_i = \lceil \log \binom{b}{n_i} \rceil$. The overall space consumption of the above encoding is $\sum_i B_i + O(m \log\log m / \log m)$. Let s denote the number of blocks, $s = O(m / \log m)$. A lemma from [2] bounds the above sum by $B + s$.

LEMMA 4.1 (Brodnik–Munro). *Let $m_0, \ldots, m_{s-1}$ and $n_0, \ldots, n_{s-1}$ be nonnegative integers. The following inequality holds:*

$$\sum_{i=0}^{s-1} \lceil \log \tbinom{m_i}{n_i} \rceil < \log \binom{\sum_{i=0}^{s-1} m_i}{\sum_{i=0}^{s-1} n_i} + s.$$

[2] Alternatively, assume that the RAM has instructions to extract the desired information.

Proof. We have $\sum_{i=0}^{s-1}\lceil\log\binom{m_i}{n_i}\rceil < \sum_{i=0}^{s-1}\log\binom{m_i}{n_i} + s \le \log\binom{\sum_{i=0}^{s-1} m_i}{\sum_{i=0}^{s-1} n_i} + s$. The latter inequality follows from the fact that there are at least $\prod_{i=0}^{s-1}\binom{m_i}{n_i}$ ways of picking $\sum_{i=0}^{s-1} n_i$ elements out of $\sum_{i=0}^{s-1} m_i$ elements (namely, by picking n_1 among the m_1 first, n_2 among the m_2 next, etc.). □

We need an efficient mechanism for extracting rank and predecessor information from the compressed representation. In particular we need a way of finding the start of the compressed representation of the ith block. The following result, generalizing a construction in [16], is used.

PROPOSITION 4.2 (Tarjan–Yao). *A sequence of integers $z_1, \ldots, z_k$, where for all $1 < i \le k$ we have $|z_i| = n^{O(1)}$ and $\max(|z_i|, |z_i - z_{i-1}|) = (\log n)^{O(1)}$, can be stored in a data structure allowing constant time random access, using $O(k \log\log n)$ bits of memory.*

Proof. Every $\lceil\log n\rceil$th integer is stored "verbatim," using a total of $O(k)$ bits. All other integers are stored as either an offset relative to the previous of these values, or as an absolute value. (One of these has size $(\log n)^{O(1)}$.) This uses $O(k \log\log n)$ bits in total. □

Placing the compressed blocks consecutively in numerical order, the sequence of pointers to the compressed blocks can be stored by this method. Also, the rank of the first element in each block can be stored like this. Finally, we may store the distance to the predecessor of the first element in each block (from which the predecessor is simple to compute). All of these data structures use $O(m \log\log m/\log m)$ bits. Ranks and predecessors of elements within a block can be found by table lookup, as sketched above. So we have the following proposition.

PROPOSITION 4.3. *A static dictionary with worst case constant query time, supporting rank and predecessor queries, can be stored in $B + O(m \log\log m/\log m)$ bits.*

4.2. Interval compression. The dictionary of section 4.1 has the drawback that the number of compressed blocks, and hence the redundancy, grows almost linearly with m. For $m \le n(\log n)^c$, where c is any integer constant, the number of "compressed units" can be reduced to $O(n \log\log n/\log n)$ by instead compressing *intervals* of varying length. We make sure that the compressed representations have length $(1 - \Omega(1))\log n$ (so that information can be extracted by lookup in a table of negligible size) by using intervals of size $O((\log n)^{c+1})$ with at most $\log n/(2c \log\log n)$ elements. We must be able to retrieve the interval number and position within the interval for any element of U in constant time. The block compression scheme of section 4.1 is then trivially modified to work with intervals, and the space for auxiliary data structures becomes $O(n(\log\log n)^2/\log n)$ bits.

We now proceed to describe the way in which intervals are formed and represented. Let $d = \lfloor\sqrt{\log n}\rfloor$, and suppose without loss of generality that d^{2c} divides m. (Considering a universe at most d^{2c} elements larger costs $O(d^{2c})$ bits, which is negligible.) Our first step is to partition U into "small blocks" U_i, satisfying $|S \cap U_i| \le \log n/(2c \log\log n)$. These will later be clustered to form the intervals. The main tool is the dictionary of Proposition 4.3, which is used to locate areas with a high concentration of elements from S. More specifically, split U into at most n blocks of size d^{2c} and store the indices of blocks that are *not* small, i.e., contain more than $\log n/(2c \log\log n)$ elements from S. Since at most $2cn \log\log n/\log n$ blocks are not small, the memory for this data structure is $O(n(\log\log n)^2/\log n)$ bits. The part of the universe contained in nonsmall blocks has size at most $2cm \log\log n/\log n \le n\, d^{2c-1}$. A rank query can be used to map the elements of nonsmall blocks injectively

and in an order preserving way to a subuniverse of this size. The splitting is repeated recursively on this subuniverse, now with at most n blocks of size d^{2c-1}. Again, the auxiliary data structure uses $O(n\,(\log\log n)^2/\log n)$ bits. At the bottom of the recursion we arrive at a universe of size at most nd, and every block of size d is small. This defines our partition of U into $O(n)$ small blocks, which we number $0, 1, 2, \ldots$ in order of increasing elements. Note that the small block number of any element in U can be computed by a rank query and a predecessor query at each level.

As every small block has size at most $(\log n)^c$, intervals can be formed by up to $\log n$ consecutive small blocks, together containing at most $\log n/(2c\log\log n)$ elements of S. The "greedy" way of choosing such compressible intervals from left to right results in $O(n\log\log n/\log n)$ intervals, as no two adjacent intervals can both contain less than $\log n/(4c\log\log n)$ elements and be shorter than $(\log n)^{c+1}$. To map the $O(n)$ block numbers to interval numbers, we use the dictionary of Proposition 4.3 to store the number of the first small block in each interval, using $O(n\,(\log\log n)^2/\log n)$ bits. A rank query on a small block number then determines the interval number. Finally, the first element of each interval is stored using Proposition 4.2, allowing positions within intervals to be computed, once again using $O(n\,(\log\log n)^2/\log n)$ bits.

THEOREM 4.4. *For $m = n\,(\log n)^{O(1)}$, a static dictionary with worst case constant query time, supporting rank and predecessor queries, can be stored in $B + O(n\,(\log\log n)^2/\log n)$ bits.*

5. Dictionary for sparse subsets. In this section we fill out the remaining details of the construction described in section 3, and provide an analysis of the redundancy obtained. By section 4 we need only consider the case $m > n\,(\log n)^c$ for some constant c. (It will turn out that $c = 3$ suffices.)

5.1. Choice of parameters. We need to specify how hash function parameters a and p are chosen. (A choice of k then follows by Lemma 3.1.) Parameter a will depend on p, but is bounded from above by $A(n)$ and from below by $A(n)/3$, where A is a function we specify later. For now, let us just say that $A(n) = n\,(\log n)^{\Theta(1)}$. (Our construction requires $A(n) = n\,(\log n)^{O(1)}$, and we want $A(n)$ large in order to make $S\backslash S_1$ small.) Parameter p will have size $O(n^2\log m)$, so it can be stored using $O(\log n + \log\log m)$ bits. It is chosen such that $u \mapsto u \bmod p$ is 1-1 on S, and such that $r = \lceil m/p\rceil\,\lceil p/a\rceil$ is not much larger than m/a.

LEMMA 5.1. *For m larger than some constant, there exists a prime p in each of the following ranges, such that $u \mapsto u \bmod p$ is 1-1 on S:*

1. $n^2\ln m \le p \le 3\,n^2\ln m$.
2. $m \le p \le m + m^{2/3}$.

Proof. The existence of a suitable prime between $n^2\ln m$ and $3\,n^2\ln m$ is guaranteed by the prime number theorem (in fact, at least half of the primes in the interval will work). See [7, Lemma 2] for details. By [9] the number of primes between m and $m + m^\theta$ is $\Omega(m^\theta/\log m)$ for any $\theta > 11/20$. Take $\theta = 2/3$ and let p be such a prime; naturally the map is then 1-1. □

A prime in the first range will be our choice for p when $m > A(n)\,n^2\ln m$; otherwise we choose a prime in the second range. In the first case, $r < (m/p+1)(p/a+1) = (1 + a/p + p/m + a/m)\,m/a$. In the second case, $r = \lceil p/a\rceil < (m + m^{2/3})/a + 1 \le (1 + a/m + m^{-1/3})\,m/a$. Since we can assume $m > a\log n$, we have in both cases that $r = (1 + O(1/\log n))\,m/a$. We make r close to a power of 2 by suitable choice of parameter a.

LEMMA 5.2. *For any $x, y \in \mathbf{R}_+$ and $z \in \mathbf{N}$, with $x/z \geq 3$, there exists $z' \in \{z+1, \ldots, 3z\}$, such that $\lceil \log \lceil x/z' \rceil + y \rceil \leq \log(x/z') + y + O(z/x + 1/z)$.*

Proof. Since $x/z \geq 3$, it follows that $\log\lceil x/z \rceil + y$ and $\log\lceil x/3z \rceil + y$ have different integer parts. So there exists z', $z < z' \leq 3z$, such that $\lceil \log \lceil x/z' \rceil + y \rceil \leq \log\lceil x/(z'-1) \rceil + y$. A simple calculation gives $\log\lceil x/(z'-1) \rceil + y = \log(x/(z'-1)) + y + O(z/x) = \log(x/z') + \log(z'/(z'-1)) + y + O(z/x) = \log(x/z') + y + O(z/x + 1/z)$, and the conclusion follows. □

Since $\log r = \log\lceil p/a \rceil + \log\lceil m/p \rceil$ and $p/A(n) \geq 3$ (for n large enough), the lemma gives an a satisfying $A(n)/3 \leq a \leq A(n)$, such that $\lceil \log r \rceil = \log r + O(a/p + 1/a) = \log((1 + O(1/\log n))\, m/a)$.

To conclude, we can choose p and a such that the number of bit patterns in each hash table cell, $2^{\lceil \log r \rceil}$, is $(1 + O(1/\log n))\, m/a$.

5.2. Storing parameters. A slightly technical point remains, concerning the storage of parameters in the data structure. If the universe size m is supposed to be implicitly known, there is no problem storing the parameters using $O(\log n + \log\log m)$ bits (say, using $O(\log\log m)$ bits to specify the number of bits for each parameter). However, if m is considered a parameter unknown to the query algorithm, it is not clear how to deal with, e.g., queries for numbers larger than m, without actually using $O(\log m)$ extra bits to store m. Our solution is to look at a slightly larger universe U', whose size is specified using $O(\log n + \log\log m)$ bits. Using $O(\log n + \log\log m)$ bits we may store $\lceil \log m \rceil$ (by assumption we know the number of bits needed to store this number within an additive constant) and the $\lceil \log n \rceil$ most significant bits of m. This defines $m' = (1 + O(1/n))\, m$, the universe size of U'. We need to estimate the information theoretical minimum of the new problem, $B' = \lceil \binom{m'}{n} \rceil$.

LEMMA 5.3. *For $n < m_1 < m_2$ we have $\log \binom{m_2}{n} - \log \binom{m_1}{n} < n \log(\frac{m_2 - n}{m_1 - n})$.*

Proof. We have $\binom{m_2}{n} / \binom{m_1}{n} = \frac{m_2\,(m_2-1)\ldots(m_2-n+1)}{m_1\,(m_1-1)\ldots(m_1-n+1)} < (\frac{m_2-n}{m_1-n})^n$. □

Thus, since $n \leq m/2$, $B' = B + O(n \log(m'/m)) = B + O(n/\log n)$. So our slight expansion of the universe is done without affecting the redundancy of Theorem 1.1.

5.3. Redundancy analysis. First note that we can assume all parts of the data structure to have size depending only on m and n (that is, not on the particular set stored). Hence, the entire data structure is a bit pattern of size $B + f(n, m)$, for some function f. To show the bound $f(n, m) = O(n\,(\log\log n)^2/\log n + \log\log m)$, we construct a function ϕ, mapping n-element subsets of U to subsets of $\{0,1\}^{B+f(n,m)}$, such that

- $\log|\phi(S)| = O(n\,(\log\log n)^2/\log n + \log\log m)$.
- $\bigcup_S \phi(S) = \{0,1\}^{B+f(n,m)}$.

This implies $B + f(n, m) \leq \log(\sum_S |\phi(S)|) = B + O(n\,(\log\log n)^2/\log n + \log\log m)$, as desired. Recall that the data structure consists of

- hash function parameters and pointers ($b_1 = O(\log n + \log\log m)$ bits);
- a dictionary supporting rank, representing the function g via a set of n_1 elements in $\{0, \ldots, a-1\}$ ($b_2 = \log \binom{a}{n_1} + O(n\,(\log\log n)^2/\log n)$ bits);
- a hash table ($b_3 = n_1 \lceil \log r \rceil$ bits);
- a dictionary representing a set of size n_2 in U ($b_4 = \log \binom{m}{n_2} + O(n_2 \log n_2 + \log\log m)$ bits).

Since the dictionary supporting rank has redundancy $O(n\,(\log\log n)^2/\log n)$, there exists a function ϕ' from the n_1-element subsets of $\{0, \ldots, a-1\}$ to $\{0,1\}^{b_2}$, such that $\cup_{\tilde{S}_1} \phi'(\tilde{S}_1) = \{0,1\}^{b_2}$ and $\log|\phi'(\tilde{S}_1)| = O(n\,(\log\log n)^2/\log n)$. Similarly, there exists a function ϕ'' from the n_2-element subsets of U to $\{0,1\}^{b_4}$, such that $\cup_{\tilde{S}_2} \phi''(\tilde{S}_2) =$

$\{0,1\}^{b_4}$ and $\log|\phi''(\tilde{S}_2)| = O(n_2 \log n_2 + \log\log m)$. Choosing $A(n) = n \log^2 n$ we have $n_2 = O(n/\log^2 n)$, and hence $\log|\phi''(\tilde{S}_2)| = O(n/\log n + \log\log m)$.

Let h_1 denote the hash function $u \mapsto (u \bmod p) \bmod a$. (Any hash function of the form (2.2) would do; we pick this one for simplicity.) By Lemma 5.3 we can assume that p divides m, since either $m \le p \le m+m^{2/3}$, in which case expanding the universe to $\{0,\ldots,p-1\}$ increases the information theoretical minimum by $O(n/m^{1/3})$, or $p = O(m/n)$, in which case the increase by rounding m to the nearest higher multiple of p is $O(1)$.

When p divides m, the number of elements hashed to a cell by h_1 is at least $\lfloor m/a \rfloor$. Hence for any function g, there is a set $T(g)$ of at least $(m/a-1)^{n_1}$ possible bit patterns in the hash table. The total number of bit patterns is $2^{\lceil \log r \rceil\, n_1} = ((1+O(1/\log n))\, m/a)^{n_1}$, so the ratio between this and the $|T(g)|$ patterns used is

$$\left(\tfrac{1+O(1/\log n)}{1-a/m}\right)^{n_1} = (1+O(1/\log n))^{n_1} = 2^{O(n/\log n)}.$$

Thus, there exists a function ϕ_g from $T(g)$ onto $\{0,1\}^{b_3}$, such that $\log|\phi_g(z)| = O(n/\log n)$. For notational convenience we will from now on denote bit patterns in the hash table simply by the corresponding set of universe elements.

We will take $\phi(S)$ as the union of sets $\phi(\tilde{S}_1, \tilde{S}_2)$, over all $\tilde{S}_1, \tilde{S}_2 \subseteq S$ with $|\tilde{S}_1| = n_1$ and $|\tilde{S}_2| = n_2$. If $|h_1[\tilde{S}_1]| \ne n_1$, we set $\phi(\tilde{S}_1, \tilde{S}_2) = \emptyset$. Otherwise $h_1[\tilde{S}_1]$ defines the function g, and we set

$$\phi(\tilde{S}_1, \tilde{S}_2) = \{s_1\, s_2\, s_3\, s_4 \mid s_1 \in \{0,1\}^{b_1},\ s_2 \in \phi'(h_1[\tilde{S}_1]),\ s_3 \in \phi_g(\tilde{S}_1),\ s_4 \in \phi''(\tilde{S}_2)\}\ .$$

By our bounds on the sizes of $\phi'(h_1[\tilde{S}_1])$, $\phi_{h_1,g}(\tilde{S}_1)$, and $\phi''(\tilde{S}_2)$, we conclude that $\log|\phi(\tilde{S}_1, \tilde{S}_2)| = O(n\,(\log\log n)^2/\log n + \log\log m)$. Since $\phi(S)$ is the union of the $2^{O(n/\log n)}$ sets of the form $\phi(\tilde{S}_1, \tilde{S}_2)$, it follows that the requirement on $|\phi(S)|$ holds.

To see that $\bigcup_S \phi(S) = \{0,1\}^{B+f(n,m)}$, take any $x \in \{0,1\}^{B+f(n,m)}$. Let $x = s_1\, s_2\, s_3\, s_4$, where $s_i \in \{0,1\}^{b_i}$. By definition of ϕ' and ϕ'', there is some set $T \subseteq \{0,\ldots,a-1\}$, $|T| = n_1$, such that $s_2 \in \phi'(T)$, and a set $\tilde{S}_2 \subseteq U$, $|\tilde{S}_2| = n_2$, such that $s_4 \in \phi''(S_2)$. The set T corresponds to a function g. From the way we defined ϕ_g, there exists a bit pattern $z \in T(g)$, such that $s_3 \in \phi_g(z)$. Let $\tilde{S}_1$ be the set of size n_1 corresponding to h_1, g, and z. We then have that $x \in \phi(\tilde{S}_1, \tilde{S}_2)$, so if we take $S \supseteq \tilde{S}_1 \cup \tilde{S}_2$, we get $x \in \phi(S)$ as desired.

6. Satellite information. We now discuss how to associate information with dictionary elements. More specifically, we consider the setting where each element of S has an associated piece of satellite information from some set $V = \{0,\ldots,s-1\}$, where $s = m^{O(1)}$. The information theoretical minimum for this problem is $B_s = \lceil \log \binom{m}{n} + n \log s \rceil$.

The quotienting technique generalizes to this setting. We simply extend the quotient function to take an extra parameter from V as follows: $q'(u,v) = q(u) + r\,v$. Note that from $q'(u,v)$ it is easy to compute $q(u)$ and v and that the range of q' has size $r\,s$. With this new quotient function, the remaining parts of the construction for $m > n\,(\log n)^3$ are unchanged.

In the dense range, the rank dictionary can be used to index into a table of V-values, but in general $\Omega(n)$ bits will be wasted in the table since $|V|$ need not be a power of 2. Thus we have the following theorem.

THEOREM 6.1. *The static dictionary problem with satellite data and worst case constant query time can be solved with storage:*

- $B_s + O(n\,(\log\log n)^2/\log n + \log\log m)$ *bits, for* $m > n\,(\log n)^3$.
- $B_s + O(n)$ *bits, otherwise.*

Using the data structure for the sparse case, it is in fact possible to achieve redundancy $o(n)$ when $n = o(m)$. The dictionary of Proposition 2.2 is then used to store S_2, and parameter a is chosen around $\sqrt{nm}$.

7. Construction. We now sketch how to construct the static dictionaries in expected time $O(n + (\log\log m)^{O(1)})$. The dictionaries of section 4 can in fact be constructed in time $O(n)$ when $m = n^{O(1)}$. The construction algorithm is quite straightforward, so we do not describe it here. As for the dictionary described in sections 3 and 5, the hardest part is finding appropriate parameters for the hash function. Once this is done, the dictionary for $h[S_1]$, the hash table, and the dictionary for S_2 can all be constructed in expected time $O(n)$ (see [7] for the latter construction algorithm).

The prime p is found by randomly choosing numbers from the appropriate interval of Lemma 5.1. Each number chosen is checked for primality (using a probabilistic check which uses expected time polylogarithmic in the number checked [1], that is, time $(\log n + \log\log m)^{O(1)}$). When a prime is found, it is checked whether $u \mapsto u \bmod p$ is 1-1 on S (the element distinctness problem on the residues, taking expected $O(n)$ time using universal hashing). The process is repeated until this is the case. Inspecting the proof of Lemma 5.1 it can be seen that the expected number of iterations is $O(1)$, so the expected total time is $O(n + (\log\log m)^{O(1)})$.

Parameter a is simple to compute according to Lemma 5.2, for example by binary search on the interval in which it is wanted.

Parameter k is tentatively chosen at random and checked in time $O(n)$ for the inequality of Lemma 3.1, with some constant c in the big-oh. For sufficiently large c, the expected number of attempts made before finding a suitable k is constant, and thus the expected time for the choice is $O(n)$.

THEOREM 7.1. *The data structure of Theorem* 1.1 *can be constructed in expected time* $O(n + (\log\log m)^{O(1)})$.

8. Conclusion and final remarks. We have seen that for the static dictionary problem it is possible to come very close to using storage at the information theoretic minimum, while retaining constant query time. From a data compression point of view this means that a sequence of bits can be coded in a number of bits close to the first-order entropy, in a way that allows efficient random access to the original bits.

The important ingredient in the solution is the concept of quotienting. Quotienting was recently applied in a space efficient dictionary supporting rank [14]. In general, quotienting can be used to save around $n \log n$ bits in hash tables. Thus, the existence of an efficiently evaluable quotient function is a desirable property for a hash function. For the quotient function to have a small range, it is necessary that the hash function used hashes U quite evenly to the entire range.

Quotienting works equally well in a dynamic setting, where it can be used directly to obtain an $O(B)$ bit scheme, equaling the result of Brodnik and Munro [2]. However, lower bounds on the time for maintaining ranks under insertions and deletions (see [8]) show that our construction involving the dictionary supporting rank will not dynamize well.

It would be interesting to determine the exact redundancy necessary to allow constant time queries. In particular, it is remarkable that no lower bound is known in the *cell probe* model (where only the number of memory cells accessed is considered). As for upper bounds, a less redundant implementation of the function g would

immediately improve the asymptotic redundancy of our scheme. There seems to be no hope of getting rid of the $O(\log \log m)$ term using our basic approach, since any hash function family ensuring that some function is 1-1 on a "large" subset of S has size $\Omega(\log m)$ [13].

Acknowledgments. The author would like to thank Peter Bro Miltersen for encouragement and advice. Thanks also to Jakob Pagter and Theis Rauhe for feedback on a previous version of this paper [12] and to Kunihiko Sadakane for pointing out an error in a draft of this paper.

REFERENCES

[1] L. M. Adleman and M.-D. Huang, *Recognizing primes in random polynomial time*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC '87), ACM Press, New York, 1987, pp. 462–469.

[2] A. Brodnik and J. I. Munro, *Membership in constant time and minimum space*, in Proceedings of the 2nd European Symposium on Algorithms (ESA '94), Lecture Notes in Comput. Sci. 855, Springer-Verlag, Berlin, 1994, pp. 72–81.

[3] A. Brodnik and J. I. Munro, *Membership in constant time and almost-minimum space*, SIAM J. Comput., 28 (1999), pp. 1627–1640.

[4] L. Carter, R. Floyd, J. Gill, G. Markowsky, and M. Wegman, *Exact and approximate membership testers*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC '78), ACM Press, New York, 1978, pp. 59–65.

[5] A. Fiat and M. Naor, *Implicit $O(1)$ probe search*, SIAM J. Comput., 22 (1993), pp. 1–10.

[6] F. Fich and P. B. Miltersen, *Tables should be sorted (on random access machines)*, in Proceedings of the the 4th Workshop on Algorithms and Data Structures (WADS '95), Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, 1995, pp. 482–493.

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

[8] M. L. Fredman and M. E. Saks, *The cell probe complexity of dynamic data structures*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89), ACM Press, New York, 1989, pp. 345–354.

[9] D. R. Heath-Brown and H. Iwaniec, *On the difference between consecutive primes*, Invent. Math., 55 (1979), pp. 49–69.

[10] K. Mehlhorn, *Data Structures and Algorithms.* 1*, Sorting and Searching*, Springer-Verlag, Berlin, 1984.

[11] P. B. Miltersen, *Lower bounds for static dictionaries on RAMs with bit operations but no multiplication*, in Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP '96), Lecture Notes in Comput. Sci. 1099, Springer-Verlag, Berlin, 1996, pp. 442–453.

[12] R. Pagh, *Low redundancy in static dictionaries with $O(1)$ lookup time*, in Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP '99), Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 595–604.

[13] R. Pagh, *Dispersing hash functions*, in Proceedings of the 4th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '00), Proceedings in Informatics 8, Carleton Scientific, Waterloo, ON, Canada, 2000, pp. 53–67.

[14] V. Raman and S. S. Rao, *Static dictionaries supporting rank*, in Proceedings of the 10th International Symposium on Algorithms And Computation (ISAAC '99), Lecture Notes in Comput. Sci. 1741, Springer-Verlag, Berlin, 1999, pp. 18–26.

[15] J. P. Schmidt and A. Siegel, *The spatial complexity of oblivious k-probe hash functions*, SIAM J. Comput., 19 (1990), pp. 775–786.

[16] R. E. Tarjan and A. C.-C. Yao, *Storing a sparse table*, Communications of the ACM, 22 (1979), pp. 606–611.

[17] A. C.-C. Yao, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615–628.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 364–374

MAINTAINING MINIMUM SPANNING FORESTS IN DYNAMIC GRAPHS*

MONIKA R. HENZINGER† AND VALERIE KING‡

Abstract. We present the first fully dynamic algorithm for maintaining a minimum spanning forest in time $o(\sqrt{n})$ per operation. To be precise, the algorithm uses $O(n^{1/3}\log n)$ amortized time per update operation. The algorithm is fairly simple and deterministic. An immediate consequence is the first fully dynamic deterministic algorithm for maintaining connectivity and bipartiteness in amortized time $O(n^{1/3}\log n)$ per update, with $O(1)$ worst case time per query.

Key words. dynamic graph, graph algorithm, minimum spanning tree, data structure

AMS subject classifications. 05C85, 68W40, 68Q25, 68P05

PII. S0097539797327209

1. Introduction.

We consider the problem of maintaining a minimum spanning forest during an arbitrary sequence of edge insertions and deletions. Given an n-vertex graph G with edge weights, the *fully dynamic minimum spanning forest problem* is to maintain a minimum spanning forest F under an arbitrary sequence of the following update operations:

insert(u,v). Add the edge $\{u,v\}$ to G. Add $\{u,v\}$ to F if it connects two previously unconnected trees of F or if it reduces the cost of F. If the latter, return the edge of F that has been replaced.

delete(u,v). Remove the edge $\{u,v\}$ from G. If $\{u,v\} \in F$, then (a) remove $\{u,v\}$ from F and (b) return the minimum-cost edge e of $G \setminus F$ that reconnects F if e exists or return *null* if e does not exist.

In addition, the data structure permits the following type of query:

connected(u,v). Determine if vertices u and v are connected.

In 1985 [7], Fredrickson introduced a data structure known as *topology trees* for the fully dynamic minimum spanning forest problem with a worst-case cost of $O(\sqrt{m})$ per update His data structure permitted connectivity queries to be answered in $O(1)$ time. In 1992, Eppstein, Galil, and Italiano [3] and Eppstein et al. [4] improved the update time to $O(\sqrt{n})$ using the *sparsification technique*. If only edge insertions are allowed, the Sleator–Tarjan dynamic tree data structure [13] can be used to maintain the minimum spanning forest in time $O(\log n)$ per insertion or query. If only edge deletions are allowed ("deletions-only"), then no algorithm faster than the $\Omega(\sqrt{n})$ fully dynamic algorithm is known.

Using randomization, it was recently shown that the fully dynamic connectivity problem, i.e., the restricted problem where all edge costs are the same, can be solved in amortized time $O(\log^2 n)$ per update and $O(\log n)$ per connectivity query [9, 10]. However, this approach could not be extended to arbitrary edge weights, leaving the question open as to whether the fully dynamic minimum spanning forest problem can be solved in time $o(\sqrt{n})$.

*Received by the editors September 16, 1997; accepted for publication (in revised form) December 19, 2000; published electronically August 8, 2001. A preliminary version of this work appeared in ICALP '97.

http://www.siam.org/journals/sicomp/31-2/32720.html

†Google, Inc., Mountain View, CA (monika@google.com).

‡Department of Computer Science, University of Victoria, Victoria, BC, Canada V8W 3P4 (val@csr.uvic.ca). The research of this author was supported by an NSERC grant.

In this paper we give a positive answer to this question: We present a fully dynamic minimum spanning forest data structure that uses $O(n^{1/3}\log n)$ amortized time per update and $O(1)$ worst case time per query when update time is averaged over any sequence of $\Omega(m_{in})$ updates for m_{in} the initial size of the graph. Our technique is very different from [7].

The result is achieved in two steps: First, we give a deletions-only minimum spanning forest algorithm that uses $O(m'^{1/3}\log n + n^{\epsilon})$ amortized time per update and $O(1)$ worst case time per query when the update time is averaged over any sequence of $\Omega(m_{in})$ updates. Here ϵ is any constant such that $0 < \epsilon < 1/3$, and m' is the number of *nontree* edges at the time of the update.

Then we present a general technique which, given a deletions-only minimum spanning forest data structure with a certain property, generates a fully dynamic data structure with the same running time as the deletions-only data structure. Let $f(m', n)$ be the amortized time per deletion in the deletions-only data structure with m' nontree edges and n vertices. The property required is that, upon inserting into the graph no more than m' edges at the same time (a "batch insertion"), the deletions-only data structure can be modified to reflect these insertions and up to m' subsequent deletions can be performed in a total of $O(m'f(m', n))$ time.

Using this technique, we develop a fully dynamic minimum spanning forest algorithm with *amortized* time per update of $O(m^{1/3}\log n)$ for a sequence of updates of length $\Omega(m_{in})$, where m is the size of G at the time of the update. In other words, letting $m_{(i)}$ denote the size of G (vertices and edges) after update i, the total amount of work for processing a sequence of updates of length l is $O(\sum_{i=0}^{l} {m_{(i)}}^{1/3}\log n)$. We then apply sparsification [3, 4] to reduce the running time for the sequence to $O(ln^{1/3}\log n)$.

Our result immediately gives faster *deterministic* fully dynamic algorithms for the following problems: connectivity, bipartiteness, k-edge witness, maximal spanning forest decomposition, and Euclidean minimum spanning tree. See [9] for all but the last reduction; see Eppstein [2] for the last reduction. For these problems, the new algorithm achieves an $O(n^{1/6}/\log n)$ factor improvement over the previously best *deterministic* running time. If randomization is allowed, however, much faster times are achievable [9, 10].

Additionally, improvements can be achieved in the following static problems (see [4, 3]): randomly sampling spanning forests of a given graph [6] and finding a color-constrained minimum spanning tree [8].

The paper is structured as follows: In section 2 we give a deletions-only minimum spanning forest algorithm. In section 3, we show how to use a sequence of deletions-only data structures to create a fully dynamic data structure.

2. Maintaining a minimum spanning forest—Deletions-only. In this section, we give an algorithm which maintains a minimum spanning forest while edges are being deleted. The amortized update time is $O(m^{1/3}\log n)$ and the query time is $O(1)$ for queries of the form "Are vertices i and j connected?". Let $G = (V, E)$ be an undirected graph with edge weights. Without loss of generality, we assume that edge weights are distinct.

Initially, we compute the minimum spanning forest F of G. Let m'_{in} be the number of nontree edges in G initially and $k = {m'}_{in}^{1/3}\log n$. We sort the nontree edges by weight and partition them into ${m'}_{in}/k$ levels of size k so that the k lightest are in level 0, the next k lightest are in level 1 and so on. The set of edges in a level i is denoted by E_i. In addition, all tree edges of the initial minimum spanning forest F

are placed in level 0. (We omit floors and ceilings to simplify notation; either can be used without affecting the asympotic analysis.)

Throughout the algorithm, the level of an edge remains unchanged, and F denotes the minimum spanning forest. For $i = 0, 1, \ldots, (m'_{in}/k) - 1$, let F_i denote the minimum spanning forest of the graph with vertex set V and edgeset $\cup_{j \le i} E_j$. (Initially, all $F_i = F$, but in later stages, an edge from any level may become a tree edge. Thus, $F_0 \subseteq F_1 \subseteq \ldots \subseteq F_{(m'_0/k)-1} = F$.) Let $T_i(x)$ denote the tree in F_i which contains x and let $T(x)$ without the subscript denote the tree in F containing x.

The main idea is the following. If a nontree edge is deleted, then the minimum spanning forest F is unchanged. Suppose a tree edge $\{u, v\}$ in level i is deleted. Then for each F_j, $j \ge i$, the deletion splits the tree in F_j containing u and v into $T_j(u)$ and $T_j(v)$. We search for the minimum weight nontree edge e (called the "replacement edge") that connects $T(u)$ and $T(v)$ by gathering and then testing a set S of candidate edges on level i. If none is found, we repeat the procedure on level $i + 1$, etc., until one is found or all levels are exhausted.

We now describe the update operations.

delete(u, v). Delete edge $\{u, v\}$ from any data structures in which it occurs. If a tree edge $\{u, v\}$ from level i is deleted, then remove $\{u, v\}$ from F and search for a replacement by calling **Replace**(i, u, v). We refer to i as the level of the call to **Replace**.

In the algorithm below, the subroutine **Search** when applied to a tree in F_i finds all nontree edges in level i which are incident to the tree. A phase consists of the examination of a single edge. (Its exact definition and the details of **Search** are given in section 2.2 below.)

Replace(i, u, v).

1. Alternating in lockstep, one phase at a time, **Search**$(T_i(u))$ and **Search**$(T_i(v))$ until $k/\log n$ phases are executed (Case A) or one of the searches has stopped (Case B).
   - Case A: Let S be the set of all nontree edges in level i.
   - Case B: Let S be the set of (nontree) edges produced by the **Search** that stopped.
2. Test every edge in S to see if it connects $T(u)$ and $T(v)$.
   - If a connecting edge is found, insert the minimum weight connecting edge into F and the data structures representing the F_j, $j \ge i$.
   - Else if i is not the last level, call **Replace**$(i + 1, u, v)$.

2.1. Data structures. The idea here is to use the ET-tree data structure developed in [9]: (1) to represent and update each tree in F, so that in constant time, we can quickly test if a given edge joins two trees; and (2) to represent each tree in an F_i in such a way that we can quickly retrieve nontree edges in E_i which are incident to the tree. To avoid excessive cost, we explicitly maintain only those F_i where i is a multiple of $m'^{1/3}_{in}/\log n$. An undesirable consequence of this is that when retrieving nontree edges in E_i, other nontree edges are also retrieved.

Below, we refer to input graph vertices as "vertices" and use "node" to mean nodes of the B-tree in which we store the "ET-sequences."

ET-trees. An *ET-sequence* is a sequence generated from a tree by listing each vertex each time it is encountered ("an occurrence of the vertex") as a tree is searched depth-first. Each ET-sequence is stored in a B-tree of degree d. This allows us to implement the deletion or insertion of an edge in the forest as follows: we split a tree by deleting an edge or join two trees by inserting an edge in time $O(d \log_d n)$, using

a constant number of splits and joins on the corresponding B-trees. Also we can test two vertices of the forest to determine whether they are in the same tree in time $O(\log_d n)$, by searching up to the roots and testing for equality. See, for example, [1, 11] for operations on B-trees. If $d = n^\alpha$, for α a positive constant, then the join and split operations take time $O(d)$ and the test operation takes time $O(1)$. We refer to the B-trees used to store ET-sequences as ET-trees.

This data structure allows us to keep information about a vertex so that the cumulative information about all vertices in a tree can be maintained. To do so, we make one arbitrary occurrence of each vertex a "designated" occurrence and, in each internal node of the ET-tree, we keep cumulative information about the designated occurrences in the subtree of the ET-tree rooted at that node. Here, for example, we want to know if there is at least one nontree edge incident to a vertex in a tree. We mark the vertex's designated occurrence in the ET-tree if there is a nontree edge in the graph incident to that vertex. We mark each internal node of the ET-tree if some node in its subtree is marked. We can find the endpoint of a nontree edge by starting at the root of the ET-tree and following a path of marked nodes down to a marked designated occurrence. In a degree d ET-tree, each split or join operation or each change to the number associated with an occurrence requires the adjustment of $O(\log_d n)$ internal nodes with each adjustment taking $O(d)$ timesteps. For other applications of ET-trees, see [9].

We maintain the following data structures.

- Each edge is labelled by its level and a bit which indicates if it is a tree edge.
- Let $d_F = \max\{{m'_{in}}^{1/3} \log n, n^\epsilon\}$, for any constant $0 < \epsilon \leq 1/3$. Each tree in F is represented as an ET-sequence which is stored in a degree d_F B-tree. Note that these B-trees have a constant number of levels, so that two vertices can be tested to determine if they lie in the same tree of F in $O(1)$ time.
- Let $c = {m'_{in}}^{1/3} / \log n$. We partition consecutive levels into classes of size c. Each class is represented by the smallest level in the class, i.e., the level j such that $c|j$ ("c divides j"). That is, level i is mapped to the class $f(i) = c\lfloor i/c \rfloor$. For each representative level j,
  - we represent each tree in F_j as an ET-sequence which is stored in a binary B-tree;
  - for each vertex v, we create a list $L_j(v)$ which contains
    (i) all nontree edges incident to v which are in any level $i \in f^{-1}(j)$ and
    (ii) all tree edges incident to v which are in any level $i > j, i \in f^{-1}(j)$;
  - we mark each designated occurrence of a vertex v whose list $L_j(v)$ is nonempty. Each internal node of the ET-tree is marked if its subtree contains a marked occurrence.

2.2. The Search routine. **Search**$(T_i(u))$ returns all nontree edges in level i incident to $T_i(u)$. It begins by searching $T_{f(i)}(u)$ which is a subtree of $T_i(u)$. It proceeds by examining all edges in $L_{f(i)}(v)$ for all vertices v in the tree being searched. Nontree edges in level i are picked out and tree edges in levels $i', f(i) < i' \leq i$, are followed to other trees of $F_{f(i)}$ which are then searched in turn. Note that all such tree edges lead to other trees of $F_{f(i)}$ which are subtrees of $T_i(u)$; and all subtrees of $T_i(u)$ will be found by this procedure. A *phase* of the algorithm consists of the examination of one edge e in a list L.

Search$(T_i(u))$.

1. $S' \leftarrow \emptyset$;
2. $treelist \leftarrow T_{f(i)}(u)$;

3. Repeat until *treelist* is empty:
   - Remove an ET-tree from the *treelist*.
   - For each marked vertex x in the ET-tree and for each edge $\{x, y\}$ in each $L_{f(i)}(x)$,
     - if $\{x, y\}$ is a nontree edge on level i, add it to the set S' of edges to return;
     - else if $\{x, y\}$ is a tree edge on level l such that $l \leq i$, then add $T_{f(i)}(y)$ to *treelist*.

2.3. Analysis. *Initialization.* We compute the minimum spanning forest F, create the ET-trees for F_j, for each j such that $c|j$, and partition the nontree edges by weight. Recall that m'_{in} is the number of nontree edges in the initial graph. Let t be the number of edges in the initial minimum spanning forest. The creation of all the lists L takes time proportional to the number of nontree edges m'_{in}. The building of ET-trees for F and all F_j such that $c|j$ and the marking of internal nodes takes time proportional to the size of each forest or $O(((m'_{in}/k)/c)t + m'_{in}) = O(m'^{1/3}_{in}t + m'_{in})$.

Deletions of nontree edges. Deleting a nontree edge on any level may require unmarking a designated occurrence of a vertex in some ET-tree, which may require unmarking internal nodes on the path to the root in $O(\log n)$ time.

Deletions and insertions of tree edges. Deleting a tree edge takes worst-case time $O(d_F)$ to delete it from the ET-tree of F and worst-case time $O(\log n)$ to delete it from the ET-tree of each F_j such that $c|j$, for a total of $O(d_F + ((m'_{in}/k)/c)\log n)$ time per edge. Inserting a replacement edge takes the same time.

Finding a replacement edge. We first analyze the cost of **Search**. Let the *weight* $w(T)$ *of a tree* T *of some* F_i be $\sum |L_{f(i)}(v)|$ summed over all vertices v in T. It costs $O(\log n)$ to move down the path from the root to a leaf in an ET-tree to find a marked occurrence of a vertex, or to move up a tree from an occurrence to the root. Thus, the cost of **Search**$(T_i(x))$ is $O(\log n)$ times the number of edges examined, or $O(w(T_i(x))\log n)$, if **Search** is carried out until it ends, and $O(k)$, if it is run for $k/\log n$ phases.

In **Replace**(u, v, i), if $w(T_i(u)) \leq w(T_i(v))$, then we refer to $T_i(u)$ as the *smaller component* T_1; otherwise, T_1 is $T_i(v)$. The cost of a call to **Replace**(u, v, i) is the cost of the **Search** plus the cost of testing each edge in S. The number of edges in S is $O(\min\{k, w(T_1)\})$. We can use the d_F-degree ET-tree representation for F to test each edge at cost $O(1)$. Thus the cost of a call to **Replace** is $O(\min\{k, w(T_1)\log n\})$.

To pay for these costs, If a replacement edge is found on level i then we charge the cost of **Replace**(u, v, i) to the deletion. In addition, we charge the cost of modifying F to the deletion so the total cost charged to the deletion is $O(\min\{k, w(T_1)\log n\} + ((m'_{in}/k)/c)\log n + d_F) = O(((m'_{in}/k)/c)\log n + d_F)$.

If no replacement edge is found on level i then a tree of F_i which was split by the deletion remains split. We use the following.

Claim 2.1. $O(\sum w(T_1))$ *summed over all smaller components* T_1 *which split from a tree* T *on any given level during all* **Replace** *operations is* $O(w(T)\log n)$.

The proof of the claim follows [5]. The first time a smaller component T_1 of a tree T is searched, it can have weight no greater than $w(T)/2$. Between two successive times that $|L_{f(i)}(v)|$ contributes to the weight of a smaller component T_1 and that component splits off, the weight of a smaller component T_1 containing v is no more than half its weight the previous time. Hence $|L_{f(i)}(v)|$ contributes to the weight of any T_1 no more than $\log_2 w(T) = O(\log n)$ times. That is, $O(\sum w(T_1)) = O(\sum_{v\in T} |L_{f(i)}(v)|)\log n) = O(w(T)\log n)$.

There are at most k edges per level (except for level 0, which has at most k nontree edges). Each $L_j(v)$ consists of edges from c levels. Since level 0 tree edges do not belong to any list $L_j(v)$, the maximum weight of a tree $w(T)$ is ck. Thus the total cost charged to a level is $O(ck \log^2 n)$. Summing over all levels we have $O((m'_{in}/k)(ck \log^2 n) = O(m'_{in} c \log^2 n)$, or an amortized cost per deletion of $O(c \log^2 n) = O(m'^{1/3}_{in} \log n)$, if $\Omega(m'_{in})$ edges are deleted.

The cost charged to each deletion is $O((m'_{in}/ck)(\log n) + d_F)$. For $d_F = max\{m'^{1/3}_{in} \log n, n^\epsilon\}$ and $c = m'^{1/3}_{in}/\log n$, this is $O(m'^{1/3}_{in} \log n + n^\epsilon)$.

To summarize the cost of initialization when amortized over $\Omega(m_{in})$ operations is $O(m'^{1/3}_{in})$ and the cost per deletion of an edge and finding replacement edges, when amortized over $\Omega(m'_{in})$ operations is $O(m'^{1/3}_{in} \log n + n^\epsilon)$. Thus for a sequence of $\Omega(m_{in})$ operations, the amortized time per update is $O(m'^{1/3}_{in} \log n + n^\epsilon)$.

Finally, we note that the query of the form "Are vertices i and j connected?" can be answered using the ET-tree data structure for F in $O(1)$ time.

3. From deletions-only to fully dynamic. In this section, we show a general technique to develop a fully dynamic data structure using several deletions-only data structures with an added operation. (We call these "extended" deletions-only data structures.) As before, we assume the edge weights are distinct.

First, we define the following operation on a deletions-only data structure A.

batch_add(G, E', F'): Given a graph $G = (V, E)$ with minimum spanning forest F, insert all edges of E' into G, if they are not already there. The resulting spanning forest F' is given.

We refer to the period of time which occurs between two consecutive calls to *batch_add* on a graph G, or between the start of the algorithm and the first *batch_add* on G as a *period of* G. Alternatively, a period may be terminated prematurely (see below).

We prove the following theorem.

THEOREM 3.1. *Suppose for any value of* n *and* m'_{in}*, there is an extended deletions-only data structure for any dynamic graph* $G = (V, E)$ *with* $|V| = n$ *and the number of nontree edges in the edgeset* E *is initially* m'_{in}*, such that* $(n + m'_{in})f^0(m'_{in}, n)$ *is the worst case time needed to initialize* A*, and* $(y + m'_{in})f(m'_{in}, n)$ *is an upper bound on the time to process* y *deletions.*

Suppose we can process a batch_add(H, E', F')*, following any period in which* y *edges were deleted from* G*, in time* $O((y + m'_{in} + |F' \setminus F|)f^B(m'_{in}, n))$*, where* m'_{in} *is an upper bound on the total number of nontree edges in* G *after the batch_add.*

We also assume that f^0*,* f*,* f^B *are monotone nondecreasing functions.*

For any value d *there is a fully dynamic minimum spanning forest data structure that runs in amortized cost per edge deletion or insertion of* $O(d \log_d n + \sum_{i=0}^{s}(s - i + 1)[f^0(2^i, n) + f(2^i, n) + f^B(2^i, n)])$ *where* $s \leq 3 + \lg m$ *and* m *is the size (vertices plus edges) of the dynamic graph at the time of the update. Here, costs are amortized over a sequence of* m_{in} *update operations, where* m_{in} *is the size of the initial graph.*

In section 3.4, we show that the following corollary holds by choosing $d = n^\epsilon$ for any constant $1/3 \geq \epsilon > 0$ and using the data structure of the previous section.

COROLLARY 3.2. *A minimum spanning forest can be maintained in a fully dynamic graph with amortized cost per update of* $O(m^{1/3} \log n)$*, where* m *is the size of the graph at the time of the update, for a sequence of* $\Omega(m_{in})$ *operations.*

Noting that the function $f(m, n) = O(m^{1/3} \log n)$ is "well behaved" in the sense of Theorem 3.3.2 of [4] gives the main result of the paper.

COROLLARY 3.3. *A minimum spanning forest can be maintained in a fully dynamic graph with amortized cost per update of* $O(n^{1/3} \log n)$*, where* n *is the number of vertices in the graph at the time of the update, for a sequence of* $\Omega(m_{in})$ *operations.*

We prove our theorem by constructing a fully dynamic data structure from extended deletions-only data structures.

Definitions. We refer to the current minimum spanning forest of G as the (global) MSF. Let m' be the number of nontree edges in the current graph, m'_{in} denote the number of nontree edges in the initial graph, and m denote the current size (vertices and edges) of G.

During the course of the algorithm, we simultaneously maintain up to $s \leq \max\{\lg n, \lg(4m')\}$ extended deletions-only data structures $A_0, A_1, \ldots, A_s$, where each A_i is an extended deletions-only minimum spanning forest data structure for a subgraph $G^i + (V, E^i)$ of the global graph $G = (V, E)$. We call this the *composite data structure.* We maintain each tree of the MSF as a Sleator–Tarjan dynamic tree [13] and also as an ET-tree of degree d.

The minimum spanning forest of G^i as maintained by A_i is referred to as a *local spanning forest* and denoted F^i. A *local nontree edge* of A_i is an edge of G^i which is not in A_i's local spanning forest or the MSF. We maintain x_i to be the number of local nontree edges in $\cup_{j \leq i} A_j$.

When m' falls below $2^s/4$ and $s > \lg n$, s is *reset* and the composite data structure is reinitialized. Between two consecutive resets, we define the period of time which occurs between two consecutive calls to *batch_add* on a graph G, or between the initialization or reinitialization of the composite data structure and the first *batch_add* on G as a *period of* G. A reset terminates all periods.

The *size* of a graph refers to the number of vertices plus edges.

We maintain the following invariants.

Invariant. (1) Every edge in the local forest of some A_i is (a) in the MSF or (b) is a local nontree edge in some $A_j, j \neq i$.

(2) $E = (\cup E^i) \cup MSF$.

We now describe the algorithm.

To initialize, Let the initial value of $s = \lceil \lg m'_{in} \rceil$. We initialize A_s as an extended deletions-only data structure for $G^s = G$ with $F^s = MSF$ and the set of local nontree edges being all nontree edges of G.

To perform an insertion operation, **insert**(u, v) is called, where (u, v) is an edge to be inserted into G.

insert(e):

1. Use the dynamic tree to determine if e should be added to the MSF:
   Determine if there is a path between e's endpoints in the MSF. If so, set f to the maximum weight edge on the path.
2. If e is lighter than f, remove f from the MSF.
   If there is no path between e's endpoints, either because f has been removed or there was none previously, call **insert_nontree**(f), and add e to the MSF.
3. Else call **insert_nontree**(e).

The following subroutine inserts a nontree edge e into the composite data structure.

insert_nontree(e). Let $i = \min_{\{} j \mid x_j < 2^j\}$. Let E' be the set of local nontree edges in $\cup_{j<i} E^j \cup \{e\}$.

1. Delete the edges of E' from $A_j, j < i$.
   Set $x_j = 0$.

2. If A_i is not initialized, initialize A_i on the empty graph G^i consisting of n vertices and no edges.
3. Call $batch\_add(G^i, E' \cup MSF, MSF)$.
   Adjust x_i accordingly.

After the procedure, the local nontree edges of G^i are the nontree edges previously contained in $\cup_{j \le i} A_j$. Its local forest $F^i = MSF$. Note that at the beginning of a period of G^i, $x_j = 0$ for $j < i$.

To delete an edge e from G:

delete(e):

1. Delete e from all data structures in which it appears, including all G^i, and update corresponding A_i accordingly. Thus for each local spanning forest F^i which contained e, the local replacement edge e' is determined, if there is one.
2. If e was in the MSF, use the ET-tree representation of the tree containing e in the MSF to determine which of those local replacement edges reconnect the two subtrees which result from the deletion of e. Insert the lightest connecting edge into the MSF.
3. All other local replacement edges are reinserted into the composite data structure using the procedure *insert_nontree*.
4. If $n < x_s \le 2^{s-2}$, reinitialize the composite data structure. That is, set $s = \lceil \lg x_s \rceil$; initialize A_s as an extended deletions-only data structure for $G^s = G$ with $F^s = MSF$ and the set of local nontree edges being all nontree edges of G.

3.1. Proof of correctness. It is easy to see that the invariants are maintained, by induction on the number of operations. Initially, the invariants hold since $G^s = G$. Invariant (2) is preserved after each insertion, since each edge when added to G is either added to the MSF or some G^i. Each edge, when deleted from G, is deleted from all data structures in which it appears. Invariant (1) holds for A_i when A_i is initialized or a *batch_add* is executed since the local forest $F_i = MSF$. The local forest of A_i changes only when an edge is deleted and is replaced by some edge e. Edge e is then either put into the MSF or reinserted into the composite data structure. In that case, it is added to some A_j by a *batch_add* operation. If e is not in the MSF, then e becomes a local nontree edge of A_j. In either case, invariant (1) is preserved.

The correctness of the algorithm follows easily from the invariants. We use the well-known fact that an edge is in the minimum spanning tree iff it is not the heaviest edge in any cycle ("red rule" [14]). We also note that every edge in the composite data structure is an edge in G.

Let e be an edge of the MSF which is deleted. Let e' be the correct replacement edge. Consider the state of the composite data structures right before the deletion of e. By the invariant, since e' was not in the MSF, it was a local nontree edge in some A_i.

Suppose e' is a local nontree edge in A_i. Since e' is the correct replacement edge for e in the MSF, then after e's deletion, e' is not the heaviest edge in any cycle of G and therefore is not the heaviest edge of any cycle of G^i. Hence, after e's deletion, e' becomes a local forest edge, i.e., e' is a local replacement edge for e in G^i. Recall that e' is the minimum weight edge which connects the two subtrees of the MSF resulting from the deletion of e. Thus, e' is the lightest connecting edge from the set of local replacement edges and is chosen in Step 2 of the **delete** algorithm.

3.2. Analysis. We first prove the following claims.

CLAIM 3.4. *During any full period of G^i, there were at least $2^{i-1}/(s-i+1)$ updates to G.*

Proof. For $i > 0$, immediately before *batch_add* is executed on G^i, $x_{i-1} \geq 2^{i-1}$. Immediately afterwards, $x_{i-1} = 0$.

We examine the types of insertions into the composite data structure to see how they affect x_i: (a) when a nontree edge is inserted into G, (b) when an edge is replaced in the MSF after an insertion, and (c) when an edge is deleted in G and it is replaced in up to s local spanning forests. The first two cases cause x_i to increase by no more than one. The third case may cause up to s insertions. However, the s insertions do not affect all A_i the same. Each insertion in this case results from a local nontree edge e becoming a local forest edge. Hence if this occurs in some $A_j, j \leq i$, the increase of x_i resulting from the insertion of a copy of e into the composite data structure is offset by the decrease of x_i caused by the change in status of e from a local nontree edge to a local tree edge. Thus x_s is unchanged by a case-(c) insertion into the composite data structure, x_{s-1} is increased by at most 1, and in general, x_i is increased by at most $s-i$.

Hence, at least $2^{i-1}/(s-i+1)$ insertions or deletions occurred during any full period of G^i. This concludes the proof of the claim.

We are now ready to analyze the costs of the algorithm.

Initialization. Since each A_i is initialized once, the cost for initialization during the algorithm is $(n+2^i)f^0(2^i,n)$. Note that A_i is initialized only if the number of nontree edges exceed 2^{i-1}.

We amortize the initialization costs of the first data structure A_s and all A_i for $i < \lg n$ by requiring there to be $\Omega(m_{in})$ operations, where m_{in} is the size (vertices and edges) of the initial graph. We note that for at least half these operations, the current size of the graph $m \geq m_{in}/2$.

We amortize the cost of initializing A_i, $i \geq \lg n$ over the operations of the preceding period when at least $2^{i-1}/(s-i+1)$ operations occurred.

The cost of *reinitialization* of the composite data structure can be charged to the $2^s/2$ deletions which must have occurred since the previous reset. Note that a reset only occurs when $2^s > n$, so that the initialization cost of $(n+2^{s-2})f^0(2^{s-2},n)$ results in a charge of $f^0(2^{s-2},n)$ per operation.

Execution of batch_add. By assumption, $(y+2^i+|MSF \setminus F^i|)f^B(2^i,n)$ is an upper bound on the time to perform *batch_add*$(G^i, E' \cup MSF \setminus F^i, MSF)$, where y is the number of deletions performed on G^i in the preceding period.

We can charge the cost of $yf^B(2^i,n)$ to the y deletions for a cost of $f^B(2^i,n)$ each.

To charge the $2^i f^B(2^i,n)$, by the claim, *batch_add* is called on G^i after at least $2^{i-1}/(s-i+1)$ insertions and deletions occurred in the preceding period. Charging the $2^i f^B(2^i,n)$ to those updates gives a cost per update of $(s-i+1)f^B(2^i,n)$.

To charge the $|MSF \setminus F^i| f^B(2^i,n)$, we note that at the start of the period, $F^i = MSF$, and for each i, each insertion or deletion in G can cause at most one edge to be added to and/or one edge to be deleted from F^i. Thus we can charge the $|MSF \setminus F^i| f^B(2^i,n)$ to the operations in the preceding period, for a cost of $O(f^B(2^i,n))$ each.

Performing deletions during a period of G^i. The cost of maintaining A_i during a period containing y deletions of edges in G^i is, by assumption, bounded above by $(y+2^i)f(2^i,n)$. These costs can be charged in the same way as the costs for *batch_add*

were charged to the operations of the preceding period.

In the unique case of the initial A_s, where $s = \lceil \lg m'_{in} \rceil$ when there was no preceding period, costs are amortized over the initial sequence of $\Omega(m_{in})$ deletions and insertions, as in the analysis of the initialization costs.

After a reset of s, the cost of performing deletions in A_s, after A_s is reinitialized, is charged to the deletions which resulted in the reset, as in the analysis of the reinitialization costs.

Summary. For each i, the cost per operation is therefore $O((s-i+1)[f^0(2^i,n) + f(2^i,n) + f^B(2^i,n)])$.

Except for the initialization and reinitialization of A_s, we have charged operations of the preceding period for all costs incurred in the following period. Since the preceding periods occur in between resets of the value of s, we know that for the indices of the A_i, $i \leq s \leq max\{\lg 4m', \lg n\}$. Hence $s \leq 2 + \lg m$, m being the size of the graph at the time of the operation.

For the initialization and reinitialization of A_s, we charge operations which occurred when s may have been smaller by 1. Hence $s \leq 3 + \lg m$, m is the size of the graph at the time of the operation.

Each operation requires a constant number of updates in the dynamic tree data structure and the degree-d ET-tree data structure storing the MSF. This takes time $O(d \log_d n)$.

Summing over i, we have of $O(d \log_d n + \sum_{i=0}^{s}(s-i+1)[f^0(2^i,n) + f(2^i,n) + f^B(2^i,n)]$, where m is the current size of G at the time of the operation, when amortized over a sequence of m_{in} update operations and m_{in} is the size of the initial graph.

3.3. Implementing *batch_add*. In this section, we show how a deletions-only data structure A in section 2 for a graph G which initially had m'_{in} nontree edges can be extended so that the operation *batch_add* which occurs after a sequence σ of y edge deletions can be implemented in time $O((y + m'_{in} + |F' \setminus F|)({m'_{in}}^{1/3} \log n))$.

We begin by restoring the ET-trees of A to MSF_{old}, the minimum spanning forest of G before the start of the sequence σ of deletions. The cost of joining two ET-trees is asympotically the same as splitting them; thus the calculations of section 2 apply. For each deletion, the cost of restoration is $O(({m'}_{in}^{1/3}) \log n + n^{\epsilon})$.

We next transform MSF_{old} to MSF by again modifying the ET-trees. We remove every edge in $MSF_{old} \setminus MSF$ and insert every edge in $MSF \setminus MSF_{old}$ for a cost of $O({m'}_{in}^{1/3} \log n + n^{\epsilon})$.

To determine the transformations required, we keep a list of sorted changes which occurred since the last *batch_add*.

We remove all nontree edges which are stored in A and sort the nontree edges of $E \cup E'$, assign them to levels, and store them with the appropriate list L. The cost per edge of removing, sorting, and then storing is $O(\log n)$ per edge for the unique (binary) ET-tree in which the edge is stored.

Let $f'(m,n) = m^{1/3} \log n + n^{\epsilon}$. We have shown an extended deletions-only data structure such that $O((n + m'_{in})f'(m_{in},n))$ is an upper bound on the worst case time needed to initialize A, and $O((y + m'_{in})f'(m_{in},n))$ is an upper bound on the time to process y deletions.

We can process a *batch_add*(G, E', F'), following any period in which y edges were deleted from G, in time $O((y + m'_{in} + |F' \setminus F|)f'(m'_{in},n))$, where m'_{in} is an upper bound on the total number of nontree edges in G after the *batch_add*.

3.4. Proof of corollary. Choose $d = n^\epsilon$ for any constant ϵ with $0 < \epsilon \leq 1/3$. Substituting f' for f^B, f, and f^0, we conclude that there is a fully dynamic minimum spanning forest data structure that runs in amortized cost per edge deletion or insertion of $O(\sum_{i=0}^{s}(s-i+1)f'(2^i,n))$, where $s \leq 3 + \lg m$.

Substituting for f' and 2^i, we have $O(\sum_{i=0}^{s}(i+1)(2^{s-i})^{1/3}\log n + n^\epsilon) = O((2^s)^{1/3}\log n + n^\epsilon \log n) = O(m^{1/3}\log n + n^{\epsilon'})$ for ϵ' any constant.

REFERENCES

[1] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1989, pp. 381–399.

[2] D. Eppstein, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, Discrete. Comput. Geom., 13 (1995), pp. 111–122.

[3] D. Eppstein, Z. Galil, and G. F. Italiano, *Improved Sparsification*, Tech. Report 93-20, Department of Information and Computer Science, University of California, Irvine, CA.

[4] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—A technique for speeding up dynamic graph algorithms*, J. ACM, 44 (1997), pp. 669–696.

[5] S. Even and Y. Shiloach, *An on-line edge-deletion problem*, J. ACM, 28 (1981), pp. 1–4.

[6] T. Feder and M. Mihail, *Balanced matroids*, in Proceedings of the 24th ACM Symposium on Theory of Computing, ACM Press, New York, 1992, pp. 26–38.

[7] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees*, SIAM J. Comput., 14 (1985), pp. 781–798.

[8] G. N. Frederickson and M. A. Srinivas, *Algorithms and data structures for an expanded family of matroid intersection problems*, SIAM J. Comput., 18 (1989), pp. 112–138.

[9] M. R. Henzinger and V. King, *Randomized dynamic graph algorithms with polylogarithmic time per operation*, J. ACM, 46 (1999), pp. 502–516.

[10] M. R. Henzinger and M. Thorup, *Improved sampling with applications to dynamic graph algorithms*, in Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Comput. Sci. 1099, Springer-Verlag, New York, 1996, pp. 290–299.

[11] K. Mehlhorn, *Data Structures and Algorithms* 1*: Sorting and Searching,* Springer-Verlag, 1984.

[12] H. Nagamochi and T. Ibaraki, *Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[13] D. D. Sleator, R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 24 (1983), pp. 362–381.

[14] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983, p. 71.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 375–397

EVOLUTIONARY TREES CAN BE LEARNED IN POLYNOMIAL TIME IN THE TWO-STATE GENERAL MARKOV MODEL*

MARY CRYAN†, LESLIE ANN GOLDBERG‡, AND PAUL W. GOLDBERG‡

Abstract. The j-state general Markov model of evolution (due to Steel) is a stochastic model concerned with the evolution of strings over an alphabet of size j. In particular, the two-state general Markov model of evolution generalizes the well-known Cavender–Farris–Neyman model of evolution by removing the *symmetry* restriction (which requires that the probability that a "0" turns into a "1" along an edge is the same as the probability that a "1" turns into a "0" along the edge). Farach and Kannan showed how to probably approximately correct (PAC)-learn Markov evolutionary trees in the Cavender–Farris–Neyman model provided that the target tree satisfies the additional restriction that all pairs of leaves have a sufficiently high probability of being the same. We show how to remove both restrictions and thereby obtain the first polynomial-time PAC-learning algorithm (in the sense of Kearns et al. [*Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, 1994, pp. 273–282]) for the general class of two-state Markov evolutionary trees.

Key words. computational learning theory, evolutionary trees, PAC-learning, learning of distributions, Markov model

AMS subject classifications. 68Q32, 68W01

PII. S0097539798342496

1. Introduction.

The j-state general Markov model of evolution was proposed by Steel in 1994 [14]. The model is concerned with the evolution of strings (such as DNA strings) over an alphabet of size j. The model can be described as follows. A *j-state Markov evolutionary tree* consists of a *topology* (a rooted tree, with edges directed away from the root), together with the following parameters. The root of the tree is associated with j probabilities $\rho_0, \ldots, \rho_{j-1}$ which sum to 1, and each edge of the tree is associated with a stochastic transition matrix whose state space is the alphabet. A probabilistic experiment can be performed using the Markov evolutionary tree as follows: The root is assigned a letter from the alphabet according to the probabilities $\rho_0, \ldots, \rho_{j-1}$. (Letter i is chosen with probability ρ_i.) Then the letter propagates down the edges of the tree. As the letter passes through each edge, it undergoes a probabilistic transition according to the transition matrix associated with the edge. The result is a string of length n which is the concatenation of the letters obtained at the n leaves of the tree. A j-state Markov evolutionary tree thus defines a probability distribution on length-n strings over an alphabet of size j. (The probabilistic experiment described above produces a single sample from the distribution.[1])

*Received by the editors July 28, 1998; accepted for publication December 10, 2000; published electronically August 8, 2001. This was previously Research Report RR347, Department of Computer Science, University of Warwick, Coventry, UK. A preliminary version of this paper appears in the proceedings of FOCS '98. This work was partially supported by the ESPRIT Projects ALCOM-IT (Project 20244) and RAND-II (Project 21726) and by EPSRC grant GR/L60982.

http://www.siam.org/journals/sicomp/31-2/34249.html

†BRICS, Basic Research in Computer Science, Department of Computer Science, University of Aarhus, Aarhus, Denmark (maryc@brics.dk). The research was carried out while this author was a Ph.D. student at the University of Warwick.

‡Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (leslie@dcs.warwick.ac.uk, pwg@dcs.warwick.ac.uk).

[1]Biologists would view the n leaves as being existing species, and the internal nodes as being hypothetical ancestral species. Under the model, a single experiment as described above would produce a single bit position of (for example) DNA for all of the n species.

To avoid getting bogged down in detail, we work with a binary alphabet. Thus, we will consider *two-state* Markov evolutionary trees.

Following Farach and Kannan [9], Erdös et al. [7, 8], and Ambainis et al. [2], we are interested in the problem of learning a Markov evolutionary tree, given samples from its output distribution. Following Farach and Kannan and Ambainis et al., we consider the problem of using polynomially many samples from a Markov evolutionary tree M to "learn" a Markov evolutionary tree M' whose distribution is close to that of M. We use the *variation distance* metric to measure the distance between two distributions, $\mathcal{D}$ and $\mathcal{D}'$, on strings of length n. The variation distance between $\mathcal{D}$ and $\mathcal{D}'$ is $\sum_{s\in\{0,1\}^n} |\mathcal{D}(s) - \mathcal{D}'(s)|$. If M and M' are n-leaf Markov evolutionary trees, we use the notation $\mathrm{var}(M, M')$ to denote the variation distance between the distribution of M and the distribution of M'.

We use the probably approximately correct (PAC) distribution learning model of Kearns et al. [11]. Our main result is the first polynomial-time PAC-learning algorithm for the class of two-state Markov evolutionary trees (which we will refer to as METs).

THEOREM 1. *Let δ and ϵ be any positive constants. If our algorithm is given* $\mathrm{poly}(n, 1/\epsilon, 1/\delta)$ *samples from any MET M with any n-leaf topology T, then with probability at least $1-\delta$, the MET M' constructed by the algorithm satisfies* $\mathrm{var}(M, M') \leq \epsilon$.

Interesting PAC-learning algorithms for biologically important restricted classes of METs have been given by Farach and Kannan in [9] and by Ambainis et al. in [2]. These algorithms (and their relation to our algorithm) will be discussed more fully in section 1.1. At this point, we simply note that these algorithms only apply to METs which satisfy the following restrictions.

Restriction 1. All transition matrices are symmetric (the probability of a "1" turning into a "0" along an edge is the same as the probability of a "0" turning into a "1").

Restriction 2. For some positive constant α, every pair of leaves (x, y) satisfies $\Pr(x \neq y) \leq 1/2 - \alpha$.

We will explain in section 1.1 why the restrictions significantly simplify the problem of learning Markov evolutionary trees (though they certainly do not make it easy!). The main contribution of our paper is to remove the restrictions.

While we have used variation distance (L_1 distance) to measure the distance between the target distribution $\mathcal{D}$ and our hypothesis distribution $\mathcal{D}'$, Kearns et al. formulated the problem of learning probability distributions in terms of the Kullback–Leibler (KL) divergence distance from the target distribution to the hypothesis distribution (see [6]). This distance is defined as the sum over all length-n strings s of $\mathcal{D}(s)\log(\mathcal{D}(s)/\mathcal{D}'(s))$. Kearns et al. point out that the KL distance gives an upper bound on variation distance, in the sense that the KL distance from $\mathcal{D}$ to $\mathcal{D}'$ is $\Omega(\mathrm{var}(\mathcal{D}, \mathcal{D}')^2)$. Hence if a class of distributions can be PAC-learned using KL distance, it can be PAC-learned using variation distance. We justify our use of the variation distance metric by showing that the reverse is true. In particular, we prove the following lemma in the appendix.

LEMMA 2. *A class of probability distributions over the domain $\{0,1\}^n$ that is PAC-learnable under the variation distance metric is PAC-learnable under the KL-distance measure.*

The lemma is proved using a method related to the ϵ-Bayesian shift of Abe and Warmuth [3]. Note that the result requires a discrete domain of support for the target distribution, such as the domain $\{0,1\}^n$ which we use here.

The rest of this section is organized as follows: Subsection 1.1 discusses previous work related to the general Markov model of evolution, and the relationship between this work and our work. Subsection 1.2 gives a brief synopsis of our algorithm for PAC-learning Markov evolutionary trees. Subsection 1.3 discusses an interesting connection between the problem of learning Markov evolutionary trees and the problem of learning mixtures of Hamming balls, which was studied by Kearns et al. [11].

1.1. Previous work and its relation to our work. The two-state general Markov model [14] which we study in this paper is a generalization of the Cavender–Farris–Neyman model of evolution [5, 10, 13]. Before defining the Cavender–Farris–Neyman Model, let us return to the two-state general Markov model. We will fix attention on the particular two-state alphabet $\{0, 1\}$. Thus, the stochastic transition matrix associated with edge e is simply the matrix

$$\begin{pmatrix} 1 - e_0 & e_0 \\ e_1 & 1 - e_1 \end{pmatrix},$$

where e_0 denotes the probability that a "0" turns into a "1" along edge e and e_1 denotes the probability that a "1" turns into a "0" along edge e. The Cavender–Farris–Neyman model is simply the special case of the two-state general Markov model in which the transition matrices are required to be symmetric. That is, it is the special case of the two-state general Markov model in which Restriction 1 holds (so $e_0 = e_1$ for every edge e).

We now describe past work on learning Markov evolutionary trees in the general Markov model and in the Cavender–Farris–Neyman model. Throughout the paper, we will define the *weight* $w(e)$ of an edge e to be $|1 - e_0 - e_1|$.

Steel [14] showed that if a j-state Markov evolutionary tree M satisfies (i) $\rho_i > 0$ for all i, and (ii) the determinant of every transition matrix is outside of $\{-1, 0, 1\}$, then the distribution of M uniquely determines its topology. In this case, he showed how to recover the topology, given the joint distribution of every pair of leaves. In the two-state case, it suffices to know the exact value of the covariances of every pair of leaves. In this case, he defined the weight $\Lambda(e)$ of an edge e from node v to node w to be

$$\Lambda(e) = \begin{cases} w(e)\sqrt{\Pr(v=0)\Pr(v=1)} & \text{if } w \text{ is a leaf, and} \\ w(e)\sqrt{\frac{\Pr(v=0)\Pr(v=1)}{\Pr(w=0)\Pr(w=1)}} & \text{otherwise.} \end{cases} \tag{1}$$

Steel observed that these distances are multiplicative along a path and that the distance between two *leaves* is equal to their covariance. Since the distances are multiplicative along a path, their logarithms are additive. Therefore, methods for constructing trees from additive distances such as the method of Bandelt and Dress [4] can be used to reconstruct the topology. Steel's method does not show how to recover the *parameters* of a Markov evolutionary tree, even when the exact distribution is known and $j = 2$. In particular, the quantity that he obtains for each edge e is a one-dimensional distance rather than a *two*-dimensional vector giving the two transition probabilities e_0 and e_1. Our method shows how to recover the parameters exactly, given the exact distribution, and how to recover the parameters approximately (well enough to approximate the distribution), given polynomially-many samples from M.

Farach and Kannan [9] and Ambainis et al. [2] worked primarily in the special case of the two-state general Markov model satisfying the two restrictions on page 2. Farach and Kannan's paper was a breakthrough, because prior to their paper nothing

was known about the feasibility of reconstructing Markov evolutionary trees from samples. For any given positive constant α, they showed how to PAC-learn the class of METs which satisfy the two restrictions. However, the number of samples required is a function of $1/\alpha$, which is taken to be a constant. Ambainis et al. improved the bounds given by Farach and Kannan to achieve asymptotically tight upper and lower bounds on the number of samples needed to achieve a given variation distance. These results are elegant and important. Nevertheless, the restrictions that they place on the model do significantly simplify the problem of learning Markov evolutionary trees. In order to explain why this is true, we explain the approach of Farach et al.: Their algorithm uses samples from an MET M, which satisfies the restrictions above, to estimate the "distance" between any two leaves. (The distance is related to the covariance between the leaves.) The authors then relate the distance between two leaves to the amount of evolutionary time that elapses between them. The distances are thus turned into times. Then the algorithm of [1] is used to approximate the inter-leaf evolutionary times with times which are close, but form an additive metric, which can be fitted onto a tree. Finally, the times are turned back into transition probabilities. The symmetry assumption is essential to this approach because it is *symmetry* that relates a one-dimensional quantity (evolutionary time) to an otherwise two-dimensional quantity (the probability of going from a "0" to a "1" and the probability of going from a "1" to a "0"). The second restriction is also essential: If the probability that x differs from y were allowed to approach $1/2$, then the evolutionary time from x to y would tend to ∞. This would mean that in order to approximate the inter-leaf times accurately, the algorithm would have to get the distance estimates *very* accurately, which would require many samples. Ambainis et al. [2] generalized their results to a symmetric version of the j-state evolutionary model, subject to the two restrictions above.

Erdös et al. [7, 8] also considered the reconstruction of Markov evolutionary trees from samples. Like Steel [14] and unlike our paper or the papers of Farach and Kannan [9] and Ambainis et al. [2], Erdös et al. were interested in reconstructing only the *topology* of an MET (rather than its parameters or distribution), and they were interested in using as few samples as possible to reconstruct the topology. They showed how to reconstruct topologies in the j-state general Markov model when the Markov evolutionary trees satisfy the following: (i) every root probability is bounded above 0, (ii) every transition probability is bounded above 0 and below $1/2$, and (iii) for positive quantities λ and λ', the determinant of the transition matrix along each edge is between λ and $1 - \lambda'$. The number of samples required is polynomial in the worst case, but is only polylogarithmic in certain cases including the case in which the MET is drawn uniformly at random from one of several (specified) natural distributions. Note that restriction (iii) of Erdös et al. is weaker than Farach and Kannan's Restriction 2. However, Erdös et al. show only how to reconstruct the topology (thus they work in a restricted case in which the topology can be uniquely constructed using samples). They do not show how to reconstruct the parameters of the Markov evolutionary tree or how to approximate its distribution.

1.2. A synopsis of our method. In this paper, we provide the first polynomial-time PAC-learning algorithm for the class of METs. Our algorithm works as follows: First, using samples from the target MET, the algorithm estimates all of the pairwise covariances between leaves of the MET. Second, using the covariances, the leaves of the MET are partitioned into "related sets" of leaves. Essentially, leaves in different related sets have such small covariances between them that it is not always possible to use polynomially many samples to discover how the related sets are connected

in the target topology. Nevertheless, we show that we can closely approximate the *distribution* of the target MET by approximating the distribution of each related set closely, and then joining the related sets by "cut edges." The first step, for each related set, is to discover an approximation to the correct topology. Since we do not restrict the class of METs which we consider, we cannot guarantee to construct the *exact* induced topology (in the target MET). Nevertheless we guarantee to construct a good enough approximation. The topology is constructed by looking at triples of leaves. We show how to ensure that each triple that we consider has large inter-leaf covariances. We derive quadratic equations which allow us to approximately recover the parameters of the triple, using estimates of inter-leaf covariances and estimates of probabilities of particular outputs. We compare the outcomes for different triples and use the comparisons to construct the topology. Once we have the topology, we again use our quadratic equations to discover the parameters of the tree. As we show in section 2.4, we are able to prevent the error in our estimates from accumulating, so we are able to guarantee that each estimated parameter is within a small additive error of the "real" parameter in a (normalized) target MET. From this, we can show that the variation distance between our hypothesis and the target is small.

1.3. Markov evolutionary trees and mixtures of Hamming balls. A *Hamming ball distribution* [11] over binary strings of length n is defined by a *center* (a string c of length n) and a *corruption probability* p. To generate an output from the distribution, one starts with the center, and then flips each bit (or not) according to an independent Bernoulli experiment with probability p. A *linear mixture* of j Hamming balls is a distribution defined by j Hamming ball distributions, together with j probabilities $\rho_1, \ldots, \rho_j$ which sum to 1 and determine from which Hamming ball distribution a particular sample should be taken. For any fixed j, Kearns et al. give a polynomial-time PAC-learning algorithm for a mixture of j Hamming balls, *provided all j Hamming balls have the same corruption probability.*[2]

A *pure* distribution over binary strings of length n is defined by n probabilities, $\lambda_1, \ldots, \lambda_n$. To generate an output from the distribution, the ith bit is set to "0" independently with probability λ_i, and to "1" otherwise. A pure distribution is a natural generalization of a Hamming ball distribution. Clearly, every linear mixture of j pure distributions can be realized by a j-state MET with a star-shaped topology. Thus, the algorithm given in this paper shows how to learn a linear mixture of any two pure distributions. Furthermore, a generalization of our result to a j-ary alphabet would show how to learn any linear mixture of any j pure distributions.

2. The algorithm. Our description of our PAC-learning algorithm and its analysis requires the following definitions. For positive constants δ and ϵ, the input to the algorithm consists of $\text{poly}(n, 1/\epsilon, 1/\delta)$ samples from an MET M with an n-leaf topology T. We will let $\epsilon_1 = \epsilon/(20n^2)$, $\epsilon_2 = \epsilon_1/(4n^3)$, $\epsilon_3 = \epsilon_2^4/2^6$, $\epsilon_4 = \epsilon_1/(4n)$, $\epsilon_5 = \epsilon_2\epsilon_4/2^{10}$, and $\epsilon_6 = \epsilon_5\epsilon_2^3/2^7$. We have made no effort to optimize these constants. However, we state them explicitly so that the reader can verify below that the constants can be defined consistently. We define an ϵ_4-*contraction* of an MET with topology T' to be a tree formed from T' by contracting some internal edges e

[2]The kind of PAC-learning that we consider in this paper is *generation.* Kearns et al. also show how to do *evaluation* for the special case of the mixture of j Hamming balls described above. Using the observation that the output distributions of the subtrees below a node of an MET are independent, provided the bit at that node is fixed, we can also solve the evaluation problem for METs. In particular, we can calculate (in polynomial time) the probability that a given string is output by the hypothesis MET.

for which $\Lambda(e) > 1 - \epsilon_4$, where $\Lambda(e)$ is the edge-distance of e as defined by Steel [14] (see (1)). If x and y are leaves of the topology T then we use the notation $\mathrm{cov}(x, y)$ to denote the covariance of the indicator variables for the events "the bit at x is 1" and "the bit at y is 1." Thus,

$$\mathrm{cov}(x, y) = \Pr(xy = 11) - \Pr(x = 1)\Pr(y = 1). \tag{2}$$

We will use the following observations.

Observation 3. If MET M' has topology T' and e is an internal edge of T' from the root r to node v and T'' is a topology that is the same as T' except that v is the root (so e goes from v to r) then we can construct an MET with topology T'' which has the same distribution as M'. To do this, we simply set $\Pr(v = 1)$ appropriately (from the distribution of M'). If $\Pr(v = 1) = 0$ we set e_0 to be $\Pr(r = 1)$ (from the distribution of M'). If $\Pr(v = 1) = 1$ we set e_1 to be $\Pr(r = 0)$ (from the distribution of M'). Otherwise, we set $e_0 = \Pr(r = 1)(\text{old } e_1)/\Pr(v = 0)$ and $e_1 = \Pr(r = 0)(\text{old } e_0)/\Pr(v = 1)$.

Observation 4. If MET M' has topology T' and v is a degree-2 node in T' with edge e leading into v and edge f leading out of v and T'' is a topology which is the same as T' except that e and f have been contracted to form edge g then there is an MET with topology T'' which has the same distribution as M'. To construct it, we simply set $g_0 = e_0(1 - f_1) + (1 - e_0)f_0$ and $g_1 = e_1(1 - f_0) + (1 - e_1)f_1$.

Observation 5. If MET M' has topology T' then there is an MET M'' with topology T' which has the same distribution on its leaves as M' and has every internal edge e satisfy $e_0 + e_1 \leq 1$.

Proof of Observation 5. We will say that an edge e is "good" if $e_0 + e_1 \leq 1$. Starting from the root we can make all edges along a path to a leaf good, except perhaps the last edge in the path. If edge e from u to v is the first nongood edge in the path we simply set e_0 to $1 - (\text{old } e_0)$ and e_1 to $1 - (\text{old } e_1)$. This makes the edge good but it has the side effect of interchanging the meaning of 0 and 1 at node v. As long as we interchange 0 and 1 an even number of times along every path we will preserve the distribution at the leaves. Thus, we can make all edges good except possibly the last one, which we use to get the parity of the number of interchanges correct. □

We will now describe the algorithm. In subsection 2.6, we will prove that with probability at least $1 - \delta$, the MET M' that it constructs satisfies $\mathrm{var}(M, M') \leq \epsilon$. Thus, we will prove Theorem 1.

2.1. Step 1: Estimate the covariances of pairs of leaves. For each pair (x, y) of leaves, obtain an "observed" covariance $\widehat{\mathrm{cov}}(x, y)$ such that, with probability at least $1 - \delta/3$, all observed covariances satisfy

$$\widehat{\mathrm{cov}}(x, y) \in [\mathrm{cov}(x, y) - \epsilon_3, \mathrm{cov}(x, y) + \epsilon_3].$$

LEMMA 6. *Step* 1 *requires only* $\mathrm{poly}(n, 1/\epsilon, 1/\delta)$ *samples from* M.

Proof. Consider leaves x and y and let p denote $\Pr(xy = 11)$. By a Chernoff bound (see [12]), after k samples the observed proportion of outputs with $xy = 11$ is within $\pm\epsilon_3/4$ of p, with probability at least $1 - 2\exp(-k\epsilon_3^2/2^3)$. For each pair (x, y) of leaves, we estimate $\Pr(xy = 11)$, $\Pr(x = 1)$, and $\Pr(y = 1)$ within $\pm\epsilon_3/4$. From these estimates, we can calculate $\widehat{\mathrm{cov}}(x, y)$ within $\pm\epsilon_3$ using (2). □

2.2. Step 2: Partition the leaves of M into related sets. Consider the following *leaf connectivity graph* whose nodes are the leaves of M. Nodes x and y are connected by a "positive" edge if $\widehat{\mathrm{cov}}(x,y) \geq (3/4)\epsilon_2$ and are connected by a "negative" edge if $\widehat{\mathrm{cov}}(x,y) \leq -(3/4)\epsilon_2$. Each connected component in this graph (ignoring the signs of edges) forms a set of "related" leaves. For each set S of related leaves, let $s(S)$ denote the leaf in S with smallest index. METs have the property that for leaves x, y, and z, $\mathrm{cov}(y,z)$ is positive iff $\mathrm{cov}(x,y)$ and $\mathrm{cov}(y,z)$ have the same sign. To see this, use the following equation, which can be proved by algebraic manipulation from (2):

$$\mathrm{cov}(x,y) = \Pr(v=1)\Pr(v=0)(1-\alpha_0-\alpha_1)(1-\beta_0-\beta_1), \tag{3}$$

where v is taken to be the least common ancestor of x and y and α_0 and α_1 are the transition probabilities along the path from v to x and β_0 and β_1 are the transition probabilities along the path from v to y. Therefore, as long as the observed covariances are as accurate as stated in Step 1, the signs on the edges of the leaf connectivity graph partition the leaves of S into two sets S_1 and S_2 in such a way that $s(S) \in S_1$, all covariances between pairs of leaves in S_1 are positive, all covariances between pairs of leaves in S_2 are positive, and all covariances between a leaf in S_1 and a leaf in S_2 are negative.

For each set S of related leaves, let $T(S)$ denote the subtree formed from T by deleting all leaves which are not in S, contracting all degree-2 nodes, and then rooting at the neighbour of $s(S)$. Let $M(S)$ be an MET with topology $T(S)$ which has the same distribution as M on its leaves and satisfies the following:

(4)
- Every internal edge e of $M(S)$ has $e_0 + e_1 \leq 1$.
- Every edge e to a node in S_1 has $e_0 + e_1 \leq 1$.
- Every edge e to a node in S_2 has $e_0 + e_1 \geq 1$.

Observations 3, 4, and 5 guarantee that $M(S)$ exists.

Observation 7. As long as the observed covariances are as accurate as stated in Step 1 (which happens with probability at least $1-\delta/3$), then for any related set S and any leaf $x \in S$ there is a leaf $y \in S$ such that $|\mathrm{cov}(x,y)| \geq \epsilon_2/2$.

Observation 8. As long as the observed covariances are as accurate as stated in Step 1 (which happens with probability at least $1-\delta/3$), then for any related set S and any edge e of $T(S)$ there are leaves a and b which are connected through e and have $|\mathrm{cov}(a,b)| \geq \epsilon_2/2$.

Observation 9. As long as the observed covariances are as accurate as stated in Step 1 (which happens with probability at least $1-\delta/3$), then for any related set S, every internal node v of $M(S)$ has $\Pr(v=0) \in [\epsilon_2/2, 1-\epsilon_2/2]$.

Proof of Observation 9. Suppose to the contrary that v is an internal node of $M(S)$ with $\Pr(v=0) \in [0,\epsilon_2/2) \cup (1-\epsilon_2/2, 1]$. Using Observation 3, we can re-root $M(S)$ at v without changing the distribution. Let w be a child of v. By (3), every pair of leaves a and b which are connected through (v,w) satisfy $|\mathrm{cov}(a,b)| \leq \Pr(v=0)\Pr(v=1) < \epsilon_2/2$. The observation now follows from Observation 8. □

Observation 10. As long as the observed covariances are as accurate as stated in Step 1 (which happens with probability at least $1-\delta/3$), then for any related set S, every edge e of $M(S)$ has $w(e) \geq \epsilon_2/2$.

Proof of Observation 10. This follows from Observation 8 using (3). (Recall that $w(e) = |1-e_0-e_1|$.) □

2.3. Step 3: For each related set S, find an ϵ_4-contraction $T'(S)$ of $T(S)$. In this section, we will assume that the observed covariances are as accurate as stated in Step 1. (This happens with probability at least $1-\delta/3$.) Let S be a related set. With probability at least $1-\delta/(3n)$ we will find an ϵ_4-contraction $T'(S)$ of $T(S)$. Since there are at most n related sets, all ϵ_4-contractions will be constructed with probability at least $1-\delta/3$. Recall that an ϵ_4-contraction of $M(S)$ is a tree formed from $T(S)$ by contracting some internal edges e for which $\Lambda(e) > 1-\epsilon_4$. We start with the following observation, which will allow us to redirect edges for convenience.

Observation 11. If e is an internal edge of $T(S)$ then $\Lambda(e)$ remains unchanged if e is redirected as in Observation 3.

Proof. The observation can be proved by algebraic manipulation from (1) and Observation 3. Note (from Observation 9) that every endpoint v of e satisfies $\Pr(v = 0) \in (0,1)$. Thus, the redirection in Observation 3 is not degenerate and $\Lambda(e)$ is defined. □

We now describe the algorithm for constructing an ϵ_4-contraction $T'(S)$ of $T(S)$. We will build up $T'(S)$ inductively, adding leaves from S one by one. That is, when we have an ϵ_4-contraction $T'(S')$ of a subset S' of S, we will consider a leaf $x \in S-S'$ and build an ϵ_4-contraction $T'(S'\cup\{x\})$ of $T(S'\cup\{x\})$. Initially, $S' = \emptyset$. The precise order in which the leaves are added does not matter, but we will not add a new leaf x unless S' contains a leaf y such that $|\widehat{\mathrm{cov}}(x,y)| \geq (3/4)\epsilon_2$. When we add a new leaf x we will proceed as follows. First, we will consider $T'(S')$, and for every edge $e' = (u', v')$ of $T'(S')$, we will use the method in the following section (section 2.3.1) to estimate $\Lambda(e')$. More specifically, we will let u and v be nodes which are adjacent in $T(S')$ and have $u \in u'$ and $v \in v'$ in the ϵ_4-contraction $T'(S')$. We will show how to estimate $\Lambda(e)$. Afterwards (in section 2.3.2), we will show how to insert x.

2.3.1. Estimating $\Lambda(e)$. In this section, we suppose that we have an MET $M(S')$ on a set S' of leaves, all of which form a single related set. $T(S')$ is the topology of $M(S')$ and $T'(S')$ is an ϵ_4-contraction of $T(S')$. The edge $e' = (u', v')$ is an edge of $T'(S')$. $e = (u, v)$ is the edge of $T(S')$ for which $u \in u'$ and $v \in v'$. We wish to estimate $\Lambda(e)$ within $\pm\epsilon_4/16$. We will ensure that the overall probability that the estimates are not in this range is at most $\delta/(6n)$.

The proof of the following equations is straightforward. We will typically apply them in situations in which z is the error of an approximation:

$$\frac{x+z}{y-z} = \frac{x}{y} + \left(\frac{z}{y-z}\right)\left(1+\frac{x}{y}\right), \tag{5}$$

$$\frac{1+z}{1-z} \leq 1+4z \quad \text{if } z \leq 1/2, \tag{6}$$

$$\frac{1-z}{1+z} \geq 1-2z \quad \text{if } z \geq 0. \tag{7}$$

Case 1*: e' is an internal edge.* We first estimate e_0, e_1, $\Pr(u=0)$, and $\Pr(v=0)$ within $\pm\epsilon_5$ of the correct values. By Observation 9, $\Pr(u=0)$ and $\Pr(v=0)$ are in $[\epsilon_2/2, 1-\epsilon_2/2]$. Thus, our estimate of $\Pr(u=0)$ is within a factor of $(1\pm 2\epsilon_5/\epsilon_2) = (1\pm\epsilon_4 2^{-9})$ of the correct value. Similarly, our estimates of $\Pr(u=1)$, $\Pr(v=0)$, and $\Pr(v=1)$ are within a factor of $(1\pm\epsilon_4 2^{-9})$ of the correct values. Now using (1) we can estimate $\Lambda(e)$ within $\pm\epsilon_4/16$. In particular, our estimate of $\Lambda(e)$ is at most

$$\begin{aligned}
&(w(e)+2\epsilon_5)\sqrt{\frac{\Pr(v=0)\Pr(v=1)}{\Pr(w=0)\Pr(w=1)}}\frac{(1+\epsilon_4 2^{-9})}{(1-\epsilon_4 2^{-9})}\\
&\le (w(e)+2\epsilon_5)\sqrt{\frac{\Pr(v=0)\Pr(v=1)}{\Pr(w=0)\Pr(w=1)}}(1+\epsilon_4 2^{-7})\\
&\le \Lambda(e)+\epsilon_4/16.
\end{aligned}$$

In the inequalities, we used (6) and the fact that $\Lambda(e) \le 1$. Similarly, by (7), our estimate of $\Lambda(e)$ is at least

$$\begin{aligned}
&(w(e)-2\epsilon_5)\sqrt{\frac{\Pr(v=0)\Pr(v=1)}{\Pr(w=0)\Pr(w=1)}}\frac{(1-\epsilon_4 2^{-9})}{(1+\epsilon_4 2^{-9})}\\
&\ge (w(e)-2\epsilon_5)\sqrt{\frac{\Pr(v=0)\Pr(v=1)}{\Pr(w=0)\Pr(w=1)}}(1-\epsilon_4 2^{-8})\\
&\ge \Lambda(e)-\epsilon_4/16.
\end{aligned}$$

We now show how to estimate e_0, e_1, $\Pr(u=0)$, and $\Pr(v=0)$ within $\pm\epsilon_5$. We say that a path from node α to node β in an MET is *strong* if $|\mathrm{cov}(\alpha,\beta)| \ge \epsilon_2/2$. It follows from (3) that if node γ is on this path, then

$$|\mathrm{cov}(\gamma,\beta)| \ge |\mathrm{cov}(\alpha,\beta)|, \tag{8}$$

$$|\mathrm{cov}(\alpha,\beta)| \ge |\mathrm{cov}(\alpha,\gamma)|\,|\mathrm{cov}(\gamma,\beta)|, \tag{9}$$

We say that a quartet $(c,b \mid a,d)$ of leaves a, b, c, and d is a *good estimator* of the edge $e=(u,v)$ if e is an edge of $T(S')$ and the following hold in $T(S')$ (see Figure 1):

1. a is a descendent of v.
2. The undirected path from c to a is strong and passes through u then v.
3. The path from u to its descendent b is strong and only intersects the (undirected) path from c to a at node u.
4. The path from v to its descendent d is strong and only intersects the path from v to a at node v.

We say that $(c,b \mid a,d)$ is an *apparently good estimator* of e' if the following hold in the ϵ_4-contraction $T'(S')$:

1. a is a descendent of v'.
2. The undirected path from c to a is strong and passes through u' then v'.
3. The path from u' to its descendent b is strong and only intersects the (undirected) path from c to a at node u'.
4. The path from v' to its descendent d is strong and only intersects the path from v' to a at node v'.

Observation 12. If e is an edge of $T(S')$ and $(c,b \mid a,d)$ is a good estimator of e, then any leaves $x,y \in \{a,b,c,d\}$ have $|\mathrm{cov}(x,y)| \ge (\epsilon_2/2)^3$.

Proof. The observation follows from (8) and (9) and from the definition of a good estimator. ∎

LEMMA 13. *If $(c,b \mid a,d)$ is a good estimator of e, then it can be used (along with $\mathrm{poly}(n,1/\epsilon,1/\delta)$ samples from $M(S')$) to estimate e_0, e_1, $\Pr(u=0)$, and $\Pr(v=0)$ within $\pm\epsilon_5$. (If we use sufficiently many samples, then the probability that any of the estimates is not within $\pm\epsilon_5$ of the correct value is at most $\delta/(12n^7)$).*

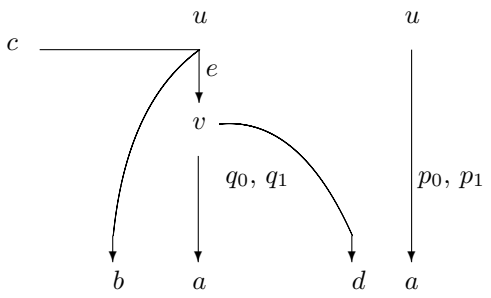


FIG. 1. *Finding* $\Pr(u=1)$, e_0, *and* e_1.

Proof. Let q_0 and q_1 denote the transition probabilities from v to a (see Figure 1) and let p_0 and p_1 denote the transition probabilities from u to a. We will first show how to estimate p_0, p_1, and $\Pr(u=1)$ within $\pm\epsilon_6$. Without loss of generality (by Observation 3) we can assume that c is a descendant of u. (Otherwise we can re-root $T(S')$ at u without changing the distribution on the nodes or p_0 or p_1.) Let β be the path from u to b and let γ be the path from u to c. We now define

$$\begin{aligned}\text{(10)}\qquad \operatorname{cov}(b,c,0) &= \Pr(abc=011)\Pr(a=0)-\Pr(ab=01)\Pr(ac=01),\\ \operatorname{cov}(b,c,1) &= \Pr(abc=111)\Pr(a=1)-\Pr(ab=11)\Pr(ac=11).\end{aligned}$$

(These do not quite correspond to the conditional covariances of b and c, but they are related to these.) We also define

$$\begin{aligned}F &= \frac{1}{2}\left(\frac{\operatorname{cov}(b,c)+\operatorname{cov}(b,c,0)-\operatorname{cov}(b,c,1)}{\operatorname{cov}(b,c)}\right), \text{ and}\\ D &= F^2-\operatorname{cov}(b,c,0)/\operatorname{cov}(b,c).\end{aligned}$$

The following equations can be proved by algebraic manipulation from (10), (2), and the definitions of F and D.

$$\begin{aligned}\text{(11)}\qquad \operatorname{cov}(b,c,0) &= \Pr(u=1)\Pr(u=0)(1-\beta_0-\beta_1)(1-\gamma_0-\gamma_1)p_1(1-p_0),\\ \operatorname{cov}(b,c,1) &= \Pr(u=1)\Pr(u=0)(1-\beta_0-\beta_1)(1-\gamma_0-\gamma_1)p_0(1-p_1),\end{aligned}$$

$$\text{(12)}\qquad F=\frac{1+p_1-p_0}{2},$$

$$\text{(13)}\qquad D=\frac{(1-p_0-p_1)^2}{4}.$$

Case 1a*:* $a\in S_1$. In this case, by (4) and by Observation 10, we have $1-p_0-p_1>0$. Thus, by (13), we have

$$\text{(14)}\qquad \sqrt{D}=\frac{1-p_0-p_1}{2}.$$

Equations (12) and (14) imply

$$\text{(15)}\qquad p_1=F-\sqrt{D},$$

$$\text{(16)}\qquad p_0=1-F-\sqrt{D}.$$

Also, since $\Pr(a=0) = \Pr(u=1)p_1 + (1-\Pr(u=1))(1-p_0)$, we have

$$\Pr(u=1) = \frac{1}{2} + \frac{F - \Pr(a=0)}{2\sqrt{D}}. \tag{17}$$

From these equations, it is clear that we could find p_0, p_1, and $\Pr(u=1)$ if we knew $\Pr(a=0)$, $\mathrm{cov}(b,c)$, $\mathrm{cov}(b,c,0)$, and $\mathrm{cov}(b,c,1)$ exactly. We now show that with polynomially many samples, we can approximate the values of $\Pr(a=0)$, $\mathrm{cov}(b,c)$, $\mathrm{cov}(b,c,0)$, and $\mathrm{cov}(b,c,1)$ sufficiently accurately so that using our approximations and the above equations, we obtain approximations for p_0, p_1, and $\Pr(u=1)$ which are within $\pm\epsilon_6$. As in the proof of Lemma 6, we can use (2) and (10) to estimate $\Pr(a=0)$, $\mathrm{cov}(b,c)$, $\mathrm{cov}(b,c,0)$, and $\mathrm{cov}(b,c,1)$ within $\pm\epsilon'$ for any ϵ' whose inverse is at most a polynomial in n and $1/\epsilon$. Note that our estimate of $\mathrm{cov}(b,c)$ will be nonzero by Observation 12 (as long as $\epsilon' \le (\epsilon_2/2)^3$), so we will be able to use it to estimate F from its definition. Now, using the definition of F and (5), our estimate of $2F$ is at most

$$2F + \frac{3\epsilon'}{\mathrm{cov}(b,c) - 3\epsilon'}(1+2F).$$

By Observation 12, this is at most

$$2F + \frac{3\epsilon'}{(\epsilon_2/2)^3 - 3\epsilon'}(1+2). \tag{18}$$

The error is at most ϵ'' for any ϵ'' whose is inverse is at most polynomial in n and $1/\epsilon$. (This is accomplished by making ϵ' small enough with respect to ϵ_2 according to (18).) We can similarly bound the amount that we underestimate F. Now we use the definition of D to estimate D. Our estimate is at most

$$(F+\epsilon'')^2 - \frac{\mathrm{cov}(b,c,0) - \epsilon'}{\mathrm{cov}(b,c) + \epsilon'}.$$

Using (5), this is at most

$$D + 2\epsilon'' F + \epsilon''^2 + \frac{\epsilon'}{\mathrm{cov}(b,c)+\epsilon'}\left(1 + \frac{\mathrm{cov}(b,c,0)}{\mathrm{cov}(b,c)}\right).$$

Once again, by Observation 12, the error can be made within $\pm\epsilon'''$ for any ϵ''' whose inverse is polynomial in n and $1/\epsilon$ (by making ϵ' and ϵ'' sufficiently small). It follows that our estimate of $\sqrt{D}$ is at most $\sqrt{D}(1+\epsilon'''/(2D))$ and (since Observation 12 gives us an upper bound on the value of D as a function of ϵ_2), we can estimate $\sqrt{D}$ within $\pm\epsilon''''$ for any ϵ'''' whose inverse is polynomial in n and $1/\epsilon$. This implies that we can estimate p_0 and p_1 within $\pm\epsilon_6$. Observation 12 and (3) imply that $w(p) \ge (\epsilon_2/2)^3$. Thus, the estimate for $\sqrt{D}$ is nonzero. This implies that we can similarly estimate $\Pr(u=1)$ within $\pm\epsilon_6$ using (17).

Now that we have estimates for p_0, p_1, and $\Pr(u=1)$ which are within $\pm\epsilon_6$ of the correct values, we can repeat the trick to find estimates for q_0 and q_1 which are also within $\pm\epsilon_6$. We use leaf d for this. Observation 4 implies that

$$e_0 = \frac{p_0 - q_0}{1 - q_0 - q_1} \text{ and } e_1 = \frac{p_1 - q_1}{1 - q_0 - q_1}.$$

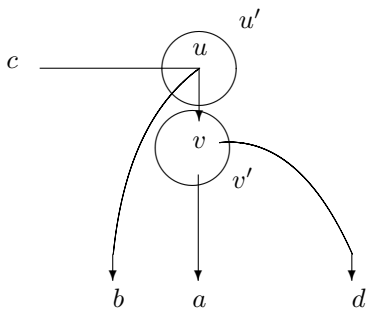


FIG. 2. $(c, b \mid a, d)$ *is a good estimator of* $e = (u, v)$ *and an apparently good estimator of* $e' = (u', v')$.

Using these equations, our estimate of e_0 is at most

$$\frac{p_0 - q_0 + 2\epsilon_6}{1 - q_0 - q_1 - 2\epsilon_6}.$$

Equation (5) and our observation above that $w(p) \geq (\epsilon_2/2)^3$ imply that the error is at most

$$\frac{2\epsilon_6}{(\epsilon_2/2)^3 - 2\epsilon_6}\left(1 + \frac{p_0 - q_0}{1 - q_0 - q_1}\right),$$

which is at most $2^7\epsilon_6/\epsilon_2^3 = \epsilon_5$. Similarly, the estimate for e_0 is at least $e_0 - \epsilon_5$ and the estimate for e_1 is within $\pm\epsilon_5$ of e_1. We have now estimated e_0, e_1, and $\Pr(u = 0)$ within $\pm\epsilon_5$. As we explained in the beginning of this section, we can use these estimates to estimate $\Lambda(e)$ within $\pm\epsilon_4/16$.

Case 1b*:* $a \in S_2$. In this case, by (4) and by Observation 10, we have $1 - p_0 - p_1 < 0$. Thus, by (13), we have

$$\sqrt{D} = -\left(\frac{1 - p_0 - p_1}{2}\right). \tag{19}$$

Equations (12) and (19) imply

$$p_1 = F + \sqrt{D}, \tag{20}$$

$$p_0 = 1 - F + \sqrt{D}. \tag{21}$$

Equation (17) remains unchanged. The process of estimating p_0, p_1, and $\Pr(u = 1)$ (from the new equations) is the same as for Case 1a. This concludes the proof of Lemma 13. □

Observation 14. Suppose that e' is an edge from u' to v' in $T'(S')$ and that $e = (u, v)$ is the edge in $T(S')$ such that $u \in u'$ and $v \in v'$. There is a good estimator $(c, b \mid a, d)$ of e. Furthermore, every good estimator of e is an apparently good estimator of e'. (Refer to Figure 2.)

Proof. Leaves c and a can be found to satisfy the first two criteria in the definition of a good estimator by Observation 8. Leaf b can be found to satisfy the third criterion by Observation 8 and (8) and by the fact that the degree of u is at least 3 (see the

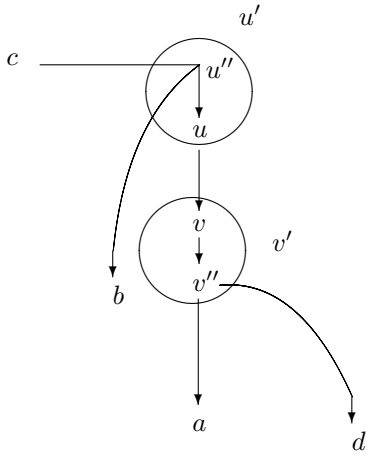


FIG. 3. $(c, b \mid a, d)$ *is an apparently good estimator of* $e' = (u', v')$ *and a good estimator of* $p = (u'', v'')$. $\Lambda(p) \leq \Lambda(u, v)$.

text just before (4)). Similarly, leaf d can be found to satisfy the fourth criterion. $(c, b \mid a, d)$ is an apparently good estimator of e' because only internal edges of $T(S')$ can be contracted in the ϵ_4-contraction $T'(S')$. □

Observation 15. Suppose that e' is an edge from u' to v' in $T'(S')$ and that $e = (u, v)$ is an edge in $T(S')$ such that $u \in u'$ and $v \in v'$. Suppose that $(c, b \mid a, d)$ is an apparently good estimator of e'. Let u'' be the meeting point of c, b, and a in $T(S')$. Let v'' be the meeting point of c, a, and d in $T(S')$. (Refer to Figure 3.) Then $(c, b \mid a, d)$ is a good estimator of the path p from u'' to v'' in $T(S')$. Also, $\Lambda(p) \leq \Lambda(e)$.

Proof. The fact that $(c, b \mid a, d)$ is a good estimator of p follows from the definition of good estimator. The fact that $\Lambda(p) \leq \Lambda(e)$ follows from the fact that the distances Λ are multiplicative along a path, and bounded above by 1. □

Observations 14 and 15 imply that in order to estimate $\Lambda(e)$ within $\pm\epsilon_4/16$, we need only estimate $\Lambda(e)$ using each apparently good estimator of e' and then take the maximum. By Lemma 13, the failure probability for any given estimator is at most $\delta/(12n^7)$, so with probability at least $1 - \delta/(12n^3)$, all estimators give estimates within $\pm\epsilon_4/16$ of the correct values. Since there are at most $2n$ edges e' in $T'(S')$, and we add a new leaf x to S' at most n times, all estimates are within $\pm\epsilon_4/16$ with probability at least $1 - \delta/(6n)$.

Case 2*: e' is not an internal edge.* In this case $v = v'$ since v' is a leaf of $T(S')$. We say that a pair of leaves (b, c) is a good estimator of e if the following holds in $T(S')$: The paths from leaves v, b, and c meet at u and $|\mathrm{cov}(v, b)|$, $|\mathrm{cov}(v, c)|$, and $|\mathrm{cov}(b, c)|$ are all at least $(\epsilon_2/2)^2$. We say that (b, c) is an apparently good estimator of e' if the following holds in $T'(S')$: The paths from leaves v, b, and c meet at u' and $|\mathrm{cov}(v, b)|$, $|\mathrm{cov}(v, c)|$, and $|\mathrm{cov}(b, c)|$ are all at least $(\epsilon_2/2)^2$. As in the previous case, the result follows from the following observations.

Observation 16. If (b, c) is a good estimator of e then it can be used (along with $\mathrm{poly}(n, 1/\epsilon, 1/\delta)$ samples from $M(S')$) to estimate e_0, e_1, and $\Pr(u = 0)$ within $\pm\epsilon_5$.

(The probability that any of the estimates is not within $\pm\epsilon_5$ of the correct value is at most $\delta/(12n^3)$.)

Proof. This follows from the proof of Lemma 13. □

Observation 17. Suppose that e' is an edge from u' to leaf v in $T'(S')$ and that $e = (u, v)$ is an edge in $T(S')$ such that $u \in u'$. There is a good estimator (b, c) of e. Furthermore, every good estimator of e is an apparently good estimator of e'.

Proof. This follows from the proof of Observation 14 and from (9). □

Observation 18. Suppose that e' is an edge from u' to leaf v in $T'(S')$ and that $e = (u, v)$ is an edge in $T(S')$ such that $u \in u'$. Suppose that (b, c) is an apparently good estimator of e'. Let u'' be the meeting point of b, v, and c in $T(S')$. Then (b, c) is a good estimator of the path p from u'' to v in $T(S')$. Also, $\Lambda(p) \leq \Lambda(e)$.

Proof. This follows from the proof of Observation 15. □

2.3.2. Using the estimates of $\Lambda(e)$. We now return to the problem of showing how to add a new leaf x to $T'(S')$. As we indicated above, for every internal edge $e' = (u', v')$ of $T'(S')$, we use the method in section 2.3.1 to estimate $\Lambda(e)$ where $e = (u, v)$ is the edge of $T(S')$ such that $u \in u'$ and $v \in v'$. If the observed value of $\Lambda(e)$ exceeds $1 - 15\epsilon_4/16$, then we will contract e. The accuracy of our estimates will guarantee that we will not contract e if $\Lambda(e) \leq 1 - \epsilon_4$, and that we definitely contract e if $\Lambda(e) > 1 - 7\epsilon_4/8$. We will then add the new leaf x to $T'(S')$ as follows. We will insert a new edge (x, x') into $T'(S')$. We will do this by either (i) identifying x' with a node already in $T'(S')$, or (ii) splicing x' into the middle of some edge of $T'(S')$.

We will now show how to decide where to attach x' in $T'(S')$. We start with the following definitions. Let S'' be the subset of S' such that for every $y \in S''$ we have $|\mathrm{cov}(x, y)| \geq (\epsilon_2/2)^4$. Let T'' be the subtree of $T'(S')$ induced by the leaves in S''. Let S''' be the subset of S' such that for every $y \in S'''$ we have $|\widehat{\mathrm{cov}}(x, y)| \geq (\epsilon_2/2)^4 - \epsilon_3$. Let T''' be the subtree of $T'(S')$ induced by the leaves in S'''.

Observation 19. If $T(S' \cup \{x\})$ has x' attached to an edge $e = (u, v)$ of $T(S')$ and e' is the edge corresponding to e in $T'(S')$ (that is, $e' = (u', v')$, where $u \in u'$ and $v \in v'$), then e' is an edge of T''.

Proof. By Observation 14 there is a good estimator $(c, b \mid a, d)$ for e. Since x is being added to S' (using (8)), $|\mathrm{cov}(x, x')| \geq \epsilon_2/2$. Thus, by Observation 12 and (9), every leaf $y \in \{a, b, c, d\}$ has $|\mathrm{cov}(x, y)| \geq (\epsilon_2/2)^4$. Thus, a, b, c, and d are all in S'' so e' is in T''. □

Observation 20. If $T(S' \cup \{x\})$ has x' attached to an edge $e = (u, v)$ of $T(S')$ and u and v are both contained in node u' of $T'(S')$ then u' is a node of T''.

Proof. Since u is an internal node of $T(S')$, it has degree at least 3. By Observation 8 and (8), there are three leaves a_1, a_2, and a_3 meeting at u with $|\mathrm{cov}(u, a_i)| \geq \epsilon_2/2$. Similarly, $|\mathrm{cov}(u, v)| \geq \epsilon_2/2$. Thus, for each a_i, $|\mathrm{cov}(x, a_i)| \geq (\epsilon_2/2)^3$ so a_1, a_2, and a_3 are in S''. □

Observation 21. $S'' \subseteq S'''$.

Proof. This follows from the accuracy of the covariance estimates in Step 1. □

We will use the following algorithm to decide where to attach x' in T'''. In the algorithm, we will use the following tool. For any triple (a, b, c) of leaves in $S' \cup \{x\}$, let u denote the meeting point of the paths from leaves a, b, and c in $T(S' \cup \{x\})$. Let M^u be the MET which has the same distribution as $M(S' \cup \{x\})$, but is rooted at u. (M^u exists, by Observation 3.) Let $\Lambda_c(a, b, c)$ denote the weight of the path from u to c in M^u. By observation 11, $\Lambda_c(a, b, c)$ is equal to the weight of the path from u to c in $M(S' \cup \{x\})$. (This follows from the fact that re-rooting at u only redirects

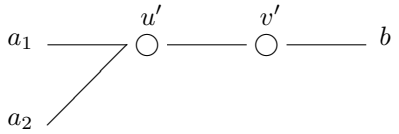


FIG. 4. *The setting for Test* $1(u', v', a_1, a_2, b)$ *and Test* $2(u', v', a_1, a_2, b)$ *when* v' *is an internal node of* T'''. *(If* v' *is a leaf, we perform the same tests with* $v' = b$.*)*

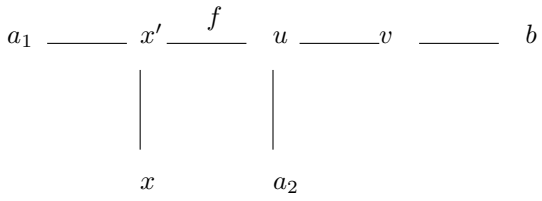


FIG. 5. *Either Test* $1(u', v', a_1, a_2, b)$ *fails or Test* $2(u', v', a_1, a_2, b)$ *fails.*

internal edges.) It follows from the definition of Λ (as in (1)) and from (3) that

$$\Lambda_c(a, b, c) = \sqrt{\frac{\mathrm{cov}(a, c)\mathrm{cov}(b, c)}{\mathrm{cov}(a, b)}}. \tag{22}$$

If a, b, and c are in $S''' \cup \{x\}$, then by the accuracy of the covariance estimates and (8) and (9), the absolute value of the pairwise covariance of any pair of them is at least $\epsilon_2^8/2^{10}$. As in section 2.3.1, we can estimate $\mathrm{cov}(a, c)$, $\mathrm{cov}(b, c)$, and $\mathrm{cov}(a, b)$ within a factor of $(1 \pm \epsilon')$ of the correct values for any ϵ' whose inverse is at most a polynomial in n, and $1/\epsilon$. Thus, we can estimate $\Lambda_c(a, b, c)$ within a factor of $(1 \pm \epsilon_4/16)$ of the correct value. We will take sufficiently many samples to ensure that the probability that any of the estimates is outside of the required range is at most $\delta/(6n^2)$. Thus, the probability that any estimate is outside of the range for any x is at most $\delta/(6n)$.

We will now determine where in T''' to attach x'. Choose an arbitrary internal root u' of T'''. We will first see where x' should be placed with respect to u'. For each neighbor v' of u' in T''', each pair of leaves (a_1, a_2) on the "u'" side of (u', v'), and each leaf b on the "v'" side of (u', v') (see Figure 4), perform the following two tests.

- *Test* $1(u', v', a_1, a_2, b)$: The test succeeds if the observed value of $\Lambda_x(a_1, x, b)/\Lambda_x(a_2, x, b)$ is at least $1 - \epsilon_4/4$.
- *Test* $2(u', v', a_1, a_2, b)$: The test succeeds if the observed value of $\Lambda_b(a_1, a_2, b)/\Lambda_b(a_1, x, b)$ is at most $1 - 3\epsilon_4/4$.

We now make the following observations.

Observation 22. If x is on the "u side" of (u, v) in $T(S''' \cup \{x\})$ and u is in u' in T''' and v is in $v' \neq u'$ in T''' then some test fails.

Proof. Since u' is an internal node of T''', it has degree at least 3. Thus, we can construct a test such as the one depicted in Figure 5. (If $x' = u$ then the figure is still correct; that would just mean that $\Lambda(f) = 1$. Similarly, if v' is a leaf, we simply have $\Lambda(f') = 1$ where f' is the edge from v to b.) Now we have

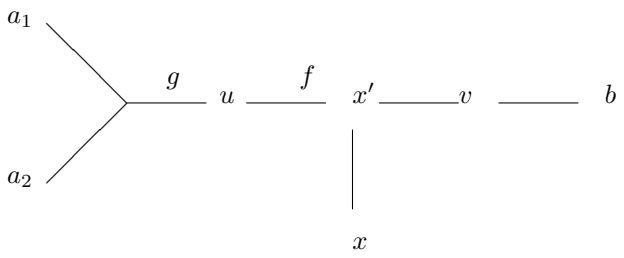


FIG. 6. *Test* $1(u', v', a_1, a_2, b)$ *and Test* $2(u', v', a_1, a_2, b)$ *succeed for all choices of* a_1, a_2, *and* b.

$$\frac{1}{\Lambda(f)} = \frac{\Lambda_x(a_1, x, b)}{\Lambda_x(a_2, x, b)} = \frac{\Lambda_b(a_1, a_2, b)}{\Lambda_b(a_1, x, b)}.$$

However, Test $1(u', v', a_1, a_2, b)$ will succeed only if the left hand fraction is at least $1 - \epsilon_4/4$. Furthermore, Test $2(u', v', a_1, a_2, b)$ will only succeed if the right hand fraction is at most $1 - 3\epsilon_4/4$. Since our estimates are accurate to within a factor of $(1 \pm \epsilon_4/16)$, at least one of the two tests will fail. □

Observation 23. If x is between u and v in $T(S''' \cup \{x\})$ and the edge f from u to x' has $\Lambda(f) \leq 1 - 7\epsilon_4/8$ then Test $1(u', v', a_1, a_2, b)$ and Test $2(u', v', a_1, a_2, b)$ succeed for all choices of a_1, a_2, and b.

Proof. Every such test has the form depicted in Figure 6, where again g might be degenerate, in which case $\Lambda(g) = 1$. Observe that $\Lambda_x(a_1, x, b)/\Lambda_x(a_2, x, b) = 1$, so its estimate is at least $1 - \epsilon_4/4$ and Test 1 succeeds. Furthermore,

$$\frac{\Lambda_b(a_1, a_2, b)}{\Lambda_b(a_1, x, b)} = \Lambda(f)\Lambda(g) \leq \Lambda(f) \leq 1 - 7\epsilon_4/8,$$

so the estimate is at most $1 - 3\epsilon_4/4$ and Test 2 succeeds. □

Observation 24. If x is on the "v side" of (u, v) in $T(S''' \cup \{x\})$ and $\Lambda(e) \leq 1 - 7\epsilon_4/8$ (recall from the beginning of section 2.3.2 that $\Lambda(e)$ *is* at most $1 - 7\epsilon_4/8$ if u and v are in different nodes of T'''), then Test $1(u', v', a_1, a_2, b)$ and Test $2(u', v', a_1, a_2, b)$ succeed for all choices of a_1, a_2, and b.

Proof. Note that this case only applies if v is an internal node of $T(S''')$. Thus, every such test has one of the forms depicted in Figure 7, where some edges may be degenerate. Observe that in both cases $\Lambda_x(a_1, x, b)/\Lambda_x(a_2, x, b) = 1$, so its estimate is at least $1 - \epsilon_4/4$ and Test 1 succeeds. Also in both cases

$$\frac{\Lambda_b(a_1, a_2, b)}{\Lambda_b(a_1, x, b)} = \Lambda(e)\Lambda(f)\Lambda(g) \leq \Lambda(e) \leq 1 - 7\epsilon_4/8,$$

so the estimate is at most $1 - 3\epsilon_4/4$ and Test 2 succeeds. □

Now note (using Observation 22) that node u' has at most one neighbor v' for which all tests succeed. Furthermore, if there is no such v', Observations 23 and 24 imply that x' can be merged with u'. The only case that we have not dealt with is the case in which there is exactly one v' for which all tests succeed. In this case, if v' is a leaf, we insert x' in the middle of edge (u', v'). Otherwise, we will either insert x' in the middle of edge (u', v'), or we will insert it in the subtree rooted at v'. In order to decide which, we perform similar tests from node v', and we check whether Test $1(v', u', a_1, a_2, b)$ and Test $2(v', u', a_1, a_2, b)$ both succeed for all choices of a_1, a_2, and b. If so, we put x' in the middle of edge (u', v'). Otherwise, we recursively place x' in the subtree rooted at v'.

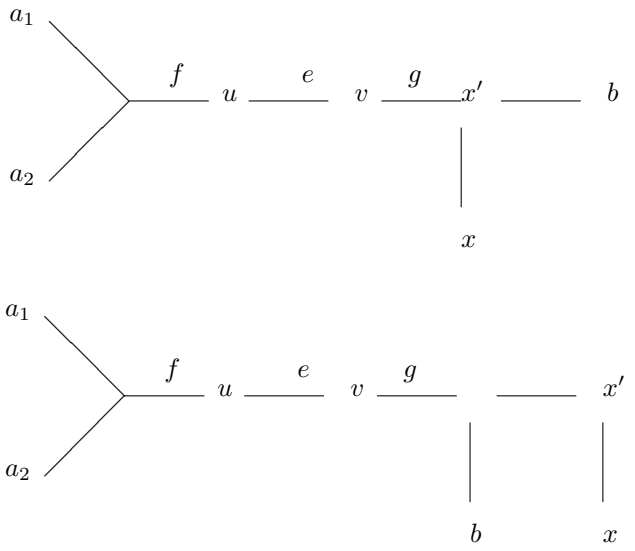


FIG. 7. *Test* $1(u', v', a_1, a_2, b)$ *and Test* $2(u', v', a_1, a_2, b)$ *succeed for all choices of* a_1, a_2, *and* b.

2.4. Step 4: For each related set S, construct an MET $M'(S)$ which is close to $M(S)$. For each set S of related leaves we will construct an MET $M'(S)$ with leaf-set S such that each edge parameter of $M'(S)$ is within $\pm\epsilon_1$ of the corresponding parameter of $M(S)$. The topology of $M'(S)$ will be $T'(S)$. We will assume without loss of generality that $T(S)$ has the same root as $T'(S)$. The failure probability for S will be at most $\delta/(3n)$, so the overall failure will be at most $\delta/3$.

We start by observing that the problem is easy if S has only one or two leaves.

Observation 25. If $|S| < 3$ then we can construct an MET $M'(S)$ such that each edge parameter of $M'(S)$ is within $\pm\epsilon_1$ of the corresponding parameter of $M(S)$.

We now consider the case in which S has at least three leaves. Any edge of $T(S)$ which is contracted in $T'(S)$ can be regarded as having e_0 and e_1 set to 0. The fact that these are within $\pm\epsilon_1$ of their true values follows from the following lemma.

LEMMA 26. *If e is an internal edge of $M(S)$ from v to w with $\Lambda(e) > 1 - \epsilon_4$ then* $e_0 + e_1 < 2\epsilon_4 = \epsilon_1/(2n)$.

Proof. First observe from Observation 9 that $\Pr(w = 0) \notin \{0, 1\}$ and from Observation 10 that $e_0 + e_1 \neq 1$. Using algebraic manipulation, one can see that

$$\Pr(v = 1) = \frac{\Pr(w = 1) - e_0}{1 - e_0 - e_1},$$
$$\Pr(v = 0) = \frac{\Pr(w = 0) - e_1}{1 - e_0 - e_1}.$$

Thus, by (1),

$$\Lambda(e)^2 = \left(1 - \frac{e_0}{\Pr(w = 1)}\right)\left(1 - \frac{e_1}{\Pr(w = 0)}\right).$$

Since $\Lambda(e)^2 \geq 1 - 2\epsilon_4$, we have $e_0 \leq 2\epsilon_4 \Pr(w = 1)$ and $e_1 \leq 2\epsilon_4 \Pr(w = 0)$, which proves the observation. □

Thus, we need only show how to label the remaining parameters within $\pm\epsilon_1$. Note that we have already shown how to do this in section 2.3.1. Here the total failure

probability is at most $\delta/(3n)$ because there is a failure probability of at most $\delta/(6n^2)$ associated with each of the $2n$ edges.

2.5. Step 5: Form M' from the METs $M'(S)$. Make a new root r for M' and set $\Pr(r=1)=1$. For each related set S of leaves, let u denote the root of $M'(S)$, and let $\overline{p}$ denote the probability that u is 0 in the distribution of $M'(S)$. Make an edge e from r to u with $e_1=\overline{p}$.

2.6. Proof of Theorem 1. Let M'' be an MET which is formed from M as follows.

- Related sets are formed as in Step 2.
- For each related set S, a copy $M''(S)$ of $M(S)$ is made.
- The METs $M''(S)$ are combined as in Step 5.

Theorem 1 follows from the following lemmas.

LEMMA 27. *Suppose that for every set S of related leaves, every parameter of $M'(S)$ is within $\pm\epsilon_1$ of the corresponding parameter in $M(S)$. Then* $\mathrm{var}(M'',M')\le\epsilon/2$.

Proof. First, we observe (using a crude estimate) that there are at most $5n^2$ parameters in M'. (Each of the (at most n) METs $M'(S)$ has one root parameter and at most $4n$ edge parameters.) We will now show that changing a single parameter of a MET by at most $\pm\epsilon_1$ yields at MET whose variation distance from the original is at most $2\epsilon_1$. This implies that $\mathrm{var}(M'',M')\le 10n^2\epsilon_1=\epsilon/2$. Suppose that e is an edge from u to v and e_0 is changed. The probability that the output has string s on the leaves below v and string s' on the remaining leaves is

$$\begin{aligned}&\Pr(u=0)\Pr(s'\mid u=0)(e_0\Pr(s\mid v=1)+(1-e_0)\Pr(s\mid v=0))\\ &+\Pr(u=1)\Pr(s'\mid u=1)(e_1\Pr(s\mid v=0)+(1-e_1)\Pr(s\mid v=1)).\end{aligned}$$

Thus, the variation distance between M'' and an MET obtained by changing the value of e_0 (within $\pm\epsilon_1$) is at most

$$\begin{aligned}&\epsilon_1\sum_s\sum_{s'}\Pr(u=0)\Pr(s'\mid u=0)(\Pr(s\mid v=1)+\Pr(s\mid v=0))\\ &\le\epsilon_1\Pr(u=0)\left(\sum_{s'}\Pr(s'\mid u=0)\right)\left(\left(\sum_s\Pr(s\mid v=1)\right)+\left(\sum_s\Pr(s\mid v=0)\right)\right)\\ &\le 2\epsilon_1.\end{aligned}$$

Similarly, if ρ_1 is the root parameter of an MET then the probability of having output s is

$$\rho_1\Pr(s\mid r=1)+(1-\rho_1)\Pr(s\mid r=0).$$

So the variation distance between the original MET and one in which ρ_1 is changed within $\pm\epsilon_1$ is at most

$$\sum_s\epsilon_1(\Pr(s\mid r=1)+\Pr(s\mid r=0))\le 2\epsilon_1. \qquad \square$$

LEMMA 28. $\mathrm{var}(M'',M)\le\epsilon/2$.

Before we prove Lemma 28, we provide some background material. Recall that the *weight* $w(e)$ of an edge e of an MET is $|1-e_0-e_1|$ and define the weight $w(\ell)$ of

a leaf ℓ to be the product of the weights of the edges on the path from the root to ℓ. We will use the following lemma.

LEMMA 29. *In any MET with root r, the variation distance between the distribution on the leaves conditioned on $r = 1$ and the distribution on the leaves conditioned on $r = 0$ is at most $2\sum_\ell w(\ell)$, where the sum is over all leaves ℓ.*

Proof. We proceed by induction on the number of edges in the MET. In the base case there are no edges so r is a leaf, and the result holds. For the inductive step, let e be an edge from r to node x. For any string s_1 on the leaves below x and any string s_2 on the other leaves,

$$\Pr(s_1 s_2 \mid r = 0) = \Pr(s_2 \mid r = 0)(e_0 \Pr(s_1 \mid x = 1) + (1 - e_0)\Pr(s_1 \mid x = 0)).$$

Algebraic manipulation of this formula shows that $\Pr(s_1 s_2 \mid r = 1) - \Pr(s_1 s_2 \mid r = 0)$ is

$$\begin{aligned}&(1 - e_0 - e_1)\Pr(s_2 \mid r = 1)\,(\Pr(s_1 \mid x = 1) - \Pr(s_1 \mid x = 0))\\ &+ \Pr(s_1 \mid r = 0)\,(\Pr(s_2 \mid r = 1) - \Pr(s_2 \mid r = 0)).\end{aligned} \tag{23}$$

It follows that the variation distance is at most the sum over all $s_1 s_2$ of the absolute value of the quantity in (23), which is at most

$$\begin{aligned}&|1 - e_0 - e_1| \left(\sum_{s_2} \Pr(s_2 \mid r = 1)\right)\left(\sum_{s_1} |\Pr(s_1 \mid x = 1) - \Pr(s_1 \mid x = 0)|\right)\\ &+ \left(\sum_{s_1} \Pr(s_1 \mid r = 0)\right)\left(\sum_{s_2} |\Pr(s_2 \mid r = 1) - \Pr(s_2 \mid r = 0)|\right).\end{aligned}$$

The result follows by induction. □

LEMMA 30. *Suppose that m is an MET with n leaves and that e is an edge from node u to node v. Let m' be the MET derived from m by replacing e_0 with $\Pr(v = 1)$ and e_1 with $\Pr(v = 0)$. Then $\mathrm{var}(m, m') \le n^2 z$, where z is the maximum over all pairs (x, y) of leaves which are connected via e in m of $|\mathrm{cov}(x, y)|$.*

Proof. By Observation 3, we can assume without loss of generality that u is the root of m. For any string s_1 on the leaves below v and any string s_2 on the remaining leaves, we find (via a little algebraic manipulation) that the difference between the probability that m outputs $s_1 s_2$ and the probability that m' does is

$$\begin{aligned}&\Pr(u = 1)\Pr(u = 0)(1 - e_0 - e_1)(\Pr(s_2 \mid u = 1)\\ &-\ \Pr(s_2 \mid u = 0))(\Pr(s_1 \mid v = 1) - \Pr(s_1 \mid v = 0)).\end{aligned}$$

Thus, the variation distance between m and m' is $\Pr(u = 1)\Pr(u = 0)(1 - e_0 - e_1)$ times the product of the variation distance between the distribution on the leaves below v conditioned on $v = 1$ and the distribution on the leaves below v conditioned on $v = 0$ and the variation distance between the distribution on the remaining leaves conditioned on $u = 1$ and the distribution on the remaining leaves conditioned on $u = 0$. By Lemma 29, this is at most

$$\Pr(u = 0)\Pr(u = 1)\left(2\sum_{\ell \text{ below } v} w(\ell)\right)\left(2\sum_{\text{other } \ell} w(\ell)\right),$$

which by (3) is

$$4 \sum_{(x,y) \text{ connected via } e} |\mathrm{cov}(x,y)|,$$

which is at most $4(n/2)^2 z = n^2 z$. □

LEMMA 31. *If, for two different related sets, S and S', an edge e from u to v is in $M(S)$ and in $M'(S)$, then $e_0 + e_1 \leq n^2\epsilon_2/(n+1)$.*

Proof. By the definition of the leaf connectivity graph in Step 2, there are leaves $a, a' \in S$ and $b, b' \in S'$ such that the path from a' to a and the path from b' to b both go through $e = u \to v$ and

$$|\widehat{\mathrm{cov}}(a,a')| \geq (3/4)\epsilon_2 \quad \text{and} \quad |\widehat{\mathrm{cov}}(b,b')| \geq (3/4)\epsilon_2,$$

and the remaining covariance estimates amongst leaves a, a', b, and b' are less than $(3/4)\epsilon_2$. Without loss of generality (using Observation 3), assume that u is the root of the MET. Let $p_{u,a'}$ denote the path from u to a' and use similar notation for the other leaves. By (3) and the accuracy of the estimates in Step 1,

$$\begin{aligned}
\Pr(u=0)^2 \Pr(u=1)^2 w(e)^2 w(p_{u,a'}) w(p_{v,a}) w(p_{u,b'}) w(p_{v,b}) &\geq ((3/4)\epsilon_2 - \epsilon_3)^2, \\
\Pr(u=0)\Pr(u=1) w(p_{u,a'}) w(p_{u,b'}) &< (3/4)\epsilon_2 + \epsilon_3, \\
\Pr(v=0)\Pr(v=1) w(p_{v,a}) w(p_{v,b}) &< (3/4)\epsilon_2 + \epsilon_3.
\end{aligned}$$

Thus,

$$w(e) \geq \left(1 - \frac{2\epsilon_3}{(3/4)\epsilon_2 + \epsilon_3}\right) \sqrt{\frac{\Pr(v=1)\Pr(v=0)}{\Pr(u=1)\Pr(u=0)}}.$$

By (1),

$$\Lambda(e) \geq 1 - \frac{2\epsilon_3}{(3/4)\epsilon_2 + \epsilon_3}.$$

The result now follows from the proof of Lemma 26. (Clearly, the bound in the statement of Lemma 31 is weaker than we can prove, but it is all that we will need.) □

Proof of Lemma 28. Let M^* be the MET which is the same as M except that every edge e which is contained in $M(S)$ and $M(S')$ for two different related sets S and S' is contracted. Similarly, let M''^* be the MET which is the same as M'' except that every such edge has *all* of its copies contracted in M''^*. Clearly, $\mathrm{var}(M, M'') \leq \mathrm{var}(M, M^*) + \mathrm{var}(M^*, M''^*) + \mathrm{var}(M''^*, M'')$. Lemma 31 then implies that $\mathrm{var}(M, M^*) + \mathrm{var}(M''^*, M'') \leq \ell n^2 \epsilon_2$, where ℓ is the number of edges in M that are contracted. We now wish to bound $\mathrm{var}(M^*, M''^*)$. By construction, $M^*(S)$ and $M^*(S')$ do not intersect in an edge (for any related sets S and S'). Now suppose that $M^*(S)$ and $M^*(S')$ both contain node u. We can modify M^* *without changing the distribution* in a way that avoids this overlap. To do this, we just replace node u with two copies of u, and we connect the two copies by an edge e with $e_0 = e_1 = 0$. Note that this change will not affect the operation of the algorithm. Thus, without loss of generality, we can assume that for any related sets S and S', $M^*(S)$ and $M^*(S')$ do not intersect. Thus, M^* and M''^* are identical, except on edges which go between the

sub-METs $M^*(S)$. Now, any edge e going between two sub-METs has the property that for any pair of leaves, x and y connected via e, $|\mathrm{cov}(x,y)| \le \epsilon_2$. (This follows from the accuracy of our covariance estimates in Step 1.) Thus, by Lemma 30, changing such an edge according to Step 5 adds at most $n^2\epsilon_2$ to the variation distance. Thus, $\mathrm{var}(M^*, M''^*) \le \ell' n^2 \epsilon_2$, where ℓ' is the number of edges that are modified according to Step 5. We conclude that $\mathrm{var}(M, M'') \le (2n)n^2\epsilon_2 = \epsilon_1/2 \le \epsilon/2$. □

3. Appendix.

3.1. Proof of Lemma 2.

LEMMA 2. *A class of probability distributions over the domain $\{0,1\}^n$ that is PAC-learnable under the variation distance metric is PAC-learnable under the KL-distance measure.*

Proof. Let K be a polynomial in three inputs and let A be an algorithm which takes as input $K(n, 1/\epsilon, 1/\delta)$ samples from a distribution $\mathcal{D}$ from the class of distributions and, with probability at least $1-\delta$, returns a distribution $\mathcal{D}'$ such that $\mathrm{var}(\mathcal{D}, \mathcal{D}') \le \epsilon$. Without loss of generality, we can assume that ϵ is sufficiently small. For example, it will suffice to have $\epsilon \le 2/15$.

Define algorithm A' as follows. Let $\xi = \epsilon^2/(12n)$. Run A with sample size $K(n, 1/\xi, 1/\delta)$. (Note that the sample size is polynomial in n, $1/\epsilon$, and $1/\delta$.) Let $\mathcal{D}'$ be the distribution returned by A. Let $\mathcal{U}$ denote the uniform distribution on $\{0,1\}^n$ and let $\mathcal{D}''$ be the distribution defined by

$$\mathcal{D}''(s) = (1 - (\xi))\mathcal{D}'(s) + \xi\,\mathcal{U}(s).$$

With probability at least $1-\delta$, $\mathrm{var}(\mathcal{D}, \mathcal{D}') \le \xi$. By definition of $\mathcal{D}''$, $\mathrm{var}(\mathcal{D}', \mathcal{D}'') \le 2\xi$. Thus, with probability at least $1-\delta$, $\mathrm{var}(\mathcal{D}, \mathcal{D}'') < 3\xi$. Note that for all s, $\mathcal{D}''(s) \ge \xi\, 2^{-n}$. Let S be the set of all output strings s satisfying $\mathcal{D}''(s) < \mathcal{D}(s)$. S contains all the strings which contribute positively to the KL-distance from $\mathcal{D}$ to $\mathcal{D}''$. Thus,

$$\begin{aligned}\mathrm{KL}(\mathcal{D}, \mathcal{D}'') &\le \sum_{s\in S} \mathcal{D}(s)(\log \mathcal{D}(s) - \log \mathcal{D}''(s)) \\ &= \sum_{s\in S} (\mathcal{D}(s) - \mathcal{D}''(s))(\log \mathcal{D}(s) - \log \mathcal{D}''(s)) + \sum_{s\in S} \mathcal{D}''(s)(\log \mathcal{D}(s) - \log \mathcal{D}''(s)).\end{aligned}$$

We have seen that $\mathrm{var}(\mathcal{D}, \mathcal{D}'') \le 3\xi$. Thus, $\sum_{s\in S}(\mathcal{D}(s) - \mathcal{D}''(s)) \le 3\xi$. So, the first term is at most

$$\begin{aligned}&\max_{s\in S}(\log \mathcal{D}(s) - \log \mathcal{D}''(s)) \sum_{s\in S}(\mathcal{D}(s) - \mathcal{D}''(s)) \\ &\le 3\xi \max_{s\in S}(\log \mathcal{D}(s) - \log \mathcal{D}''(s)) \\ &\le 3\xi \max_{s\in S}(-\log \mathcal{D}''(s)) \\ &\le 3\xi(-\log(\xi\, 2^{-n})) \\ &= 3\xi(n - \log(\xi)).\end{aligned}$$

Furthermore, the second term is at most

$$\sum_{s\in S} \mathcal{D}''(s)(\log \mathcal{D}(s) - \log \mathcal{D}''(s))$$
$$= \sum_{s\in S} \mathcal{D}''(s)(\log(\mathcal{D}''(s) + h_s) - \log \mathcal{D}''(s)),$$

where $h_s = \mathcal{D}(s) - \mathcal{D}''(s)$, which is a positive quantity for $s \in S$. By concavity of the logarithm function, the above quantity is at most

$$\sum_{s\in S} \mathcal{D}''(s) h_s \Big[\frac{d}{dx}(\log(x))\Big]_{x=\mathcal{D}''(s)} = \sum_{s\in S} h_s \le 3\xi.$$

Thus, $\mathrm{KL}(\mathcal{D}, \mathcal{D}'') \le 3\xi(1 + n - \log \xi)$. This quantity is at most ϵ for all $n \ge 1$ by the definition of ξ. □

The method in the proof of Lemma 2 converts a hypothesis distribution which is close (in variation distance) to the target distribution to a hypothesis distribution which is close (in KL-distance) to the target distribution. However, if the original hypothesis is given as a two-state MET, then the modified hypothesis would require a three-state MET to realize it. We conclude the paper by explaining how to perform a similar trick using only two-state METs. The distribution obtained is not quite the same as the one used in the proof of Lemma 2, but it has the properties needed to show that small KL-distance is achieved.

Let M be the target Markov evolutionary tree. We run the PAC learning algorithm with accuracy parameter $\xi = \epsilon^2/(12n^3)$ to obtain MET M'. Now we construct a new hypothesis M'' by adjusting some of the parameters of M' as follows:

For each edge $e = (u, l)$ of M' where l is a leaf, let e_0 and e_1 be its parameters. If $e_0 < \xi$ then we set $e_0 = \xi$ and if $e_0 > 1 - \xi$ then set $e_0 = 1 - \xi$. We make the same change to e_1. By the proof of Lemma 27, $\mathrm{var}(M', M'') \le 4n\xi$, since $2n$ parameters have each been changed by at most ξ. Hence, with probability at least $1 - \delta$, $\mathrm{var}(M, M'') \le (1 + 4n)\xi$.

For each string $s \in \{0, 1\}^n$, $M''(s) \ge \xi^n$ (where $M''(s)$ denotes the probability that M'' outputs s). Using a similar argument to the proof of Lemma 2,

$$\mathrm{KL}(M, M'') \le (1 + 4n)\xi(1 - \log(\xi^n)) = (1 + 4n)\xi(1 - n\log \xi)$$

$$= (1 + 4n)\frac{\epsilon^2}{12n^3}(1 - n(2\log \epsilon - 3\log n - \log 12))$$

which as before is at most ϵ for all $n \ge 1$.

Acknowledgment. We thank Mike Paterson for useful ideas and discussions.

REFERENCES

[1] R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup, *On the approximability of numerical taxonomy (Fitting distances by tree metrics),* in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 365–372.

[2] A. Ambainis, R. Desper, M. Farach, and S. Kannan, *Nearly tight bounds on the learnability of evolution,* in Proceedings of the 38th Annual IEEE Symposium on the Foundations of Computer Science, 1997, pp. 524–533.

[3] N. Abe and M.K. Warmuth, *Polynomial learnability of probabilistic concepts with respect to the Kullback-Leibler divergence,* in Proceedings of the 1992 Conference on Computational Learning Theory, 1992, pp. 277–289.

[4] H.J. Bandelt and A. Dress, *Reconstructing the shape of a tree from observed dissimilarity data,* Adv. Appl. Math., 7 (1987), pp. 309–343.
[5] J.A. Cavender, *Taxonomy with confidence,* Math. Biosci., 40 (1978), pp. 271–280.
[6] T.H. Cover and J.A. Thomas, *Elements of Information Theory,* John Wiley, New York, 1991.
[7] P.L. Erdös, M.A. Steel, L.A. Székely, and T.J. Warnow, *A Few Logs Suffice to Build (Almost) All Trees* I, DIMACS Technical Report 97-71, 1997.
[8] P.L. Erdös, M.A. Steel, L.A. Székely, and T.J. Warnow, *A Few Logs Suffice to Build (Almost) All Trees* II, DIMACS Technical Report 97-72, 1997. (A preliminary version appeared as *Inferring big trees from short quartets,* in Proceedings of the 24th International Colloquium on Automata, Languages and Programming, 1997, pp. 827–837.)
[9] M. Farach and S. Kannan, *Efficient algorithms for inverting evolution,* in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 230–236.
[10] J.S. Farris, *A probability model for inferring evolutionary trees,* Syst. Zool., 22 (1973), pp. 250–256.
[11] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie, *On the learnability of discrete distributions,* in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, 1994, pp. 273–282.
[12] C. McDiarmid, *On the method of bounded differences,* in Surveys in Combinatorics 1989, London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.
[13] J. Neyman, *Molecular studies of evolution: A source of novel statistical problems,* in Statistical Decision Theory and Related Topics, S.S. Gupta and J. Yackel, eds., Academic Press, London, 1971, pp. 1–27.
[14] M. Steel, *Recovering a tree from the leaf colourations it generates under a Markov model,* Appl. Math. Lett., 7 (1994), pp. 19–24.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 398–427

THE COMPLEXITY OF COUNTING IN SPARSE, REGULAR, AND PLANAR GRAPHS*

SALIL P. VADHAN†

Abstract. We show that a number of graph-theoretic counting problems remain $\mathcal{NP}$-hard, indeed $\#\mathcal{P}$-complete, in very restricted classes of graphs. In particular, we prove that the problems of counting matchings, vertex covers, independent sets, and extremal variants of these all remain hard when restricted to planar bipartite graphs of bounded degree or regular graphs of constant degree. We obtain corollaries about counting cliques in restricted classes of graphs and counting satisfying assignments to restricted classes of monotone 2-CNF formulae. To achieve these results, a new interpolation-based reduction technique which preserves properties such as constant degree is introduced.

Key words. $\#\mathcal{P}$, completeness, matchings, vertex covers, independent sets, Fibonacci numbers, polynomial interpolation

AMS subject classification. 68Q17

PII. S0097539797321602

1. Introduction.

From the time that Valiant [48, 49] introduced the class $\#\mathcal{P}$ of counting problems and gave a complexity-theoretic explanation for the apparent difficulty of enumeration, counting has held an important place in theoretical computer science. Although many researchers have continued Valiant's work by adding to the list of $\#\mathcal{P}$-complete problems, our understanding of the complexity of counting still pales in comparison to our understanding of decision problems.

This is unfortunate because, aside from being mathematically interesting, counting is closely related to important practical problems. For instance, reliability problems are often equivalent to counting problems. Computing the probability that a graph remains connected given a probability of failure on each edge is essentially equivalent to counting the number of ways that the edges could fail without losing connectivity. Counting problems also arise naturally in artificial intelligence research [33, 36, 40]. As explained by Roth [40], various methods used in reasoning, such as computing "degree of belief" and "Bayesian belief networks" are computationally equivalent to counting the number of satisfying assignments to a propositional formula. Thus, understanding the types of propositional formulae for which counting satisfying assignments is feasible tells us the extent to which these reasoning techniques might be useful. Graph-theoretic counting problems such as the ones we consider also appear often in statistical physics (cf. [15, 22, 19, 29]).

Perhaps the most significant deficiency in our understanding of counting is that, in many cases, we do not know whether hard counting problems remain hard when additional restrictions are placed on the problem instances. A quick glance at Garey and Johnson's famous catalogue of $\mathcal{NP}$-complete problems [13] reveals that the restricted-case complexity of most difficult decision problems is understood in detail. This in-

*Received by the editors May 19, 1997; accepted for publication (in revised form) December 8, 2000; published electronically August 8, 2001.
http://www.siam.org/journals/sicomp/31-2/32160.html

†Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (salil@deas.harvard.edu, http://deas.harvard.edu/~salil). Much of this work originally appeared in the author's undergraduate thesis [45], done at Harvard University under the supervision of Leslie Valiant, and some of it was done while the author was at MIT, supported by a DOD NDSEG graduate fellowship.

formation is useful, because a complexity-theoretic hardness result often leads us to ask whether the instances we are interested in possess special properties which make the problem tractable. Restricted-case complexity results tell us when such special properties do not make a problem any easier, closing the gap between what we can do and what we know we cannot.

While researchers have managed to prove a number of restricted-case hardness results for counting problems, the techniques have been somewhat ad hoc, requiring new ideas for each problem. One reason for this is that many of the known reductions between counting problems employ "blow-up" techniques, which destroy special properties of the original problem instance. This makes it difficult to deduce additional restricted-case results from known restricted-case results. For example, although Dagum and Luby [8] have shown that counting perfect matchings remains $\#\mathcal{P}$-complete when restricted to 3-regular bipartite graphs, the standard reduction from counting perfect matchings to counting matchings in [49] blows up the degree of the graph and does not enable us to conclude that counting matchings remains difficult in either regular or bounded-degree graphs.

Our results. In this paper, we introduce a new reduction technique that yields restricted-case complexity results for many problems of interest. In particular, we show in Theorem 4.1 that counting matchings, vertex covers, independent sets, and variants of these structures remains difficult in planar bipartite graphs of bounded degree and in regular graphs of constant degree. As immediate corollaries, we deduce hardness results for counting cliques and satisfying assignments in restricted classes of graphs and formulae. Our main reduction technique, like some of those in [49], is based on polynomial interpolation. However, in contrast to the reductions in most earlier papers on the complexity of counting, our reductions preserve graph properties such as regularity and degree-boundedness. Moreover, the technique is quite general, and it can be applied to different problems in an almost mechanical manner.

A summary of our results, together with previous work, is given in Tables 1.1 and 1.2. Precise definitions of the problems we consider can be found in sections 3 and 4 and a more detailed summary of related work is given in section 2. We note that our results for vertex covers, independent sets, monotone 2-CNF satisfying assignments, and cliques are essentially restatements of each other via well-known equivalences (cf. Propositions 3.1 and 3.2), but we list them separately on Tables 1.1 and 1.2 for clarity and comparison to previous work.

Several of our results answer open problems explicitly stated in previous work: counting maximal matchings in unrestricted graphs [49], counting satisfying assignments to monotone 2-CNF formulae in which every variable appears a bounded number of times [40], and counting satisfying assignments to planar 2-CNF formulae [17].

When counting remains hard even in restricted cases, the natural alternative is to seek *approximate* counting algorithms. However, restricted-case complexity results for approximate counting are even harder to come by than ones for exact counting. In Proposition 4.3, we obtain such a result, as we show that counting minimum cardinality vertex covers is $\mathcal{NP}$-hard even in graphs of maximal degree 3. Perhaps this could be used as a starting point for achieving other such results.

Techniques. Our main technique is best illustrated with an example: reducing #Perfect Matchings to #Matchings while preserving the sparsity of the input graph. Suppose we are given an oracle which counts all matchings in a graph, and we want to use this oracle to count the number of perfect matchings in a graph G. Let $v_1, \ldots, v_n$ be the vertices of G. For $s = 0, \ldots, n$, consider the graph G_s obtained by adding disjoint chains $v_i - v_{i,1} - v_{i,2} - \cdots - v_{i,s}$ to each vertex v_i of G.

TABLE 1.1 *$\#\mathcal{P}$-completeness results for counting problems in restricted classes of graphs.*

Problem	Polynomial-time solvable	Previous results	This paper
#PERFECT MATCHINGS	• planar [12, 27, 43] • $\Delta \leq 2$	• k-regular bipartite, any $k \geq 3$ [8] • $(n-k)$-regular bipartite, any $k \geq 3$ [8]	
#MATCHINGS	• $\Delta = 2$	• bipartite [49] • planar [19]	• bipartite, $\Delta = 4$ • planar bipartite, $\Delta = 6$ • k-regular, any $k \geq 5$
#MAXIMAL MATCHINGS	• $\Delta = 2$		• bipartite, $\Delta = 5$ • planar bipartite, $\Delta = 7$
#VERTEX COVERS #INDEPENDENT SETS	• $\Delta = 2$ • $n - \delta$ constant	• bipartite [38]	• planar bipartite, $\Delta = 4$ • k-regular, any $k \geq 5$[1]
#MIN CARDINALITY VERTEX COVERS #MAX CARDINALITY INDEPENDENT SETS	• $\Delta = 2$ • $n - \delta$ constant	• bipartite [38] • planar [17]	• planar bipartite, $\Delta = 3$ • k-regular, any $k \geq 4$
#MINIMAL VERTEX COVERS #MAXIMAL INDEPENDENT SETS	• $\Delta = 2$ • $n - \delta$ constant	• bipartite [38][2]	• planar bipartite, $\Delta = 5$ • regular
#CLIQUES, #MAXIMAL CLIQUES #MAXCARDINALITY CLIQUES[3]	• $\delta = n - 2$ • Δ constant • planar • bipartite	• unrestricted [49][2]	• regular • $\delta = n - 5$
#BIPARTITE CLIQUES, #MAXIMAL BIPARTITE CLIQUES #MAX CARDINALITY BIPARTITE CLIQUES[3]	• $\delta = n - 2$ • Δ constant • planar	• bipartite [38][2]	• bipartite • $\delta = n - 5$

Remarks.
• Δ (resp., δ) denotes an upper (resp., lower) bound on the maximum (resp., minimum) vertex degree.
• k is always a constant.
• n denotes the number of vertices, except when referring explicitly to bipartite graphs, in which case it refers to the number of vertices on each side.
[1] Greenhill [14] has improved this to $k = 3$.
[2] These results are not stated explicitly in [49, 38], but follow readily from the results and reductions in those papers.
[3] In general, any result for independent sets also holds for cliques *in the complement graph*, and conversely (cf. Proposition 3.1). An analogous relationship holds between independent sets in bipartite graphs and bipartite cliques in the bipartite complement graph (cf. Proposition 3.2). For brevity, we do not enumerate all the results we obtain in this way.

TABLE 1.2 *$\#\mathcal{P}$-completeness results for counting satisfying assignments in restricted classes of formulae.*

Problem	Polynomial-time solvable	Previous results	This paper
#SAT	• monotone 2-CNF, each variable appears at most twice [40] • monotone 2-CNF, each variable fails to appear in a clause with only a constant number of other variables • acyclic monotone 2-CNF [40] • acyclic Horn 2-CNF [40] • Horn 2-CNF, each variable appears at most twice [40]	• planar 3-CNF [17][1] • bipartite monotone 2-CNF [38] • Horn 2-CNF, each variable appears at most 3 times [40] • CNF, each literal appears exactly once [5] • monotone CNF, each variable appears at most twice [5]	• planar bipartite monotone 2-CNF, each variable appears at most 4 times • monotone 2-CNF, each variable appears exactly k times, any $k \geq 5$[2]
#MINTERMS	• monotone 2-CNF, each variable appears at most twice • monotone 2-CNF, each variable fails to appear in a clause with only a constant number of other variables	• bipartite monotone 2-CNF [49, 38][3]	• planar bipartite monotone 2-CNF, each variable appears at most 5 times • monotone 2-CNF, each variable appears the same number of times
#MIN WEIGHT SAT	• monotone 2-CNF, each variable appears at most twice • monotone 2-CNF, each variable fails to appear in a clause with only a constant number of other variable	• bipartite monotone 2-CNF [49, 38][3] • planar monotone 2-CNF [17][3]	• planar bipartite monotone 2-CNF, each variable appears at most 3 times • monotone 2-CNF, each variable appears exactly k times, and $k \geq 4$

Remarks.
• Terms such as planar, acyclic, etc., refer to properties of the graph obtained from a CNF formula by having a vertex for each variable and connecting two vertices if they appear in the same clause.
• k is always a constant.
[1] In [17], the graph associated to a CNF formula is the bipartite graph on variables and clauses in which a clause is connected to the variables it contains. For 2-CNF, the planarity of this graph is equivalent to the planarity of the graphs we consider.
[2] Greenhill [14] has improved this to $k = 3$.
[3] These results are not stated explicitly in [49, 38, 17], but follow readily from the results and reductions in those papers.

We will describe the number of matchings in G_s in terms of matchings in G. Consider any matching M in G, and let us count the number of ways M can be extended to a matching in G_s. For each vertex v_i of G which is matched by M, the edge $(v_i, v_{i,1})$ cannot be added to the matching, but we can choose an arbitrary matching for the chain $v_{i,1} - \cdots - v_{i,s}$. For each vertex v_i which is not matched by M, we can add an arbitrary matching of the chain $v_i - v_{i,1} - \cdots - v_{i,s}$ to M. Thus the number of ways to extend M to a matching in G_s is exactly $x_s^j x_{s+1}^{n-j}$, where j is the number of vertices matched by M and x_t denotes the number of matchings in a chain of t nodes. Therefore, if we let A_j denote the number matchings in G in which exactly j nodes are matched, then G_s has exactly $\sum_{j=0}^{n} A_j x_s^j x_{s+1}^{n-j}$ matchings. We can obtain these values for $s = 0, \ldots, n$ with $n+1$ oracle calls. Dividing by x_{s+1}^n, we obtain the evaluation of the polynomial $f(x) = \sum_{j=0}^{n} A_j x^j$ at the points (x_s/x_{s+1}) for $s = 0, \ldots, n$. Now, we would like to use polynomial interpolation to recover the coefficients of f, and in particular, the leading coefficient A_n which is the number of perfect matchings in G. We can do this provided we can compute the values x_s and the evaluation points x_s/x_{s+1} are distinct.

Luckily, it is not hard to get a handle on x_s, the number of matchings in a chain of s nodes. It turns out that x_s is simply the sth Fibonacci number! One can use the Fibonacci recurrence to compute the values x_s, and the well-known closed form for the Fibonacci numbers (cf. [44, section 7.3]) can be used to show that their consecutive ratios x_s/x_{s+1} never repeat. (We formally prove all this in Lemma 6.3.)

In contrast to the previously known reduction from #Perfect Matchings to #Matchings [49], the above reduction only increases the maximum vertex degree by one. It also preserves other graph properties, such as bipartiteness and planarity. Moreover, this technique generalizes quite easily to other problems and graph properties (such as regularity). We used only a few properties of chains and matchings in the reduction:

1. The number of matchings in the graphs G_s is related to matchings in the original graph G via polynomial evaluation.
2. The evaluation points can be expressed in terms of the number of matchings in a chain of length s (and the number in which one end vertex is not matched).
3. The evaluation points can be computed efficiently.
4. The evaluation points do not repeat.

We will see by inspection that analogues of the first two conditions still hold when we replace matchings with a variety of other problems, and when chains are replaced with other gadgets. For the last two conditions, we will use gadgets possessing a repetitive structure, so that the evaluation points satisfy simple linear recurrence relations. These recurrences will certainly allow the evaluation points to be computed efficiently, but we are left with proving that they do not repeat. To this end, in section 6 we prove the following lemma, which gives general conditions under which (ratios of) sequences defined by 2×2 linear recurrences do not repeat.

Lemma 6.2. *Let A, B, C, D, x_0, and y_0 be rational numbers. Define the sequences $\{x_n\}$ and $\{y_n\}$ recursively by $x_{n+1} = Ax_n + By_n$ and $y_{n+1} = Cx_n + Dy_n$. Then the sequence $\{z_n = x_n/y_n\}$ never repeats as long as all of the following conditions hold:*

$$AD - BC \neq 0, \tag{1}$$

$$D^2 - 2AD + A^2 + 4BC \neq 0, \tag{2}$$

$$D + A \neq 0, \tag{3}$$

$$D^2 + A^2 + 2BC \neq 0, \tag{4}$$

$$D^2 + AD + A^2 + BC \neq 0, \tag{5}$$

$$D^2 - AD + A^2 + 3BC \neq 0, \tag{6}$$

$$By_0^2 - Cx_0^2 - (A - D)x_0y_0 \neq 0. \tag{7}$$

Greenhill [14] has observed that when the coefficients A,B,C,D are all positive, Conditions (2)–(6) are guaranteed to hold, so only the first and last must be checked.

This lemma, with the approach outlined above, gives an almost mechanical way to find reductions that preserve properties such as sparsity and regularity. We will refer to this method as the *Fibonacci technique*, in reference to its simplest incarnation described above. The reductions we obtain from the Fibonacci technique have some interesting features not present in many previous interpolation-based reductions, such as those in [49]. First, our interpolation points are typically rapidly converging sequences of rational numbers (e.g., the consecutive ratios of Fibonacci numbers converge to the golden ratio), whereas previous methods often interpolated at distinct integer points, which seems difficult to do without losing special properties of the original graph. Second, we do not know how to reduce the number of oracle calls in these reductions to a constant. This is in contrast to the reductions done in [49]. There, Valiant asserts that all the reductions can be done with a single oracle call, because the arithmetic can be simulated by operations on the graph or formula in question. Here that does not appear to work, because the graph operations used by Valiant blow up the degree.

Of course, it is not enough to have reductions that preserve properties such as sparsity and regularity; we need initial hardness results for restricted classes of graphs from which to reduce. For this, we rely heavily on results and techniques from previous work, such as [8, 38, 19, 49, 40]. In particular, to obtain results about planar graphs, we use the Fibonacci technique to refine the reduction of Jerrum [19] so that properties such as sparsity and bipartiteness are preserved, and also extend his approach to problems involving vertex covers.

2. Related work. Our results for counting in sparse bipartite graphs first appeared in the author's undergraduate thesis [45], and the first version of this paper [47] added our results for planar and regular graphs. Some of the related work, namely that of Luby and Vigoda [28] and Bubley and Dyer [5, 6], was done subsequent to [45], but independently of [47]. We describe those works, together with more recent developments, under the heading "subsequent work." Throughout this section, we discuss only works that address the same counting problems as we do. The reader is referred to [50, 34, 45, 20] for more general surveys on the complexity of counting, and [41, 25, 31, 46, 23] for approximate counting.

Previous work. In his seminal paper [48], Valiant introduced the class $\#\mathcal{P}$ of counting problems and proved that counting perfect matchings in bipartite graphs is $\#\mathcal{P}$-complete. In [49], he showed that a number of other counting and reliability problems are $\#\mathcal{P}$-complete, including the unrestricted versions of most of the problems we study in this paper. Some of these problems, such as #Vertex Covers, were shown to remain hard in bipartite graphs by Provan and Ball [38], who also proved hardness results for reliability problems. Provan [37] obtained restricted-case results (acyclic planar graphs of maximum degree 3) for some reliability problems, but not the problems that we investigate. Jerrum [19] showed that counting matchings in planar graphs is $\#\mathcal{P}$-complete, in striking contrast to perfect matchings which can efficiently counted in planar graphs via algorithms due to Fisher, Kasteleyn, and Temperley [12, 27, 43]. Broder [4] proved that counting perfect matchings in bipartite

graphs of minimum vertex degree at least $n/2$ is $\#\mathcal{P}$-complete. Dagum and Luby [8] obtained even stronger results, showing that counting perfect matchings remains hard even in k-regular and $(n-k)$-regular bipartite graphs, for any constant $k \geq 3$.

From the start, the $\#\mathcal{P}$-completeness of counting satisfying assignments to a propositional formula was seen to follow immediately from parsimonious versions of Cook's reduction [48]. Valiant [49] proved that the problem remained just as hard in the dramatically restricted case of monotone 2-CNF, which was restricted further to bipartite monotone 2-CNF by Provan and Ball [38]. Roth [40] showed that counting satisfying assignments is $\#\mathcal{P}$-complete even in 2-CNF Horn formulae in which each variable appears at most 3 times, along with giving a number of polynomial-time algorithms to count satisfying assignments in other restricted types of 2-CNF formulae. Hunt et al. [17] showed that counting satisfying assignments to planar 3-CNF formulae is $\#\mathcal{P}$-complete, and gave a number of other counting problems that are hard in planar graphs, including counting minimum cardinality vertex covers.

The general theory of approximate counting was developed in the works of Stockmeyer [42], Karp and Luby [26], and Jerrum, Valiant, and Vazirani [24]. The first positive result for approximate counting that relates to our work is due to Karp and Luby [26], who gave a polynomial-time algorithm for approximately counting satisfying assignments to a DNF formula. After that, most of the positive results on approximate counting have come via the theory of "rapidly mixing Markov chains." In the first dramatic application of this approach to approximating a $\#\mathcal{P}$-complete counting problem, Jerrum and Sinclair [21] analyzed a Markov chain proposed by Broder [4] and thereby showed that it is possible to approximately count the number of perfect matchings in a graph of minimum vertex degree at least $n/2$ in polynomial time. They also gave an algorithm for approximately counting all matchings in an arbitrary graph. Their perfect matching algorithm was simplified by Dagum and Luby [8], who thereby obtained algorithms for approximately counting perfect matchings in αn-regular bipartite graphs for any constant α (and in fact a more general class of graphs).

For the counting problems we consider, the only previous inapproximability results involved approximately counting independent sets (equivalently, vertex covers or cliques) in general graphs [41, 51]. Specifically, Sinclair [41] used the "blow-up" technique introduced in [24] to show that it is $\mathcal{NP}$-hard to approximate the number of independent sets in a graph, even up to an approximation factor of $2^{n^{1-\varepsilon}}$, for any constant $\varepsilon > 0$. This result directly translates to the $\mathcal{NP}$-hardness of approximately counting the number of satisfying assignments to a monotone 2-CNF formula to within a factor of $2^{n^{1-\varepsilon}}$ [40] (cf. Proposition 3.1). Zuckerman [51], using techniques from the theory of probabilistically checkable proofs (PCP) [11, 2, 1], showed that it is hard to approximate arbitrarily iterated logarithms of the number of independent sets in a graph unless $\mathcal{NP}$ has slightly superpolynomial-time randomized algorithms.

Subsequent work. Since this work was done, there has been a substantial improvement in our understanding of counting independent sets (equivalently, vertex covers) in sparse graphs. With respect to exact counting, our results left a gap between the easy and hard cases; specifically, we showed that counting independent sets in graphs of maximal degree 4 is $\#\mathcal{P}$-complete, whereas the problem is polynomial-time solvable in graphs of maximal degree 2. For regular graphs, our hardness result only held for degrees ≥ 5. Greenhill [14] has closed these gaps, showing that counting independent sets in 3-regular graphs is $\#\mathcal{P}$-complete. We note that her hardness result uses ours as a starting point and also makes use of (generalizations of) our Fibonacci technique.

For approximately counting independent sets in sparse graphs, almost nothing was known at the time of this work. There were no polynomial-time approximation algorithms and no inapproximability results other than Sinclair's [41] and Zuckerman's [51] results for unrestricted graphs and our result about counting maximum cardinality independent sets. Luby and Vigoda [28] have remedied this situation in both respects. First, using the Markov chain approach, they have given a polynomial-time algorithm to approximately count independent sets in graphs of maximal degree at most 4. (Extensions and improvements can be found in [10, 29, 39].) Second, combining a blow-up technique of [41] with PCP-based inapproximability results [35, 2, 1], they proved that for some (large) constant Δ, approximately counting independent sets in graphs of maximal degree Δ is $\mathcal{NP}$-hard. Using a more sophisticated reduction and results in [16], Dyer, Frieze, and Jerrum [9] reduce the degree for this inapproximability result to $\Delta = 25$, and give evidence that the Markov chain approach is unlikely to work for any $\Delta \geq 6$. These results suggest that the PCP theorem, which has yielded many inapproximability results for optimization problems, may also be the right starting point for proving hardness of approximate counting.

Bubley and Dyer [6] have considered the problem of counting independent sets of a given size s in a graph of maximal degree Δ. Our results on counting maximum cardinality independent sets imply that the exact (resp., approximate) version of this problem is $\#\mathcal{P}$-complete (resp., $\mathcal{NP}$-hard) even when $\Delta = 3$, if there is no restriction on s. They show that in fact the approximate counting problem can be solved in polynomial time for $s < n/2(\Delta+1)+1$, whereas the exact counting problem remains $\#\mathcal{P}$-complete under this restriction.

In another work, Bubley and Dyer [5] have proven some new results about counting satisfying assignments in restricted classes of formulae. Instead of looking at 2-CNF formulae (as we do, and as happens when translating results about independent sets), they do not restrict the number of variables per clause, but only allow each variable to appear at most twice. They have shown that it is possible to efficiently approximate the number of satisfying assignments to such formulae. On the other hand, they show that exactly counting satisfying assignments remains $\#\mathcal{P}$-complete in CNF formula in which each literal appears exactly once, and monotone CNF formulae in which each variable appears at most twice. They also relate these versions of #Sat to counting "sink-free orientations" in directed graphs.

3. Preliminaries. Nearly all of the counting problems we will be considering are in Valiant's class $\#\mathcal{P}$, and the remainder are closely related to $\#\mathcal{P}$. Below, we informally review some basic definitions. For a more detailed discussion of $\#\mathcal{P}$, the reader is referred to any of [50, 45, 20, 34].

Following [24], $\#\mathcal{P}$ can be defined in terms of p-relations. Let Σ be a finite alphabet. A relation $R \subset \Sigma^* \times \Sigma^*$ is said to be a *p-relation* iff it is polynomially-balanced, i.e., there exists a polynomial p such that $\langle x, y\rangle \in R \Rightarrow |y| \leq p(|x|)$; and it can be "checked quickly," i.e., the language $L = \{\langle x, y\rangle \in R\}$ can be decided in polynomial time. The *counting problem* $\#R$ associated with R is the following: Given $x \in \Sigma^*$, output $|R(x)| = |\{y \in \Sigma^* : \langle x, y\rangle \in R\}|$. $\#\mathcal{P}$ is the class of all such counting problems.

In the above definition, x should be thought of as an instance of a problem, such as a boolean formula F, and $R(x)$ as the set of solutions associated with x, such as the satisfying assignments to F. It is easy to see that $\mathcal{NP}$ consists exactly of problems of the following form: Given x, decide whether $R(x)$ is nonempty. Thus $\#\mathcal{P}$ is the set of counting problems naturally associated with $\mathcal{NP}$ languages.

In contrast to $\mathcal{NP}$, it turns out that standard Karp reductions (i.e., polynomial-time many-one reductions) are not sufficient to describe the relative difficulty of counting problems. It is easy to construct counting problems which are obviously equivalent in difficulty, but for which there can be no one-to-one correspondence between solution sets. Hence, following [48], we consider a problem Π to be as hard as a problem Γ iff Γ can be solved by a polynomial-time algorithm with an oracle for Π, and we denote this by $\Gamma \propto \Pi$. Such a reduction is known as a *Cook reduction* (or *polynomial-time Turing reduction*), and this is the only form of reduction we will refer to in this paper. A problem Π is said to be *$\#\mathcal{P}$-hard* iff all problems in $\#\mathcal{P}$ reduce to it; if, in addition, $\Pi \in \#\mathcal{P}$ it is called *$\#\mathcal{P}$-complete*. Last, a problem is said to be *$\#\mathcal{P}$-easy* if it can be reduced to some problem in $\#\mathcal{P}$. Occasionally, we will be able to reduce one problem to another via a polynomial-time mapping of problem instances that preserves the number of solutions. Such a reduction is called a *parsimonious* reduction and these are important because they preserve inapproximability.

Having defined all the complexity-theoretic notions we will need, we proceed to define the combinatorial objects we will be studying. Let $G = (V, E)$ be an undirected graph. The *(maximum) degree* of G is the maximum number of edges incident to any vertex, and the *minimum degree* of G is defined similarly. A *vertex cover* in G is a subset S of V such that every edge in E has at least one endpoint in S. An *independent set* in G is a subset S of V such that no two vertices in S are connected by an edge in E. A *clique* in G is a subset S of V such that every two vertices in S are connected by an edge in E. It is well known that cliques, vertex covers, and independent sets are intimately related objects. Their relationship is formalized by the following proposition.

PROPOSITION 3.1. *Let $G = (V, E)$ be an undirected graph and let $\overline{G} = (V, \overline{E})$ be its (edge-)complement. Let F be the monotone 2-CNF formula on variables V given by $F = \bigwedge_{(u,v)\in E}(u \vee v)$. For $S \subset V$, let $\chi_S : V \to \{0, 1\}$ be the assignment which maps $v \in V$ to 1 iff $v \in S$. Then the correspondence $S \leftrightarrow (V - S) \leftrightarrow (V - S) \leftrightarrow \chi_S$ establishes bijections between the vertex covers in G, the independent sets in G, the cliques in $\overline{G}$, and the satisfying assignments of F.*

Proof. The proof is by definition. □

We will also be examining the complexity of these problems in bipartite graphs. However, the study of cliques in bipartite graphs is not very interesting, as the only cliques are edges. So, for a bipartite graph $G = (V, E)$, we will instead look at *bipartite cliques*, which are subsets $S \subset V$ of vertices which can be partitioned $S = S_1 \cup S_2$ so that $S_1 \times S_2 \subset E$. To obtain an analogue of Proposition 3.1, we say that bipartite graphs $G = (V, E)$ and $H = (V, F)$ are *bipartite complements* if V can be partitioned $V = V_1 \cup V_2$ such that $E \subset V_1 \times V_2$ and $F = V_1 \times V_2 \setminus E$. Note that a bipartite complement of a graph can be found in polynomial time, and is unique if the graph is connected. Proposition 3.1 has the following bipartite analogue.

PROPOSITION 3.2. *Let bipartite graphs $G = (V, E)$ and $H = (V, F)$ be bipartite complements. Then $S \subset V$ is an independent set in G iff it is a bipartite clique in H.*

Proof. The proof is by definition. □

The above propositions will enable us to immediately deduce hardness results for all of the above problems given a hardness result for one of them. Therefore, we will concentrate primarily on the vertex cover problem.

We will also study extremal variants of all of the above problems. A vertex cover S is said to be *minimal* iff no proper subset of S is a vertex cover. It is said to be of *minimum cardinality* iff there is no vertex cover with fewer vertices. Similarly, we

speak of maximal and maximum cardinality independent sets or cliques.

By Proposition 3.1, it is easy to see that minimal vertex covers correspond to *minterms* of a monotone 2-CNF formula — that is, satisfying assignments for which changing any variable from true to false would no longer satisfy the formula. It is clear that the smallest DNF form for a monotone formula F is simply the disjunction of all minterms, writing an individual minterm M as the conjunction of the variables in M. Hence, restricted-case hardness results for counting minimal vertex covers immediately imply that determining the size of the minimal DNF form is hard even for restricted classes of CNF formulae.

It is clear that if the decision problem associated with a p-relation is $\mathcal{NP}$-complete, then the associated counting problem is also $\mathcal{NP}$-hard. Thus, complexity of counting results are only interesting when the related decision problem is easy. The first nontrivial result of this form, due to Valiant, involved another type of graph-theoretic structure, known as a perfect matching. A *matching* in an undirected graph $G = (V, E)$ is a set $M \subset E$ of edges, no two of which share an endpoint. A *perfect matching* is a matching M in which every vertex in V is the endpoint of an edge in M. Valiant's theorem [48] states that counting perfect matchings in bipartite graphs is $\#\mathcal{P}$-complete. Matchings can be related to the other structures we mentioned via the following construction.

Let $G = (V, E)$ be an undirected graph. The *line graph of* G is the undirected graph $L(G) = (E, H)$, where $(e_1, e_2) \in H$ iff e_1 and e_2 share an endpoint in G.

LEMMA 3.3. *Let $G = (V, E)$ be an undirected graph. Then $M \leftrightarrow E - M$ establishes a bijective correspondence between matchings in G and vertex covers in the line graph of G.*

Proof. Notice that $M \subset E$ is a matching in G iff M is an independent set in $L(G)$. The relationship with vertex covers follows from Proposition 3.1. □

One of the restricted classes of graphs that we will examine is the class of planar graphs. A graph is said to be *planar* iff there exists an embedding of the graph in the plane (where the vertices are points and the edges are curves connecting the points) in which no two edges intersect. The bijection of Proposition 3.1 suggests how to define planarity for CNF formulae. If F is a formula in conjunctive normal form, we define $G(F)$ to be the graph whose vertices are the variables of F, where two vertices are connected if they lie in a common clause. We call F *planar* iff $G(F)$ is planar.

There are several reasons for studying the problems described above. One is that many $\mathcal{NP}$-completeness results have come via reduction from decision versions of these problems, so it reasonable to guess that many restricted-case $\#\mathcal{P}$-completeness results could come via reduction from restricted versions of these counting problems. Another reason is that these problems are closely related to important problems in other areas. As discussed in the introduction, several problems in artificial intelligence are equivalent to counting satisfying assignments to a propositional formula [33, 36, 40]. In addition, the counting problems we consider arise naturally in statistical physics; specifically, counting matchings amounts to counting arrangements of monomer-dimer systems, and counting independent sets is tantamount to computing the partition function in the hard-core model of a gas (cf. [15, 19, 29]).

For their simplicity, their relationships to other important problems, and their potential to serve as starting points for further results, the problems of counting matchings, vertex covers, and independent sets are important ones to consider. So motivated, we now proceed to classify the cases in which these problems are and are not tractable.

4. Formal statement of results. Before stating our negative results, we explain the entries in the "Polynomial-time solvable" columns of Tables 1.1 and 1.2 for which no reference is given. The tractability of problems such as counting matchings, vertex covers, and their variants in graphs of maximal degree at most 2 follows from the simple structure of these graphs: The connected components of such graphs are cycles and chains, and objects such as matchings and vertex covers multiply across connected components, so it suffices to count them in cycles and chains. It is not hard to write down closed forms or recurrences which can be used to count these structures in chains and cycles. This is done explicitly in [45]. The polynomial-time solvability of counting cliques in graphs of maximal degree at most Δ follows from the fact that the largest clique in such a graph is of size at most Δ. Thus, one can exhaustively check the $\leq n^{\Delta}$ possibilities in polynomial time. Similarly, a planar graph cannot contain any clique of size ≥ 5, nor any bipartite clique involving at least 3 vertices on both sides of the partition, as these structures are nonplanar (cf. [44, section 1.4]). Finally, as noted earlier, the only cliques in a bipartite graph are singletons and pairs. All the other entries follow from the above observations via Propositions 3.1 and 3.2.

We now state our negative results. Below, the counting problems are named as follows: the beginning of the name indicates any restrictions on the input graph or formula, and the end of the name contains the structures to be counted. A prefix of $k\Delta$ indicates that the maximum degree of the graph is at most k. An example of a problem denoted this way is the following.

#7Δ-Planar Bipartite Maximal Matchings.

Input: A planar bipartite graph G of degree ≤ 7.

Output: The number of maximal matchings in G.

Theorem 4.1. *The following problems are #$\mathcal{P}$-complete (except for Problem* 8*, which is #$\mathcal{P}$-hard*[1]*):*

1. #4Δ-Bipartite Matchings,
2. #6Δ-Planar Bipartite Matchings,
3. #k-Regular Matchings, *any fixed* $k \geq 5$,
4. #5Δ-Bipartite Maximal Matchings,
5. #7Δ-Planar Bipartite Maximal Matchings,
6. #Planar Bipartite Vertex Covers,
7. #3Δ-Planar Bipartite Minimum Cardinality Vertex Covers,
8. #k-Regular Minimum Cardinality Vertex Covers, *any fixed* $k \geq 4$,
9. #4Δ-Planar Bipartite Vertex Covers,
10. #k-Regular Vertex Covers, *any fixed* $k \geq 5$,
11. #5Δ-Planar Bipartite Minimal Vertex Covers,
12. #Regular Minimal Vertex Covers.

In the following corollary, we use the same naming conventions as above, with some additional ones. When Bipartite Cliques appears, it refers to counting bipartite cliques in bipartite graphs. A prefix of $k\delta$ means that the minimum vertex degree is at least k. n refers to the number of vertices in the graph, except when the input is restricted to be bipartite, in which case n is the number of vertices on each side of the bipartition.

[1]Problem 8 is not likely to be in #$\mathcal{P}$, because testing whether a vertex cover is of minimum cardinality is $\mathcal{NP}$-hard [13, Theorem 3.3]. But we will reduce it to a #$\mathcal{P}$ problem, proving that it is #$\mathcal{P}$-easy. The failure of #$\mathcal{P}$ to be closed under reductions and even simpler operations is discussed in [32].

In contrast, Problem 7 is in #$\mathcal{P}$ because in *bipartite graphs*, the size of the minimum cardinality vertex cover equals the size of the maximum cardinality matching. The latter quantity can be found using any of the standard maximum cardinality matching algorithms. See [34, Problem 9.5.25].

For problems involving satisfying assignments, 2SAT denotes the problem of counting the satisfying assignments of the input formula F, MINTERMS the problem of counting the minterms of F (i.e., the number of clauses in the smallest DNF formula equivalent to F), and MIN WEIGHT 2SAT the problem of counting satisfying assignments with the fewest variables set to true. A prefix of $k\Delta$ (resp., k-REGULAR) indicates that each variable appears at most (resp., exactly) k times. A *bipartite* 2-CNF formula is one in which the variables can be partitioned into two sets such that no clause contains two variables from the same set. For example, we have the following.

#3Δ-PLANAR BIPARTITE MONOTONE MIN WEIGHT 2SAT.

Input: A planar monotone formula $F = (c_{1,1} \vee c_{1,2}) \wedge \cdots \wedge (c_{r,1} \vee c_{r,2})$ on variables $X \cup Y$, where $X \cap Y = \emptyset$; $c_{i,1} \in X$, $c_{i,2} \in Y$ for each i; and each variable appears at most 3 times.

Output: The number of satisfying assignments to F with the fewest variables set to "true."

COROLLARY 4.2. *The following problems are #$\mathcal{P}$-complete (except for Problems* 2, 9, *and* 17, *which are #$\mathcal{P}$-hard):*

1. #3Δ-PLANAR BIPARTITE MAXIMUM CARDINALITY INDEPENDENT SETS,
2. #k-REGULAR MAXIMUM CARDINALITY INDEPENDENT SETS, *any fixed* $k \geq 4$,
3. #4Δ-PLANAR BIPARTITE INDEPENDENT SETS,
4. #k-REGULAR INDEPENDENT SETS, *any fixed* $k \geq 5$,
5. #5Δ-PLANAR BIPARTITE MAXIMAL INDEPENDENT SETS,
6. #REGULAR MAXIMAL INDEPENDENT SETS,
7. #$(n-3)\delta$-MAXIMUM CARDINALITY CLIQUES,
8. #$(n-3)\delta$-MAXIMUM CARDINALITY BIPARTITE CLIQUES,
9. #$(n-k)$-REGULAR MAXIMUM CARDINALITY CLIQUES, *any fixed* $k \geq 4$,
10. #$(n-4)\delta$-CLIQUES,
11. #$(n-4)\delta$-BIPARTITE CLIQUES,
12. #$(n-k)$-REGULAR CLIQUES, *any fixed* $k \geq 5$,
13. #$(n-5)\delta$-MAXIMAL CLIQUES,
14. #$(n-5)\delta$-MAXIMAL BIPARTITE CLIQUES,
15. #REGULAR MAXIMAL CLIQUES,
16. #3Δ-PLANAR BIPARTITE MONOTONE MIN WEIGHT 2SAT,
17. #k-REGULAR MONOTONE MIN WEIGHT 2SAT, *any fixed* $k \geq 5$,
18. #4Δ-PLANAR BIPARTITE MONOTONE 2SAT,
19. #k-REGULAR MONOTONE 2SAT, *any fixed* $k \geq 5$,
20. #5Δ-PLANAR BIPARTITE MINTERMS,
21. #REGULAR MINTERMS.

The following proposition contains our inapproximability result. The prefix of f-APPROX indicates the problem of solving the given counting problem within a multiplicative approximation factor of f. Though we have adopted Cook reductions as our notion of reducibility, the following results are actually proved via Karp reductions to "gap promise problems" (cf. [3]).

PROPOSITION 4.3. *The following problems are $\mathcal{NP}$-hard for every* $\epsilon > 0$*:*

1. $2^{n^{1-\epsilon}}$-APPROX #3Δ-MINIMUM CARDINALITY VERTEX COVERS,
2. $2^{n^{1-\epsilon}}$-APPROX #3Δ-MAXIMUM CARDINALITY INDEPENDENT SETS,
3. $2^{n^{1-\epsilon}}$-APPROX #$(n-3)\delta$-MAXIMUM CARDINALITY CLIQUES,
4. $2^{n^{1-\epsilon}}$-APPROX #3Δ-MIN WEIGHT MONOTONE 2SAT.

5. The reductions. We state three facts about polynomial interpolation here that we will use repeatedly in our reductions. Let K be a finite extension of $\mathbb{Q}$. For any $z \in K$, let $\|z\|$ denote the number of bits needed to represent z. (If K is a degree d extension, elements of K are represented by polynomials of degree $\leq d-1$ with rational coefficients, and arithmetic is done modulo some irreducible polynomial defining K.) For a polynomial f in several variables over K, let $\|f\|$ be the number of bits needed to represent f, which is the sum of $\|a\|$ for the coefficients a of f. We use a *dense* representation of polynomials, which means that if some monomial $x_1^{d_1} \cdot x_2^{d_2} \cdots x_n^{d_n}$ has a nonzero coefficient in f, then the coefficients of all smaller monomials (i.e., any $x_1^{e_1} \cdots x_n^{e_n}$ such that $e_i \leq d_i$ for all i) must be included when computing $\|f\|$. (Zero coefficients count as one bit). For a rational function $q = f/g$, let $\|q\| = \|f\| + \|g\|$.

FACT 5.1. *Let $F = K(y_1, y_2, \ldots, y_k)$ be the field of rational functions over K in k variables for some constant k. Let $f(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial with coefficients in F. If $(\alpha_0, \beta_0), \ldots, (\alpha_d, \beta_d)$ such that $f(\alpha_i) = \beta_i$ are known for distinct $\alpha_i \in F$, then the coefficients of f can be recovered in time polynomial in $\max_i \|\alpha_i\|$, $\max_i \|\beta_i\|$, and d.*

FACT 5.2. *Let $f(x, y) = \sum_{i+j \leq n} a_{ij} x^i y^j$ be a polynomial in two variables with coefficients in K. If for each of $n+1$ distinct $x_i \in K$, $n+1$ distinct $y_{ij} \in K$ along with the values $z_{ij} = f(x_i, y_{ij})$ are known, then the coefficients of f can be recovered in time polynomial in $\max_{ij} \|a_{ij}\|, \max_i \|x_i\|, \max_{ij} \|y_{ij}\|$, and $\max_{ij} \|z_{ij}\|$.*

FACT 5.3. *Let $f(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial with nonnegative integer coefficients. If (α, β) is known, where $f(\alpha) = \beta$ and α is a rational number satisfying $\alpha \geq (a_i + 1)$ for each i, then the coefficients of f can be recovered in time polynomial in $\|\alpha\|$, $\|\beta\|$, and d.*

Proof of Fact 5.1. Here we just use the Lagrange interpolation formula:

$$f(x) = \sum_{i=0}^{d} \beta_i \left(\frac{(x-\alpha_1)\cdots(x-\alpha_{i-1})(x-\alpha_{i+1})\cdots(x-\alpha_d)}{(\alpha_i-\alpha_1)\cdots(\alpha_i-\alpha_{i-1})(\alpha_i-\alpha_{i+1})\cdots(\alpha_i-\alpha_d)} \right).$$

This is a polynomial which agrees with f at $d+1$ distinct points, so it must be the same polynomial. By multiplying out and collecting terms, we can obtain the coefficients of f all at once, in polynomial time. □

Proof of Fact 5.2. Define, for each $0 \leq r \leq n$, $g_r(y) = \sum_{i,j} a_{ij} x_r^i y^j$. For each r, we know the evaluation of $g_r(y)$ at the $n+1$ points $y_{r0}, \ldots, y_{rn}$, so we can recover the polynomials $g_r(y)$ by Fact 5.1. These are the evaluations of $f(x, y)$, considered as a polynomial in x with coefficients in $K(y)$, at the points $x_0, \ldots, x_n$. By Fact 5.1, we can recover the coefficients of f. □

Proof of Fact 5.3. All we need to do is write β as a number in base α and the digits are our coefficients. In more detail, note that $\sum_{i=0}^{d-1} a_i \alpha^i \leq \sum_{i=0}^{d-1} (\alpha - 1)\alpha^i = \alpha^d - 1 < \alpha^d$, so $a_d = \lfloor \beta / \alpha^d \rfloor$, which we can obtain quickly by just integer multiplication and division. Now consider $f_1(x) = f(x) - a_d x^d$. We can repeat this process for f_1, obtaining the coefficients in sequence. □

Proof of Theorem 4.1.

1. #4Δ-BIPARTITE MATCHINGS. The reduction given in the introduction reduces #PERFECT MATCHINGS to #MATCHINGS while only increasing the degree by 1 and preserving bipartiteness. Dagum and Luby [8] have proven the former to be #$\mathcal{P}$-complete in 3-regular bipartite graphs; our result follows. (The facts about matchings in chains and Fibonacci sequences needed for the reduction are given by Lemma 6.3.)

2. #6Δ-Planar Bipartite Matchings. The starting point for this proof is the work of Jerrum [19] which shows that counting matchings in planar graphs is $\#\mathcal{P}$-complete. As is, his reduction produces graphs that are neither bipartite nor of bounded degree. We show how an additional step added in the middle of his reduction can transform the graphs produced into bipartite ones. We then show how a reduction like the one in Reduction 1 can replace the final step of his reduction so that the degree does not blow up.

In the course of his reduction, Jerrum considers a weighted form of #Matchings: Let $G = (V, E)$ be a graph in which each vertex $v \in V$ is assigned a weight $w(v) \in \mathbb{C}$. If $M \subset E$ is a matching in G, then we let $C(M)$ be the set of vertices in V which are covered by M, i.e., are endpoints of edges in M. Then the *weight* of M is $w(M) = \prod_{v \notin C(M)} w(v)$, i.e., the product of the weights of all vertices *not* covered by M; for a perfect matching M, $w(M) = 1$. The *weighted matching sum* $W(G)$ of G is $\sum_M w(M)$, where this sum is taken over all matchings in G. Thus if all vertices have weight 1, $W(G)$ is simply the number of matchings in G. We say that G has k *weights* if the number of distinct values other than 1 occurring as weights in G is at most k. The #$k\lambda$-Weighted Matchings problem is the following.

#$k\lambda$-Weighted Matchings.

Input: A vertex-weighted graph G with k weights.

Output: $W(G)$.

As usual, we also consider variants of this problem for bipartite, planar, and bounded degree graphs. The prefix $d\Delta$ is used to restrict to graphs of degree at most d. Our reduction will proceed in three stages. The first will show that computing $W(G)$ is hard for planar graphs of bounded degree. The second will transform any graph into a bipartite graph without changing $W(G)$, losing planarity, or increasing the degree. The third will remove weights one by one without losing any of the graph properties, showing that counting matchings in bipartite planar graphs of bounded degree is hard.

2a. #3-Regular Perfect Matchings $\propto$ #5Δ-2λ-Planar Weighted Matchings. Jerrum [19] gives a reduction from #Perfect Matchings to #2λ-Planar Weighted Matchings. We observe that his reduction produces a graph of degree 5 when applied to a graph of degree 3.

2b. #5Δ-2λ-Planar Weighted Matchings $\propto$ #5Δ-4λ-Planar Bipartite Weighted Matchings. Consider the weighted graph H with vertex set $\{v_1, v_2, v_3, v_4\}$, edges (v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4), and vertex weights $w(v_1) = w(v_4) = 1$, $w(v_2) = \zeta$, $w(v_3) = \zeta^2$, where $\zeta = e^{2\pi i/3}$ is a primitive cube root of unity. (See Figure 5.1.) Straightforward calculations show the following:

1. $W(H) = W(H \setminus \{v_1, v_4\}) = 1$,
2. $W(H \setminus \{v_1\}) = W(H \setminus \{v_4\}) = 0$.

Above, the notation $H \setminus S$ denotes the graph formed by removing from H all vertices in S and any edges incident to them.

Now let G be a planar weighted graph with 2 weights and let $e = (u, v)$ be any edge in G. Consider the graph G' obtained from G by removing edge e; adding a disjoint copy of H; and adding edges $e_1 = (u, v_1)$ and $e_2 = (v, v_4)$. (See Figure 5.2.) We will show that $W(G') = W(G)$. Let M be any matching in G' that does not contain any of the edges of H. Then observation 2 above tells us that if M contains exactly one of e_1 and e_2, the net contribution to $W(G')$ of all matchings formed by adding H-edges to M will be zero. If M contains neither e_1 nor e_2, then, since $w(H) = 1$, the net contribution of all matchings formed by adding H-edges to M will

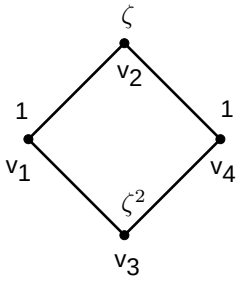


FIG. 5.1. *The graph H in Reduction 2b.*

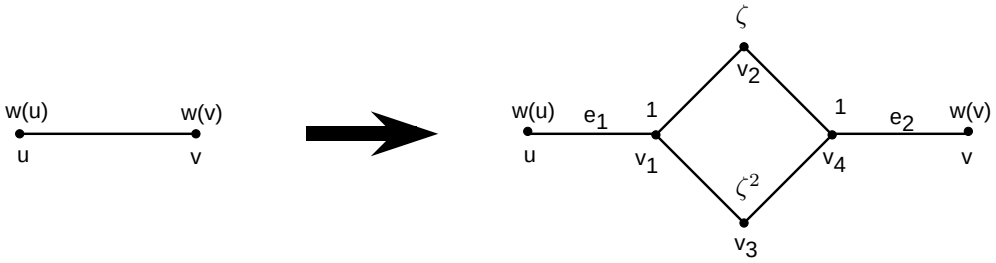


FIG. 5.2. *The transformation of Reduction 2b.*

be $\prod w(x)$ where the product is taken over all vertices $x \notin C(M) \cup \{v_1, v_2, v_3, v_4\}$. This is the same as the G-weight of M. Similarly, since $w(H \setminus \{v_1, v_4\}) = 1$, if M contains both e_1 and e_2 the net contribution will be the same as the G-weight of $(M - \{e_1, e_2\}) \cup \{e\}$. Thus, the net weight of all matchings in G' accounts exactly for the net weight of all matchings in G and $W(G') = W(G)$. Therefore, if we do this procedure to all edges in G, we will end up with a planar bipartite graph $\hat{G}$ with 4 weights (ζ and ζ^2 are the only new weights) such that $W(\hat{G}) = W(G)$. It is clear that this reduction can be carried out in polynomial time.

In the next part of the reduction, we remove weights one by one, only increasing the degree by 1. We use the prefix $k\mu$ to restrict to graphs with k weights in which all vertices with weight 1 have degree at most 6 and all other vertices have degree at most 5. In particular, an instance of #5Δ-4λ-PLANAR BIPARTITE WEIGHTED MATCHINGS satisfies this condition for $k = 4$.

2c. #$k\mu$-PLANAR BIPARTITE WEIGHTED MATCHINGS $\propto$ #$(k-1)\mu$-PLANAR BIPARTITE WEIGHTED MATCHINGS. Jerrum [19] gives a reduction which removes weights one by one, but his reduction blows up the degree. To replace his reduction, we use the Fibonacci technique.

Let G be a bipartite planar graph with k weights, in which all vertices with weight 1 have degree at most 6 and all other vertices have degree at most 5. Let $\alpha \neq 1$ be a vertex weight that occurs in G and let $v_1, \ldots, v_m$ be the vertices with weight α. For

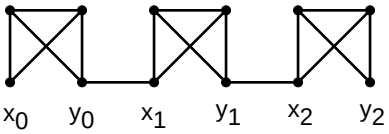


FIG. 5.3. *The graph L_2^3 in Reduction* 3.

each $s = 0, \ldots, m$, construct a graph G_s as follows: Add nodes $v_{i,j}$ to G for $1 \le i \le m$. Add edges to create disjoint chains $v_i - v_{i,1} - v_{i,2} - \cdots - v_{i,s}$ of $s+1$ nodes. Assign each of the new vertices weight 1, assign $v_1, \ldots, v_m$ weight 1, and keep all other vertex weights the same as in G. Notice that G_s is a valid instance of $\#(k-1)\mu$-PLANAR BIPARTITE WEIGHTED MATCHINGS.

Now, for every matching M in G, we can obtain matchings in G_s by adding some of the new edges to M. Suppose M covers exactly i of $v_1, \ldots, v_m$. Then the net contribution to $W(G_s)$ of the matchings formed by adding new edges to M is $x_s^i (x_{s+1}/\alpha)^{m-i} w_G(M)$, where x_t is the number of matchings in a chain of t vertices. Let $A_i = \sum w_G(M)$, where the sum is taken over all matchings in G which match exactly i of $v_1, \ldots, v_m$. Then $W(G_s) = (x_{s+1}/\alpha)^m \sum_{i=0}^m A_i (\alpha x_s / x_{s+1})^i$. As in the original application of the Fibonacci technique, we can recover the coefficients A_i with $m+1$ oracle calls and polynomial interpolation, using Lemma 6.3 to verify that the evaluation points are distinct. Then it is easy to compute $W(G) = \sum_{i=0}^m A_i$, as desired.

It may seem odd that the graphs G_s constructed in this reduction do not depend on the value of α. This is because this reduction works even if α is regarded as an indeterminate—$W(G)$ is then a polynomial in α, and we are essentially recovering this polynomial. Finally, note that $\#0\mu$-PLANAR BIPARTITE WEIGHTED MATCHINGS is exactly $\#6\Delta$-PLANAR BIPARTITE MATCHINGS.

3. $\#k$-REGULAR MATCHINGS, any fixed $k \ge 5$. We reduce from $\#(k-2)$-REGULAR BIPARTITE PERFECT MATCHINGS, which was shown to be $\#\mathcal{P}$-complete by Dagum and Luby [8]. Let H be the complete graph on $k+1$ vertices with one edge removed. Consider the graph L_s^k formed by taking $s+1$ disjoint copies of H, labelled $H_0, \ldots, H_s$, and attaching y_i to x_{i+1} for each $0 \le i < s$, where x_i and y_i are the two vertices in H_i of degree $k-1$. (See Figure 5.3.)

Now let G be any $(k-2)$-regular graph on n nodes $u_1, u_2, \ldots, u_n$. For any $0 \le s \le n$, consider the disjoint union of G with n copies of L_s^k. Let the vertices of degree $k-1$ in the ith copy of L_s^k be labelled v_i and w_i. We can form a k-regular graph G_s by connecting u_i to both v_i and w_i for each i. Let A_i be the number of matchings in G in which exactly i vertices are unmatched. Then the number of matchings in G_s is $\sum_{i=0}^n A_i (x_s + 4y_s + 3z_s)^i (x_s + 2y_s + z_s)^{n-i}$, where x_s is the number of matchings in L_s^k in which v and w—the two vertices of degree $k-1$—are both matched, y_s is the number in which v but not w is matched, and z_s is the number in which neither v nor w is matched. By Lemma 6.4, we can compute x_s, y_s, and z_s in polynomial time and the sequence $(x_s + 4y_s + 3z_s)/(x_s + 2y_s + z_s) = 1 + 2(y_s + z_s)/(x_s + 2y_s + z_s)$ never repeats. So, with $n+1$ oracle calls and interpolation, we can recover A_0, the number of perfect matchings in G.

4. $\#4\Delta$-BIPARTITE MATCHINGS$\propto$ $\#5\Delta$-BIPARTITE MAXIMAL MATCHINGS. The reduction we use is from [38], though they use it for vertex cover problems. Given a bipartite undirected graph $G = (V, E)$ of degree ≤ 4, construct $G' = (V', E')$, where $V' = V \cup \{v' : v \in V\}$ and $E' = E \cup \{(v, v') : v \in V\}$. Note that every matching W

in G yields a unique maximal matching $W' = W \cup \{(v, v') : v \text{ not matched by } W\}$ in G' and every maximal matching of G' can be obtained in this fashion.

5. #6Δ-Planar Bipartite Matchings$\propto$ #7Δ-Planar Bipartite Maximal Matchings. Notice that Reduction 4 preserves planarity, so it applies here, too.

6. #Planar Bipartite Vertex Covers. This result will come via a sequence of reductions beginning with #Vertex Covers, whose #$\mathcal{P}$-completeness follows immediately from Lemma 3.3 and the #$\mathcal{P}$-completeness of #Matchings. In the spirit of Reduction 2, the intermediate problems involve a generalization of vertex covers which we call *edge-weighted sets*. Suppose G is a graph in which each edge $e = (u, v)$ is labelled with a triple $d_e/s_e/n_e$ and S is a set of vertices in G. Then define the *weight of S with respect to e* as

$$w_e(S) = \begin{cases} d_e & \text{if } |\{u, v\} \cap S| = 2, \\ s_e & \text{if } |\{u, v\} \cap S| = 1, \\ n_e & \text{if } |\{u, v\} \cap S| = \emptyset, \end{cases}$$

so that d_e, s_e, and n_e correspond to weights if e is "doubly covered," "singly covered," or "not covered" by S, respectively. The *weight of S with respect to G* is then defined as $w_G(S) = \prod_e w_e(S)$. We now define the *edge-weighted sum* $\mathrm{EW}(G)$ to be $\sum_{S \subseteq V} w_G(S)$. Notice that if all the edges in G are labelled 1/1/0, then $\mathrm{EW}(G)$ is simply the number of vertex covers in G, so we have indeed generalized #Vertex Covers. With this in mind, we call an edge of such a labelled graph *normal* if its label is 1/1/0. For technical reasons, we will restrict to graphs with only a constant number of distinct labels. We will say a graph labelled as above has k *labels* if the number of distinct labels other than 1/1/0 is at most k.

Our first aim in reducing #Vertex Covers to planar graphs is to simplify the types of graphs we deal with. We call an embedding of a labelled graph G in the plane *simple* iff only normal edges are involved in crossings and each edge is in at most one crossing. Consider the following computational problem:

$k\lambda$-Simple Edge-Weighted Sum.

Input: A labelled graph G with k labels and a simple embedding of G in the plane.

Output: $\mathrm{EW}(G)$.

We now reduce #Vertex Covers to this problem.

6a. #Vertex Covers $\propto$ #1λ-Simple Edge-Weighted Sum. Let m be the number of edges in $G = (V, E)$. For any $s \geq 2m$ and any $t \geq 0$ consider the graph $G_{s,t} = (V_{s,t}, E_{s,t})$ formed by removing each edge $e = (u, v)$ of G and replacing it with the $s + 2$ vertices u^e, v^e, and $w_1^e, \ldots, w_s^e$. Also add the edges (u, u^e), (v, v^e), (u, w_1^e), (v, w_s^e), and (w_i^e, w_{i+1}^e) for $i = 1, \ldots, s - 1$. Furthermore, label the edges (u, u^e) and (v, v^e) with the label $(tF_{s-1}/F_s)/0/1$, where F_s is the sth Fibonacci number, as defined in Lemma 6.3. Label all other edges 1/1/0. Notice that it is easy to obtain a simple embedding of $G_{s,t}$ from any embedding of G in the plane, as we have broken each edge of G into $\geq 2m$ pieces and the edges of the form (u, u^e) clearly make no difference.

For each set of vertices S in G, let us consider the (weighted) number of ways that we may extend S to $G_{s,t}$, i.e., let us compute

$$\sum_{T : T \cap V = S} w_{G_{s,t}}(T).$$

If an edge $e = (u, v)$ of G is doubly covered by S, then there are $x_s(tF_{s-1}/F_s)^2$ (weighted) ways of adding vertices u^e, v^e, and $w_1^e, \ldots, w_s^e$ to S, where x_k is the

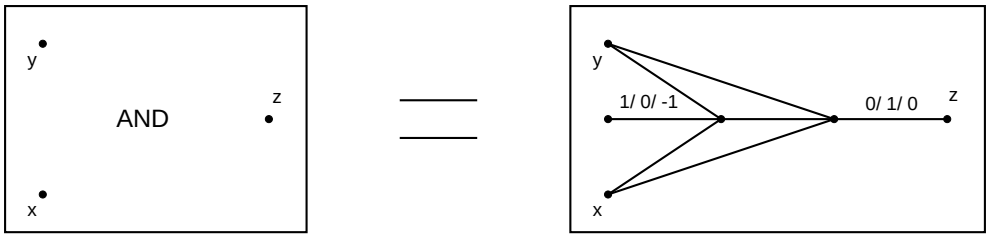


FIG. 5.4. *The AND gadget in Reduction* 6b.

number of vertex covers in a chain of k vertices. If e is singly covered by S, then there are $x_{s-1}tF_{s-1}/F_s$ ways to add these vertices. Finally, if e is not covered by S, there are x_{s-2} ways. Thus, if A_{ij} is the number of subsets of V which doubly cover i edges in G and which singly cover j edges, then

$$\mathrm{EW}(G_{s,t}) = \sum_{i,j} A_{ij}(x_s(tF_{s-1}/F_s)^2)^i(x_{s-1}tF_{s-1}/F_s)^j(x_{s-2})^{m-i-j}.$$

Writing r_k for F_k/F_{k-1} and using the fact that $x_k = F_{k+1}$ (from Lemma 6.3), we get

$$\mathrm{EW}(G_{s,t}) = \sum_{i,j} A_{ij}((F_{s+1}/F_{s-1})(t/r_s)^2)^i t^j F_{s-1}^m.$$

Using the relation $r_{s+1} = 1 + 1/r_s$, we get $F_{s+1}/F_{s-1} = r_{s+1}r_s = r_s + 1$. Substituting this above, we get

$$\mathrm{EW}(G_{s,t}) = F_{s-1}^m \sum_{i,j} A_{ij}(t^2(r_s^{-1} + r_s^{-2}))^i t^j.$$

Lemma 6.3 tells us that the sequence $\{r_s\}$ does not repeat. Since $x^{-1} + x^{-2} \neq y^{-1} + y^{-2}$ for any two distinct positive real numbers x and y, the sequence $\{r_s^{-1} + r_s^{-2}\}$ also does not repeat. Thus, by Fact 5.2, evaluating $\mathrm{EW}(G_{s,t})$ for each $t = 0, \ldots, m$ and for each $s = 2m, \ldots, 3m+1$ enables us to recover the coefficients A_{ij}. $\sum_{i+j=m} A_{ij}$ is the number of vertex covers of G.

We have reduced the problem to dealing with simple embeddings of graphs; the next step is to the problem of computing EW for planar graphs, as defined below:

$k\lambda$-PLANAR EDGE-WEIGHTED SUM.

Input: A labelled graph G with k labels and a planar embedding of G.

Output: $\mathrm{EW}(G)$.

The aim of the next reduction will be to replace crossings with planar gadgets without changing the value of $EW(G)$.

6b. $\#1\lambda$-SIMPLE EDGE-WEIGHTED SUM $\propto$ $\#5\lambda$-PLANAR EDGE-WEIGHTED SUM. First we make planar gadgets to compute elementary boolean formulae. We will write sets of vertices as functions from the set of all vertices to $\{0, 1\}$, where $S(v) = 1$ indicates that $v \in S$ and $S(v) = 0$ indicates that $v \notin S$. Consider the AND gadget in Figure 5.4.

It is easy to see that for any $a, b, c \in \{0, 1\}$,

$$\sum_{S:S(x)=a,S(y)=b,S(z)=c} w_{\mathrm{AND}}(S) = \begin{cases} 1 & \text{if } c = a \wedge b, \\ 0 & \text{otherwise.} \end{cases}$$

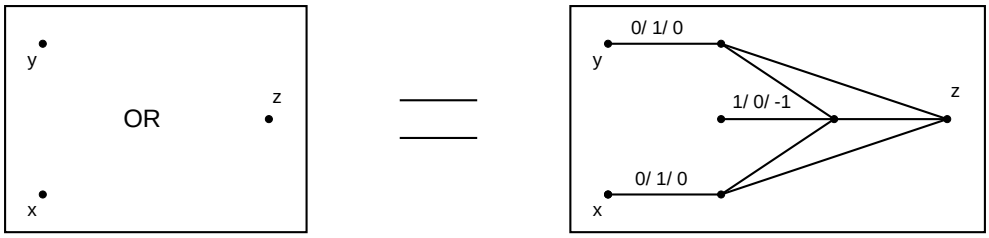


FIG. 5.5. *The OR gadget in Reduction 6b.*

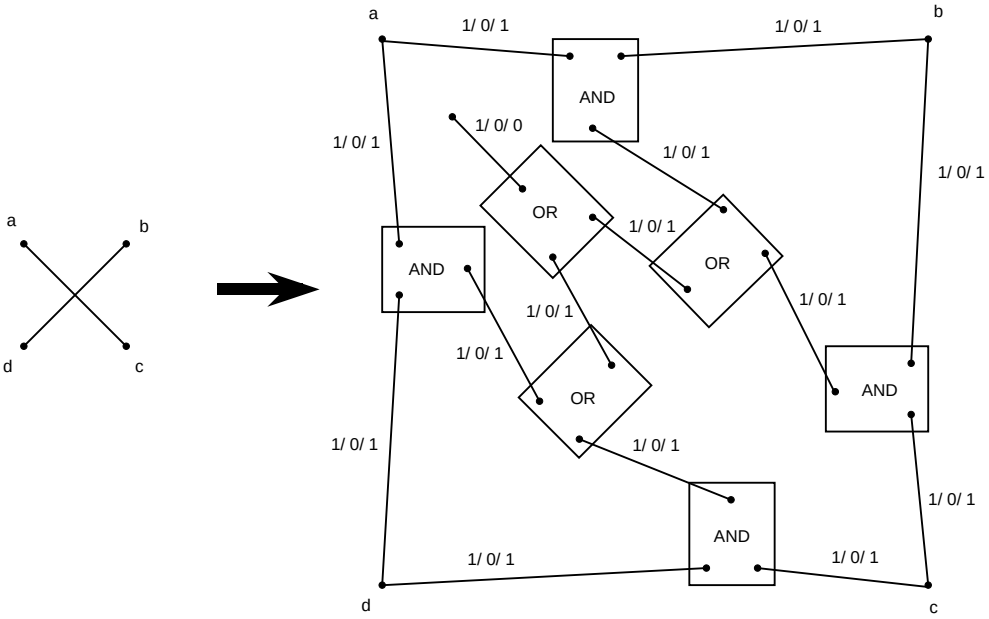


FIG. 5.6. *Replacing crossings in Reduction 6b.*

Thus, this gadget "forces" $S(z)$ to be $S(x) \wedge S(y)$. Similarly, the OR gadget of Figure 5.5 forces $S(z)$ to be $S(x) \vee S(y)$. Observe that an edge labelled 1/0/1 forces its endpoints to take on the same value and an edge labelled 1/0/0 forces both its endpoints to take on the value 1. The AND and OR gadgets constructed above along with these observations, enable us to form a complex gadget to replace crossings: Take any simple embedding of $G = (V, E)$ in the plane. Consider any two (normal) edges $e_1 = (a, c)$ and $e_2 = (b, d)$ which cross, where a, b, c, d is the order of the endpoints going clockwise around the crossing starting with a. Let G' be the graph with these edges removed. For any $S \subset V$, $w_{G'}(S) = w_G(S)$ if S contains at least one endpoint of both e_1 and e_2 and $w_G(S) = 0$ otherwise. The key observation is that S contains at least one endpoint of both e_1 and e_2 iff S contains some pair of vertices in $\{a, b, c, d\}$ which are *adjacent*, when these vertices are considered in clockwise order. Thus, if we replace the crossing with a gadget which simply forces $(S(a) \wedge S(b)) \vee (S(b) \wedge S(c)) \vee (S(c) \wedge S(d)) \vee (S(d) \wedge S(a)) = 1$, then the edge-weighted sum does not change. This can be done with the planar gadget of Figure 5.6.

If we replace all crossings of G in this manner, we obtain a planar labelled graph G' such that $\mathrm{EW}(G) = \mathrm{EW}(G')$. The gadgets only use the labels 1/0/0, 0/1/0,

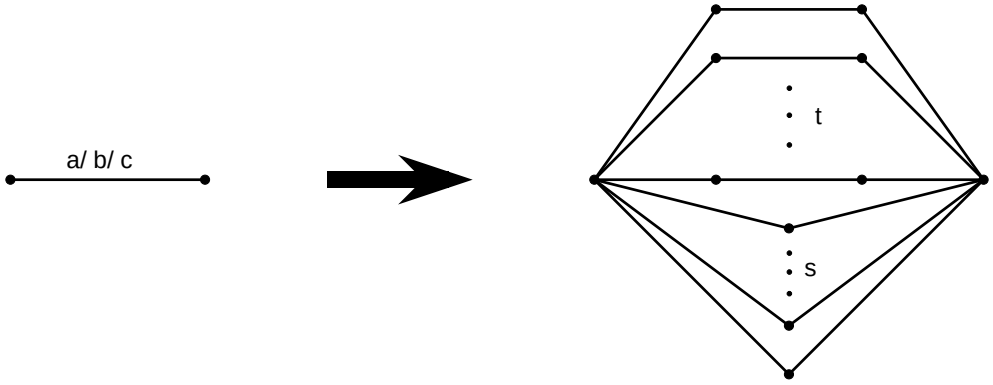


FIG. 5.7. *The transformation of Reduction* 6c.

$1/0/-1$, and $1/0/1$ in addition to the labels already present in G, so G' uses 5 labels. Finally, notice that the transformation can be performed in polynomial time.

Now we show that we can reduce the number of labels one at a time until there are none, showing that planar vertex cover is $\#\mathcal{P}$-complete.

6c. $\#k\lambda$-PLANAR EDGE-WEIGHTED SUM $\propto \#(k-1)\lambda$-PLANAR EDGE-WEIGHTED SUM. Let $G = (V, E)$ be any planar graph. Pick any label $\lambda = (a, b, c)$ used in G, let L be the set of edges with label λ and let $\ell = |L|$. Let $G_{s,t}$ be the graph formed by replacing all edges $e = (u, v)$ with label λ in the following fashion: Remove e, add vertices

$$V_e = \{w_1^e, \ldots, w_s^e, u_1^e, \ldots, u_t^e, v_1^e, \ldots, v_t^e\},$$

and add (normal) edges (u, w_i^e), (v, w_i^e), (u, u_j^e), (u_j^e, v_j^e), (v_j^e, v) for each $0 \le i \le s, 0 \le j \le t$. See Figure 5.7.

Let S be a subset of V. For each edge $e \in L$ doubly covered by S, there are $2^s 3^t$ (weighted) ways to extend S using vertices of V_e. For each edge $e \in L$ singly covered, there are 2^t ways. For each $e \in L$ not covered, there is only 1 way. Thus, if C_{ij} is the collection of subsets of V which doubly cover i edges of L and singly cover j edges of L, and

$$A_{ij} = \frac{1}{a^i b^j c^{\ell-i-j}} \sum_{S \in C_{ij}} w_G(S),$$

then

$$\mathrm{EW}(G_{s,t}) = \sum_{i,j} A_{ij} (2^s 3^t)^i (2^t)^j.$$

By Fact 5.2, if we can compute $\mathrm{EW}(G_{s,t})$ for $t = 0, \ldots, \ell$, $s = 0, \ldots, \ell$, we can recover the coefficients A_{ij} and thereby compute

$$\mathrm{EW}(G) = \sum_{i,j} A_{ij} a^i b^j c^{\ell-i-j}.$$

Notice that $\#0\lambda$-PLANAR EDGE-WEIGHTED SUM is exactly PLANAR VERTEX COVERS.

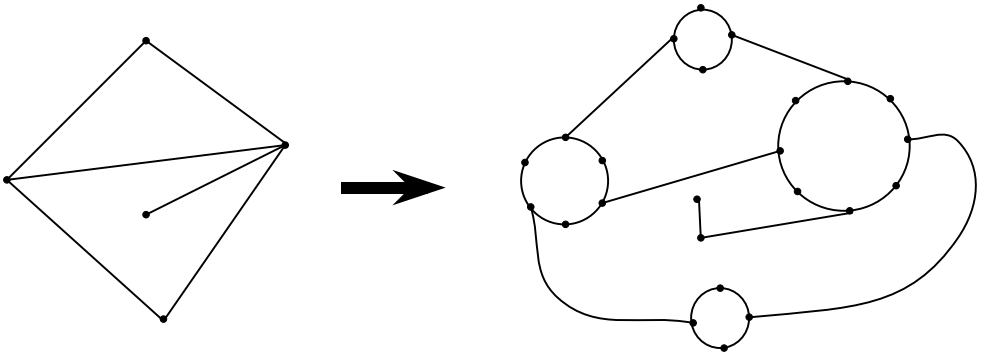


FIG. 5.8. *The transformation of Reduction* 7.

6d. #PLANAR VERTEX COVERS $\propto$ #PLANAR BIPARTITE VERTEX COVERS. Observe that the reduction of Provan and Ball [38] from #VERTEX COVERS to #BIPARTITE VERTEX COVERS preserves planarity.

7. #PLANAR BIPARTITE VERTEX COVERS $\propto$ #3Δ-PLANAR BIPARTITE MINIMUM CARDINALITY VERTEX COVERS. Let G be a bipartite planar graph. Consider the graph G' formed by taking each vertex v in G, replacing it with a cycle C_v of $2d(v)$ vertices, where $d(v)$ is the degree of v in G, and connecting the neighbors of v to alternate vertices on C_v. In order for this to preserve planarity, the neighbors must be connected with the same orientation as they have in a planar embedding of G. Let M_v be the vertices in C_v which are connected to vertices outside C_v, so $|C_v| = 2|M_v|$. See Figure 5.8 for an example.

Notice that every vertex cover in G' must be of size at least $s = \sum_{v \in V} d(v)$ to cover each cycle. Further notice that the vertex covers in G are in bijective correspondence with the covers of size s in G, under the map

$$S \mapsto \bigcup_{v \in S} M_v \cup \bigcup_{v \notin S} (C_v \setminus M_v).$$

Finally, observe that G' is bipartite (since G is), planar, and of degree at most 3.

8a. #k-REGULAR BIPARTITE PERFECT MATCHINGS $\propto$ #$(2k-2)$-REGULAR MINIMUM CARDINALITY VERTEX COVERS, any fixed $k \geq 3$. This follows immediately from Lemma 3.3, noting that the line-graph of a k-regular graph is a $(2k-2)$-regular graph.

8b. #$(k-1)$-REGULAR MINIMUM CARDINALITY VERTEX COVERS $\propto$ #k-REGULAR MINIMUM CARDINALITY VERTEX COVERS, any odd $k \geq 5$. Let H be the complete graph on $k+1$ vertices with one edge removed. Consider the sequence of graphs $\{H_n^k : n \geq 0\}$ defined as follows: H_0^k is the complete graph on $k+2$ vertices, labelled $u_1, v_1, \ldots, u_{(k-1)/2}, v_{(k-1)/2}, u, v, w$, with the edges $(u_1, v_1), \ldots, (u_{(k-1)/2}, v_{(k-1)/2}), (u, v), (v, w)$ removed. H_{n+1}^k is formed by taking the disjoint union of H_n^k and a new copy of H and and connecting one of the vertices of degree $k-1$ in H to the unique vertex in H_n^k of degree $k-1$. See Figure 5.9.

Let G be any $(k-1)$-regular graph on n nodes $u_1, u_2, \ldots, u_n$. For any $s \geq 0$, consider the disjoint union of G with n copies of H_{2s}^k. Let the vertex of degree $k-1$ in the ith copy of H_{2s}^k be labelled v_i. We can form a k-regular graph G_s by connecting u_i to v_i for each i. Lemma 6.5 tells us that there are minimum cardinality vertex covers (mcvc's) in the ith copy of H_{2s}^k both containing v_i and not containing v_i. Hence, any

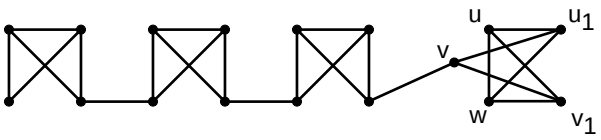


FIG. 5.9. *The graph H_3^3 in Reduction 8b.*

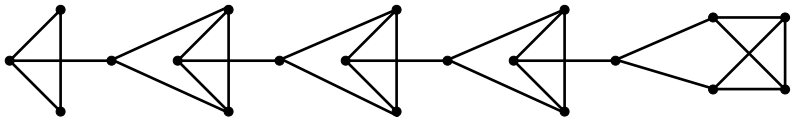


FIG. 5.10. *The graph I_4^3 in Reduction 10b.*

mcvc in G_s must be formed by taking an mcvc in G and taking mcvc's in each copy of H_{2s}^k. If G has N mcvc's and they are of size c, then G_s has $N(x_s+y_s)^c(x_s)^{n-c}$ mcvc's, where x_s is the number of mcvc's in H_{2s}^k containing the vertex of degree $k-1$ and y_s is the number not containing the vertex of degree $k-1$. With a single oracle call, we obtain the evaluation of the polynomial $f(x) = Nx^c$ at the point $z_s = 1+y_s/x_s$, which equals $1 + ((k+1)s+4)/(k-1)$ by Lemma 6.5. Notice that $f(z_0)/f(z_1) = (z_0/z_1)^c$, so with just two oracle calls we can recover c and then N. (In fact, Reduction 8a produces instances G in which we know c, so actually only one oracle call is necessary here.)

9. #3Δ-PLANAR BIPARTITE MINIMUM CARDINALITY VERTEX COVERS $\propto$ #4Δ-PLANAR BIPARTITE VERTEX COVERS. This is a standard application of the Fibonacci technique, almost identical to Reduction 1: Form G_s by attaching chains of length s to each vertex of the input graph G for $s = 0, \ldots, n$. The number of vertex covers in G_s is essentially the evaluation of a polynomial whose coefficients are the number of vertex covers in G of each size. By Lemma 6.3, these evaluation points are consecutive ratios of Fibonacci numbers, which do not repeat, so by interpolation we can recover the number of minimum cardinality vertex covers in G.

10a. #4-REGULAR MINIMUM CARDINALITY VERTEX COVERS $\propto$ #5-REGULAR VERTEX COVERS. This is another application of the Fibonacci technique. Let G be any 4-regular graph on n nodes. As in Reduction 8b, for $0 \le s \le n$, form a 5-regular graph G_s by attaching n disjoint copies of the gadget H_s^5 defined in that reduction. We recover the number of minimum cardinality vertex covers in G by polynomial interpolation, using Lemma 6.6 to guarantee that the evaluation points are distinct.

10b. #$(k-2)$-REGULAR MINIMUM CARDINALITY VERTEX COVERS $\propto$ #k-REGULAR VERTEX COVERS. This yet another application of the Fibonacci technique, with slightly different gadgets. Let H be the complete graph on $k+1$ vertices with two edges incident to some vertex v removed. Consider the sequence of graphs $\{I_n^k : n \ge 0\}$ defined as follows: I_0^k is the complete graph on $k+1$ vertices with a single edge removed. I_{n+1}^k is formed by taking the disjoint union of I_n^k and a new copy of H and connecting the vertex of degree $k-2$ in H to the two vertices of degree $k-1$ in I_n^k. See Figure 5.10. Let $(I_n^k)^+$ be the graph formed by adding to I_n^k a new vertex p and connecting p to the two vertices of degree $k-1$ in I_n^k.

Let G be any $(k-2)$-regular graph on n nodes $u_1, u_2, \ldots, u_n$. For any $0 \le s \le n$, consider the disjoint union of G with n copies of I_s^k. Let the vertices of degree $k-1$ in the ith copy of I_s^k be labelled v_i and w_i. We can form a k-regular graph

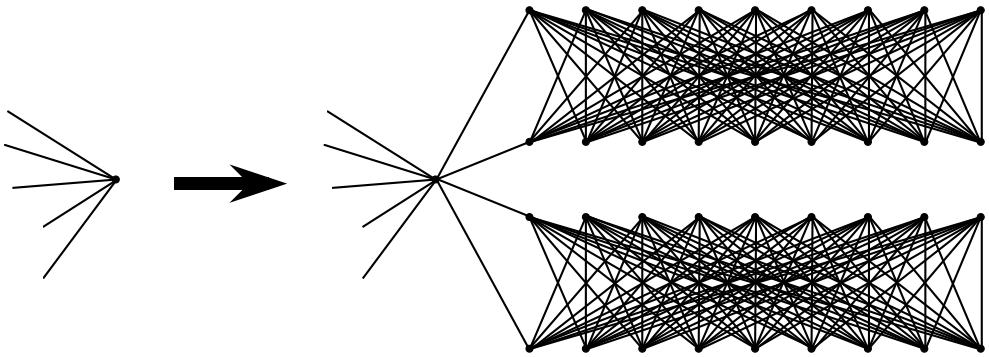


FIG. 5.11. *The transformation in Reduction* 12 *for* $r = 2$.

G_s by connecting u_i to both v_i and w_i for each i. By Lemma 6.7 and polynomial interpolation, the number of minimum cardinality vertex covers in G can be recovered from the number of vertex covers in G_s for $s = 0, \ldots, n$.

11. #4Δ-PLANAR BIPARTITE VERTEX COVERS $\propto$ #5Δ-PLANAR BIPARTITE MINIMAL VERTEX COVERS. This reduction is identical to Reduction 4.

12. #5-REGULAR VERTEX COVERS $\propto$ #REGULAR MINIMAL VERTEX COVERS. Let G be any 5-regular graph on n nodes $u_1, u_2, \ldots, u_n$. For $r \geq 0$, let J_r be the complete bipartite graph on $(5 + 2r) + (5 + 2r)$ vertices with one edge removed. Consider the disjoint union of G with nr copies of J_r, where these copies are named $H_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq r$. Let G' be the graph formed by attaching each u_i to the two vertices of degree of $4 + 2r$ in each of $H_{i,1}, H_{i,2}, \ldots, H_{i,r}$. See Figure 5.11. Notice that G' is a $(5 + 2r)$-regular graph. Notice that if G has A_i vertex covers of size i, G' has $\sum_{i=0}^{n} A_i (3^r)^i (2^r)^{n-i}$ minimal vertex covers. Dividing by 2^{rn}, we get the evaluation of $f(x) = \sum_{i=0}^{n} A_i x^i$ at $(3/2)^r$. If we choose $r = n$, we can, by Lemma 5.3, recover the coefficients of f in a single oracle call. The number of vertex covers in G is simply the sum of the coefficients. □

Proof of Corollary 4.2. These follow immediately from Theorem 4.1, Proposition 3.1, and Proposition 3.2. □

Proof of Proposition 4.3. Here we prove only that the given problems are hard to approximate within a factor $2^{n^{1/2-\epsilon}}$. Proving inapproximability within $2^{n^{1-\epsilon}}$ is more involved, and details can be found in [45] or [47].

By Proposition 3.1, we may focus on vertex covers, and the other results follow. We reduce from $2^{n^{1-\epsilon}}$-APPROX #VERTEX COVERS, which was shown to be $\mathcal{NP}$-hard by Sinclair [41] (see also Roth [40]). Note that, ignoring the planarity and bipartiteness conditions, Reduction 7 in the proof of Theorem 4.1 is a parsimonious reduction from #VERTEX COVERS to #3Δ-MINIMUM CARDINALITY VERTEX COVERS. That is, it transforms graphs G to graphs G' such that the number of minimum cardinality vertex covers in G' equals the number of vertex covers in G. Note that if G has n vertices, then the number of vertices in G' is $n' < 2n^2$, so an approximation within $2^{(n')^{1/2-\epsilon}}$ for G' gives an approximation within $2^{n^{1-\epsilon}}$ for G (for sufficiently large n). □

6. Proving that sequences do not repeat. In this section, we develop general tools for proving that sequences defined by 2×2 linear recurrences do not repeat, and apply them to deduce that the interpolation points in our reductions are distinct.

LEMMA 6.1. *Let a, b, c, d be rational numbers and let α and β be nonzero complex numbers. Let the sequence z_n be defined by*

$$z_n = \frac{a\alpha^n + b\beta^n}{c\alpha^n + d\beta^n}.$$

Then the sequence $\{z_n\}$ repeats iff $ad - bc = 0$ or α/β is a root of unity.

Proof. Cross-multiplying, we see that $z_n = z_m$ iff $(ad - bc)(\alpha^m\beta^n - \beta^m\alpha^n) = 0$ iff $ad - bc = 0$ or $(\alpha/\beta)^{n-m} = 1$. □

LEMMA 6.2. *Let A, B, C, D, x_0, and y_0 be rational numbers. Define the sequences $\{x_n\}$ and $\{y_n\}$ recursively by $x_{n+1} = Ax_n + By_n$ and $y_{n+1} = Cx_n + Dy_n$. Then the sequence $\{z_n = x_n/y_n\}$ never repeats as long as all of the following conditions hold:*

$$AD - BC \neq 0, \tag{1}$$

$$D^2 - 2AD + A^2 + 4BC \neq 0, \tag{2}$$

$$D + A \neq 0, \tag{3}$$

$$D^2 + AD + A^2 + BC \neq 0, \tag{4}$$

$$D^2 + A^2 + 2BC \neq 0, \tag{5}$$

$$D^2 - AD + A^2 + 3BC \neq 0, \tag{6}$$

$$By_0^2 - Cx_0^2 - (A - D)x_0y_0 \neq 0. \tag{7}$$

Proof. Let α and β be the eigenvalues of the matrix $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$. By basic linear algebra, as long as α and β are distinct, the general solution to the 2×2 system of linear recurrences describing x_n and y_n is given by $x_n = a\alpha^n + b\beta^n$ and $y_n = c\alpha^n + b\beta^n$, for some $a, b, c, d \in \mathbb{C}$. By the previous lemma, as long as $\alpha \neq \beta$ and neither α nor β is zero, $\{z_n\}$ can repeat only if $ad - bc = 0$ or α/β is a root of unity.

If α/β is a root of unity, it must be one of degree 1 or 2 over $\mathbb{Q}$, as $\alpha/\beta \in \mathbb{Q}(\alpha, \beta) = \mathbb{Q}(\alpha)$, which is a field extension of degree ≤ 2 over $\mathbb{Q}$. The degree of a primitive nth root of unity over $\mathbb{Q}$ is $\phi(n)$ (see, e.g., [18, section 13.2, Theorem 1]), where ϕ is Euler's totient function. By the formula $\phi(\prod p_i^{\alpha_i}) = \prod p_i^{\alpha_i - 1}(p_i - 1)$ for distinct primes p_i, one sees that only n for which $\phi(n) \leq 2$ are $1, 2, 3, 4$, and 6. The irreducible polynomials over $\mathbb{Q}$ for the corresponding primitive roots of unity are $x - 1$, $x + 1$, $x^2 + x + 1$, $x^2 + 1$, and $x^2 - x + 1$. So to check that α/β is not a root of unity, we need only check that α/β does not satisfy any of these polynomials. Using the quadratic formula, we can express α and β in terms of A, B, C, and D. The first 6 conditions in the lemma come from substituting these expressions into the polynomials that test whether (1) α or β is zero, (2) $\alpha = \beta$, and (3)–(6) α/β is a 2nd, 3rd, 4th, or 6th root of unity.

As long as $\alpha \neq \beta$ and neither are zero, we can solve for a, b, c, and d in terms of A, B, C, D, and the initial conditions x_0, y_0. Condition (7) amounts to testing whether $ad - bc = 0$ (given that $D^2 - 2AD + A^2 + 4BC \neq 0$, which is tested by condition (2). □

As observed by Greenhill [14], if A, B, C, and D are all positive (or if all are nonnegative and $A \neq D$), then only conditions (1) and (7) must be checked in the above lemma. We now apply this lemma to the various sequences that arise in our reductions.

LEMMA 6.3. *Let F_n denote the nth Fibonacci number. That is, $F_0 = 1$, $F_1 = 1$, and $F_{n+2} = F_{n+1} + F_n$ for all $n \geq 0$. The number of matchings (resp., vertex covers) in a chain of n vertices is F_n (resp., F_{n+1}). Moreover, $F_{n+1}/F_n \neq F_{m+1}/F_m$ for any $n \neq m$.*

Proof. Let x_n be the number of matchings in a chain of n vertices. Given a chain $C_n = v_1 - v_2 - \cdots - v_n$ with $n \geq 1$, the number of matchings in C_n in which v_1 is matched is x_{n-2} and the number in which v_1 is unmatched is x_{n-1}. Thus, $x_n = x_{n-1} + x_{n-2}$, which is the Fibonacci recurrence. Also note that $x_0 = 1$ and $x_1 = 1$. To obtain the result for vertex covers, observe that C_n is the line-graph of C_{n+1} and apply Lemma 3.3. The "moreover" part of the lemma follows from Lemma 6.1 with $A = B = C = x_0 = y_0 = 1$ and $D = 0$. □

LEMMA 6.4. *For $k \geq 4$, let L_s^k be the graph defined in Reduction 3 of the proof of Theorem 4.1. Let v and w be the vertices in L_s^k of degree $k-1$. Let x_s be the number of matchings in L_s^k in which both v and w are matched, let y_s be the number in which v is matched but w is not, and let z_s be the number in which neither v nor w is matched. Then x_s, y_s, and z_s can be computed in time polynomial in s and the sequence $\{w_s = (y_s + z_s)/(x_s + 2y_s + z_s)\}$ never repeats.*

Proof. For all m, let M_m denote the number of matchings in the complete graph on m vertices. Observe that $M_{m+1} = M_m + mM_{m-1}$. By inspection, we can verify that the sequences x_s and y_s satisfy the following recurrences:

$$\begin{aligned} x_{s+1} &= (k-1)M_{k-1}x_s + (k-1)M_{k-1}y_s + (k-1)M_{k-2}y_s \\ &= (k-1)M_{k-1}x_s + (M_k + (k-2)M_{k-1})y_s, \\ y_{s+1} &= M_k x_s + (M_k + M_{k-1})y_s. \end{aligned}$$

It is easy to see that y_s and z_s satisfy the same recurrence relations:

$$\begin{aligned} y_{s+1} &= (k-1)M_{k-1}y_s + (M_k + (k-2)M_{k-1})z_s, \\ z_{s+1} &= M_k y_s + (M_k + M_{k-1})z_s. \end{aligned}$$

The initial conditions are

$$\begin{aligned} z_0 &= M_{k-1}, \\ y_0 &= M_k - z_0 = M_k - M_{k-1}, \\ x_0 &= M_{k+1} - 2y_0 - 2x_0 = kM_{k-1} - M_k. \end{aligned}$$

We can compute x_s, y_s, and z_s in polynomial time using the above recurrences. Because we have three sequences here, we cannot apply Lemma 6.2 directly. However, the proof here is nearly identical to the one of Lemma 6.2, so we do not include the details that are worked out there. Since the two pairs of sequences satisfy the same recurrence relations, closed forms for these sequences will be of the form $x_s = a\alpha^s + b\beta^s, y_s = c\alpha^s + d\beta^s, z_s = e\alpha^s + f\beta^s$. So

$$w_s = \frac{(c+e)\alpha^s + (d+f)\beta^s}{(a+2c+e)\alpha^s + (b+2d+f)\beta^s}.$$

This sequence will not repeat as long as α/β is not a root of unity and $(c+e)(b+2d+f) - (d+f)(a+2c+e) \neq 0$. The conditions for α/β to not be a root of unity are the same as conditions (1)–(6) of Lemma 6.2; of these, conditions (2)–(6) are automatically satisfied since the recurrence coefficients are all positive. After simplification, condition (1) becomes

$$-M_k^2 - M_k M_{k-1} + (k-1)M_{k-1}^2 \neq 0.$$

Dividing by M_{k-1}^2 and applying the quadratic formula, we can reformulate this condition as

$$\frac{M_k}{M_{k-1}} \neq \frac{1 \pm \sqrt{4k-3}}{2}.$$

Moser and Wyman [30] have shown the following[2]:

$$\frac{1+\sqrt{4k-3}}{2} \leq \frac{M_k}{M_{k-1}} \leq \frac{1+\sqrt{4k+1}}{2}.$$

The proof of this fact is by straightforward induction, applying the recurrence $M_{k+1} = M_k + kM_{k-1}$. The same proof actually shows that *strict* inequality holds (on both sides) for all $k \geq 4$, as long as we use $k = 4$ as our base case. Thus condition (1) is also satisfied.

The only condition left to check is $(c+e)(b+2d+f) - (d+f)(a+2c+e) \neq 0$. Using the initial conditions to solve for these values, this reduces to

$$-2M_k^3 + (6-2k)M_{k-1}M_k^2 + (4k-6)M_{k-1}^2 M_k + (2k^2-6k+4)M_{k-1}^2 \neq 0.$$

Dividing by M_{k-1}^3, we obtain a cubic polynomial in M_k/M_{k-1} which vanishes iff

$$\frac{M_k}{M_{k-1}} \in \left\{ -k+2, \frac{1 \pm \sqrt{4k-3}}{2} \right\}.$$

The strengthened Moser–Wyman result shows that this cannot hold for any $k \geq 4$. □

LEMMA 6.5. *Fix k to be an odd integer ≥ 3. Let H_n^k be the graph defined in Reduction* 8b *of the proof of Theorem* 4.1. *Let v be the unique vertex in H_n^k of degree $k-1$. Then the number of minimum cardinality vertex covers in H_{2m}^k containing v is $(k-1)^{m+1}/2$ and the number not containing v is $(k-1)^m((k+1)m+4)/2$.*

Proof. First, let H be the complete graph on $k+1$ vertices with one edge removed. We now prove the lemma by induction on m.

$m = 0$: It is easily verified that the size of the mcvc in H_0^k is k, that there are $(k-1)/2$ such covers containing v, and that there are 2 such covers not containing v.

Induction step: Let v' be the vertex of degree $k-1$ in $H_{2(m+1)}^k$ and let v be the vertex of degree $k-1$ in H_{2m}^k. Now observe that the smallest vertex cover in H is of size $k-1$ and the only such cover omits both the vertices of degree $k-1$. The two copies of H added to H_{2m}^k to form $H_{2(m+1)}^k$ cannot both simultaneously be covered by covers of size $k-1$, for this would leave the edge between them uncovered. Hence, the smallest possible cover for $H_{2(m+1)}^k$ could only come from taking an mcvc on H_{2m}^k, a cover of size $k-1$ on one of the added copies of H and a cover of size k on the other copy of H. It is now easy to treat the problem in cases: For each mcvc of H_{2m}^k that contains v, there are $k-1$ mcvc's in $H_{2(m+1)}^k$ containing v' and $k+1$ mcvc's not containing v'. For each mcvc of H_{2m}^k not containing v, there are 0 mcvc's containing v' and $k-1$ mcvc's not containing v'. By induction hypothesis, this gives a total of $(k-1)((k-1)^{m+1}/2) = (k-1)^{m+2}/2$ mcvc's containing v' and $(k+1)((k-1)^{m+1}/2)+(k-1)[(k-1)^m((k+1)m+4)/2] = (k-1)^{m+1}((k+1)(m+1)+4)/2$ not containing v'. □

[2]Moser and Wyman discuss the number of solutions to $x^2 = 1$ in the symmetric group on k elements. It is easy to see that this quantity is exactly M_k.

LEMMA 6.6. *Fix k to be an odd integer ≥ 3. Let H_n^k be the graph defined in Reduction* 8b *of the proof of Theorem* 4.1. *Let v be the unique vertex in H_n^k of degree $k-1$. Define x_s to be the number of vertex covers in H_s^k containing v and y_s to be the number not containing v. Then x_s and y_s can be computed in time polynomial in s and the sequence $\{x_s/y_s\}$ never repeats.*

Proof. The sequences x_s and y_s satisfy the following recurrences:

$$x_{s+1} = (k+1)x_s + ky_s,$$
$$y_{s+1} = 2x_s + y_s,$$

with initial conditions $x_0 = (3k+3)/2$ and $y_0 = 3$. Conditions (1) and (7) of Lemma 6.2 are

$$-k+1 \neq 0,$$
$$(9k-9)/2 \neq 0.$$

It is clear that these hold for all odd integers $k \geq 3$. □

LEMMA 6.7. *Fix k to be an integer ≥ 3. Let $(I_n^k)^+$ be the graph defined in Reduction* 10b *of the proof of Theorem* 4.1. *Let p be the vertex in $(I_n^k)^+$ of degree* 2. *Define x_s to be the number of vertex covers in I_s^k containing v and let y_s be the number not containing v. Then x_s and y_s can be computed in time polynomial in s and the sequence $\{x_s/y_s\}$ never repeats.*

Proof. The sequences x_s and y_s satisfy the following recurrences:

$$x_{s+1} = (k+1)x_s + 3y_s,$$
$$y_{s+1} = (k-1)x_s + y_s,$$

with initial conditions $x_0 = k+3$ and $y_0 = k$. Conditions (1) and (7) of Lemma 6.2 are

$$-2k+4 \neq 0,$$
$$k^2 - 3k + 9 \neq 0.$$

It is easily verified that these hold for all integers $k \geq 3$. □

7. Conclusion. The study of counting and its computational complexity is both interesting and important. However, we have only a limited understanding of how the complexity of counting problems behaves in restricted cases. The results of this paper have improved the situation somewhat, but there are still many open problems. We believe that the tools developed here are likely to prove useful in obtaining restricted-case results for other counting problems.

Even regarding just the problems studied here, several unanswered questions stand out. For one, we have shown that a number of problems are hard in bounded-degree bipartite graphs and constant-degree regular graphs, but we do not know what happens if these conditions are imposed simultaneously. Do these problems remain hard in bipartite k-regular graphs, or even just bipartite regular graphs? In addition, we know that all the problems become tractable in degree 2, but some of our results only show hardness for degree 4 or higher. Recall that, subsequent to this work, Greenhill [14] has closed this degree gap for counting independent sets, but other gaps still remain.

For approximate counting, the gaps between positive and negative results are even larger. For instance, Luby and Vigoda [28] have given a polynomial-time algorithm for

approximately counting independent sets in graphs of degree 4, but the problem is only known to become $\mathcal{NP}$-hard at degree 25, as shown by Dyer, Frieze, and Jerrum [9]. An even larger gap in our knowledge is the long-standing open problem of approximately counting perfect matchings in general graphs. (Recall that this can be solved in polynomial time for dense graphs [21].) In the context of optimization problems, a substantial body of work has yielded numerous tight inapproximability results based on the PCP Theorem (cf. [7]). There is a need to develop analogous general techniques for the inapproximability of counting problems, perhaps by designing PCP systems that are tailored for this purpose.

Note added in proof. The open problem mentioned above, about approximately counting perfect matchings in general graphs, has been resolved in the positive by Jerrum, Sinclair, and Vigoda (to appear in STOC 2001).

Acknowledgments. I am indebted to a few people for helping me complete this work: my undergraduate advisor, Leslie Valiant, who sparked my interest in complexity theory and provided advice and inspiration throughout; Dan Roth, who suggested these problems and guided me through the initial stages of research; and Jennifer Sun, who spent many hours reading and critiquing the original version of this work.

I also thank Russ Bubley, Sergey Fomin, Michel Goemans, Catherine Greenhill, Vijay Vazirani, and the anonymous referees for numerous helpful suggestions and pointers.

REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[2] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[3] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—Towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[4] A. Z. Broder, *How hard is to marry at random? (On the approximation of the permanent)*, in Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 50–58. Erratum in Proceedings of 20th STOC.

[5] R. Bubley and M. Dyer, *Graph orientations with no sink and an approximation for a hard case of #SAT*, in Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 1997, SIAM, Philadelphia, 1997, pp. 248–257.

[6] R. Bubley and M. Dyer, *Path coupling: A technique for proving rapid mixing in Markov chains*, in 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, IEEE, Piscataway, NJ, pp. 223–231.

[7] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html (2000).

[8] P. Dagum and M. Luby, *Approximating the permanent of graphs with large factors*, Theoret. Comput. Sci., 102 (1992), pp. 283–305.

[9] M. Dyer, A. Frieze, and M. Jerrum, *On counting independent sets in sparse graphs*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1999, pp. 210–217.

[10] M. Dyer and C. Greenhill, *On Markov chains for independent sets*, J. Algorithms, 35 (2000), pp. 17–49.

[11] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.

[12] M. E. Fisher, *Statistical mechanics of dimers on a plane lattice*, Physical Rev., Ser. 2, 124 (1961), pp. 1664–1672.

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

[14] C. Greenhill, *The complexity of counting colourings and independent sets in sparse graphs and hypergraphs*, Computational Complexity, 9 (2000), pp. 52–73.
[15] F. Harary, ed., *Graph Theory and Theoretical Physics*, Academic Press, London, New York, 1967.
[16] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 1–10.
[17] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns, *The complexity of planar counting problems*, SIAM J. Comput., 27 (1998), pp. 1142–1167.
[18] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Graduate Texts in Math. 84, 2nd ed., Springer-Verlag, New York, 1990.
[19] M. Jerrum, *Two-dimensional monomer-dimer systems are computationally intractable*, J. Statist. Phys., 48 (1987), pp. 121–134. Erratum, 59 (1990), pp. 1087–1088.
[20] M. Jerrum, *The computational complexity of counting*, in Proceedings of the International Congress of Mathematicians, Zürich, 1994, Birkhäuser, Basel, 1995, pp. 1407–1416.
[21] M. Jerrum and A. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.
[22] M. Jerrum and A. Sinclair, *Polynomial-time approximation algorithms for the Ising model*, SIAM J. Comput., 22 (1993), pp. 1087–1116.
[23] M. Jerrum and A. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, Boston, 1997, pp. 482–520.
[24] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.
[25] R. Kannan, *Markov chains and polynomial time algorithms*, in Annual Symposium on the Foundations of Computer Science, IEEE, Piscataway, NJ, 1994, pp. 656–669.
[26] R. M. Karp, M. Luby, and N. Madras, *Monte-Carlo approximation algorithms for enumeration problems*, J. Algorithms, 10 (1989), pp. 429–448.
[27] P. Kasteleyn, *Dimer statistics and phase transitions*, J. Math. Phys., 4 (1963), pp. 287–293.
[28] M. Luby and E. Vigoda, *Approximately counting up to four (extended abstract)*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 682–687.
[29] M. Luby and E. Vigoda, *Fast convergence of the Glauber dynamics for sampling independent sets*, Random Structures Algorithms, 15 (1999), pp. 229–241.
[30] L. Moser and M. Wyman, *On solutions of $x^d = 1$ in symmetric groups*, Canadian J. Math., 7 (1955), pp. 159–168.
[31] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995, chap. 11: Approximate counting, pp. 306–334.
[32] M. Ogihara, T. Thierauf, S. Toda, and O. Watanabe, *On the closure properties of $\#P$ in the context of $PF \circ \#P*$*, J. Comput. Systems Sci., 53 (1996), pp. 171–179.
[33] P. Orponen, *Dempster's rule of combination is $\#P$-complete*, Artificial Intelligence, 44 (1990), pp. 245–253.
[34] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[35] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[36] G. M. Provan, *A logical-based analysis of Dempster-Shafer theory*, Int. J. Approx. Reasoning, 4 (1990), pp. 451–498.
[37] J. S. Provan, *The complexity of reliability computations in planar and acyclic graphs*, SIAM J. Comput., 15 (1986), pp. 694–702.
[38] J. S. Provan and M. O. Ball, *The complexity of counting cuts and of computing the probability that a graph is connected*, SIAM J. Comput., 12 (1983), pp. 777–788.
[39] D. Randall and P. Tetali, *Analyzing Glauber dynamics by comparison of Markov chains*, in Proceedings of LATIN '98: Third Latin American Symposium on Theoretical Informatics, C. Lucchesi and A. Moura, eds., Lecture Notes in Comput. Sci. 1380, Springer-Verlag, Berlin, 1998, pp. 292–304.
[40] D. Roth, *On the hardness of approximate reasoning*, Artificial Intelligence, 82 (1996), pp. 273–302.
[41] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*, Prog. Theoret. Comput. Sci., Birkhäuser, Boston, 1993.
[42] L. Stockmeyer, *On approximation algorithms for $\#P$*, SIAM J. Comput., 14 (1985), pp. 849–861.
[43] H. N. V. Temperley and M. E. Fisher, *Dimer problem in statistical mechanics—An exact result*, Philosophical Magazine, Ser. 8, 6 (1961), pp. 1061–1063.

[44] A. Tucker, *Applied Combinatorics*, 3rd ed., John Wiley & Sons, New York, 1995.
[45] S. P. Vadhan, *The Complexity of Counting*, Undergraduate thesis, Harvard University, http://deas.harvard.edu/~salil (1995).
[46] S. P. Vadhan, *Rapidly mixing Markov chains and their applications*, Essay, Churchill College, Cambridge University, http://deas.harvard.edu/~salil (May 1996).
[47] S. P. Vadhan, *The complexity of counting in sparse, regular, and planar graphs*, Preliminary version, http://deas.harvard.edu/~salil (May 1997).
[48] L. G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[49] L. G. Valiant, *The complexity of enumeration and reliability problems*, SIAM J. Comput., 8 (1979), pp. 410–421.
[50] D. J. A. Welsh, *Complexity: Knots, Colourings, and Counting*, London Math. Soc. Lecture Note Ser. 188, Cambridge University Press, Cambridge, UK, 1993.
[51] D. Zuckerman, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 428–446

MATCHINGS MEETING QUOTAS AND THEIR IMPACT ON THE BLOW-UP LEMMA*

VOJTECH RÖDL†, ANDRZEJ RUCIŃSKI‡, AND MICHELLE WAGNER§

Abstract. A bipartite graph $G = (U, V; E)$ is called ϵ-regular if the edge density of every sufficiently large induced subgraph differs from the edge density of G by no more than ϵ. If, in addition, the degree of each vertex in G is between $(d-\epsilon)n$ and $(d+\epsilon)n$, where d is the edge density of G and $|U| = |V| = n$, then G is called super (d, ϵ)-regular. In [*Combinatorica*, 19 (1999), pp. 437–452] it was shown that if $S \subset U$ and $T \subset V$ are subsets of vertices in a super-regular bipartite graph $G = (U, V; E)$, and if a perfect matching M of G is chosen randomly, then the number of edges of M that go between the sets S and T is roughly $|S||T|/n$. In this paper, we derandomize this result using the Erdős–Selfridge method of conditional probabilities. As an application, we give an alternative constructive proof of the blow-up lemma of Komlós, Sárközy, and Szemerédi (see [*Combinatorica*, 17 (1997), pp. 109–123] and [*Random Structures Algorithms*, 12 (1998), pp. 297–312]).

Key words. ϵ-regular graphs, perfect matchings, conditional probabilities, randomized algorithms, derandomization, blow-up lemma

AMS subject classifications. 05C70, 05C80, 68R10

PII. S0097539700371053

1. A politically correct cultural revolution and its impact on the blow-up lemma.

Let X be a radical politician who, in an attempt to run a politically correct presidential election campaign, makes a promise to each minority group that it will enjoy "statistically equal" status with respect to every quality of life, such as wealth, health, or education. After winning the election, he asks his advisors how to fulfill his commitments. Fortunately, one of the advisors has a background in mathematics and she proposes a sound solution: take all citizens' social security numbers and reassign them randomly, and then let all citizens assume their new lives accordingly. Thus, with high probability every group (e.g., women, elderly, African-American, gay, etc.) will each be proportionally represented in every category of quality of life (see Proposition 1 in section 2). President X does not want to take any chances, however, and asks for a "no-risk" solution. As there are more than 250,000,000 legal citizens, searching through all posssible permutations is not an option. Hence, the mathematically inclined advisor must design an algorithm which, in reasonable time, produces a reassignment that fulfills the president's promises (see Proposition 2 in section 3).

Then another obstacle arises. One of the other advisors observes that each person can realistically be reassigned to the lives of only a subcollection of other people. As a result, the advisors produce a huge chart that illustrates all such pairs of people (a

*Received by the editors April 27, 2000; accepted for publication (in revised form) March 9, 2001; published electronically August 22, 2001. Part of this research was presented at RANDOM'98 in Barcelona, Spain (see [14]).

http://www.siam.org/journals/sicomp/31-2/37105.html

†Department of Mathematics and Computer Science, Emory University, 1784 N. Decatur Road Suite 100, Atlanta, GA 30322 (rodl@mathcs.emory.edu). This author was supported by NSF grant DMS-9704114 and by an Emory University Research grant.

‡Department of Discrete Mathematics, Adam Mickiewicz University, ul. Matejki 48/49, 60-769 Poznań, Poland (andrzej@mathcs.emory.edu). This author was supported by KBN grant 2 P03A 032 16. Part of this research was done during the second author's visits to Emory University.

§University of Wisconsin-La Crosse, 1015 Cowley Hall, 1725 State Street, La Crosse, WI 54601 (wagner@math.uwlax.edu).

"who can be whom" chart). Fortunately for President X, the talented female advisor notices that the chart enjoys a great deal of regularity and that each citizen can be assigned to a significant proportion of all citizens. After several years of intense research, she proves that a random reassignment, consistent with the "who can be whom" chart, is also very likely to produce a satisfiable outcome (see Theorem 2.2 in section 2). Moreover, the procedure can be "derandomized" in reasonable time to yield the required "no-risk" solution (see Theorem 4.1 in section 4).

Unfortunately, by the time the advisor proposes her solution, President X is no longer in office. So that her research does not go to waste, she searches for another real-life application and finds it in the "blow-up lemma" (see section 5).

2. The catching lemma. A permutation $\sigma : [n] \to [n]$ can be viewed as a perfect matching of the complete bipartite graph $K_{n,n}$. *A random perfect matching* of a graph is a perfect matching drawn randomly, with uniform distribution, from the set of all perfect matchings in that graph.

Suppose that two subsets S and T of $[n]$ are given. If $\sigma : [n] \to [n]$ is a random permutation of the set $[n]$, then the number $|\sigma(S) \cap T|$ of elements of S mapped onto the elements of T has a hypergeometric distribution with expectation $|S||T|/n$. Equivalently, given a random perfect matching σ of $K_{n,n}$, and a pair of sets (S, T) with S and T on opposite sides, the number of edges $|\sigma(S) \cap T|$ "caught" in between S and T is a hypergeometric random variable with expectation $|S||T|/n$. In fact, in the following proposition we shall see that using Chernoff's bound for the hypergeometric distribution, one can obtain sharp concentration results simultaneously for the number of edges caught by many pairs (S_i, T_i), $i = 1, 2, \ldots, k$.

The following statement of Chernoff's inequality is valid for every binomial or hypergeometric random variable X on n trials (see [7, pp. 28–29]): for $t \geq 0$,

$$P(|X - E(X)| \geq t) \leq 2 \exp\left(-\frac{2t^2}{n}\right). \tag{1}$$

In Proposition 1 below, as well as in other statements, the sets S_i and T_i may be repeated; in other words, it is possible that $S_i = S_j$ for some $i \neq j$. In fact, in our "real-life" application from section 1, the sets S range through all the minority groups, and, for a fixed set S_i, the sets T_j range through all categories of qualities of life. On the other hand, we shall see in section 5 that in other applications all sets S_i (and all sets T_i) may be pairwise distinct.

PROPOSITION 1 (the complete catching lemma). *Given positive integers n, k, and $0 < \lambda < 1$, let U and V, $|U| = |V| = n$, be the two vertex classes of a complete bipartite graph $K_{n,n}$, and let $S_i \subset U$, $T_i \subset V$, $i = 1, 2, \ldots, k$. If $\sigma : U \to V$ is a random perfect matching in $K_{n,n}$, then the event that the inequality*

$$|S_i||T_i|/n - \lambda n < |\sigma(S_i) \cap T_i| < |S_i||T_i|/n + \lambda n \tag{2}$$

holds for all $i = 1, \ldots, k$ occurs with probability at least $1 - 2k \exp\{-2\lambda^2 n\}$.

Proof. Set $s_i = |S_i|$, $t_i = |T_i|$, and $a_i = \lceil s_i t_i/n - \lambda n \rceil$, $b_i = \lfloor s_i t_i/n + \lambda n \rfloor$, $i = 1, \ldots, k$. Let $X_i = |\sigma(S_i) \cap T_i|$, and let A_i denote the event that $a_i \leq X_i \leq b_i$, $i = 1, 2, \ldots, k$. For every integer x, we have

$$P(X_i = x) = \frac{\binom{t_i}{x}\binom{n-t_i}{s_i-x}}{\binom{n}{s_i}}$$

and

$$P(A_i) = \sum_{x \in [a_i, b_i]} P(X_i = x).$$

By (1), it follows that

$$P(\overline{A_i}) = \sum_{x \in [a_i, b_i]} P(X_i = x) \leq 2 \exp\{-2\lambda^2 n\}.$$

Now if X counts the number of events A_i that do not occur, then the probability that (2) fails to hold for some $i = 1, 2, \ldots, k$ is

$$P(X > 0) \leq E(X) = \sum_i P(\overline{A_i}) < 2k \exp\left\{-2\lambda^2 n\right\}. \qquad \square$$

In [13], a generalization of Proposition 1 was proved. Given a bipartite graph G with bipartition (U, V) and two subsets $U' \subset U$ and $V' \subset V$, denote by $E_G(U', V')$ the set of edges of G with one endpoint in U', and the other in V', and set $e_G(U', V') = |E_G(U', V')|$. Define the *density* $d_G(U', V')$ of the pair (U', V') in G by $d_G(U', V') = \frac{e_G(U', V')}{|U'||V'|}$. The quantity $d_G(U, V)$ is also called *the density of* G.

The graph G is called *ϵ-regular* if for every pair of sets (U', V'), $U' \subset U$, $V' \subset V$, $|U'| > \epsilon|U|$, $|V'| > \epsilon|V|$, we have

$$|d_G(U', V') - d_G(U, V)| < \epsilon.$$

A bipartite graph G whose bipartition classes each have size n is referred to as a *bipartite graph with $2n$ vertices*. Let $N_G(u)$ stand for the neighborhood of vertex u in graph G, and let $deg_G(u) = |N_G(u)|$. It is easy to check that in an ϵ-regular bipartite graph G with $2n$ vertices, all but at most $3\epsilon n^2$ pairs of vertices $u_1, u_2 \in U$ satisfy

$$(d - \epsilon)n < deg(u_1), deg(u_2) < (d + \epsilon)n$$

and

$$(d - \epsilon)^2 n < |N_G(u_1) \cap N_G(u_2)| < (d + \epsilon)^2 n.$$

In fact, this implication can be reversed at the cost of enlarging ϵ slightly (see [1]).

LEMMA 2.1. *Given a positive integer n and $\epsilon > 0$, if $G = (U, V; E)$, $|U| = |V| = n$, is a bipartite graph containing at least $(1 - 5\epsilon)n^2/2$ pairs of vertices $u_1, u_2 \in U$ that satisfy*

(i) $deg(u_1), deg(u_2) > (d - \epsilon)n$,

(ii) $|N_G(u_1) \cap N_G(u_2)| < (d + \epsilon)^2 n$,

then G is $(16\epsilon)^{1/5}$-regular.

Sometimes it is desired that *all* vertices have their degrees close to the average. Given $\epsilon > 0$ and $0 < d < 1$, an ϵ-regular bipartite graph G with $2n$ vertices is called *super (d, ϵ)-regular* if the minimum degree $\delta(G)$ and the maximum degree $\Delta(G)$ of G satisfy

$$(d - \epsilon)n \leq \delta(G) \leq \Delta(G) \leq (d + \epsilon)n.$$

Using Hall's theorem, it is straighforward to show the following.

FACT 1. *Every ϵ-regular bipartite graph with $2n$ vertices and minimum degree at least ϵn contains a perfect matching.*

Moreover, in [2] it was shown that in every large super (d,ϵ)-regular bipartite graph, the number of perfect matchings is roughly the same as the number of perfect matchings expected in the random bipartite graph with edge probability d. More precisely, if $M(G)$ denotes the number of perfect matchings in G, then, for $0<\epsilon<d/2$ and n sufficiently large,

$$(d-2\epsilon)^n n! \le M(G) \le (d+2\epsilon)^n n!.$$

If G is only ϵ-regular with density d, then the upper bound still holds. This estimate was used to prove the following result (see [13]). Unlike the original statement, the probability bound is given explicitly here.

THEOREM 2.2 (catching lemma). *Given real numbers $0<d<1$, $0<\epsilon<(d/2)^4$, $\lambda>0$, and a sufficiently large integer $n_0=n_0(\epsilon)$, the following holds. Let G be a super (d,ϵ)-regular graph with bipartition (U,V), $|U|=|V|=n>n_0$, and let $S_i\subset U$, $T_i\subset V$, $i=1,2,\dots,k$. If $\sigma: U\to V$ is a random perfect matching of G, then the event that (2) holds for each $i=1,2,\dots,k$ occurs with probability at least $1-2k\exp\{4\epsilon^{1/4}n-2\lambda^2 n\}$.* □

The main goal of this paper is to show how to derandomize this probabilistic result; that is, we show how to efficiently construct a perfect matching with the property described in Theorem 2.2. For this, we use the standard Erdős–Selfridge method of conditional probabilities ([4]; cf. [3]) but in a nonstandard setting. First, we illustrate this method by proving the following lemma.

LEMMA 2.3. *Let m, n, and r be positive integers such that*

$$(r-1)(n+1)\exp\{-m^{1/3}/2\}<1,$$

and let $N_1,\dots,N_n$ be subsets of $V=\{v_1,\dots,v_m\}$. Given $0<p_j<1$, $j=1,2,\dots,r$, where $\sum_{j=1}^r p_j=1$, there exists a polynomial (in m and n) time algorithm **PARTITION** *that constructs a partition $V=R_1\cup\cdots\cup R_r$ such that for each $j=1,2,\dots,r$, $|R_j|=p_j m$, and for each $i=1,\dots,n$, $||R_j\cap N_i|-p_j|N_i||<m^{2/3}$.*

Proof. For unification, we set $N_{n+1}=V$. First notice that if a set R was constructed randomly by m independent coin flippings, each with probability p, then by (1) for the binomial distribution, the probability that for some $i=1,2,\dots,n+1$, $|R\cap N_i|-p|N_i||>\frac{1}{2}m^{2/3}$, would be at most

$$(n+1)\exp\{-m^{4/3}/2m\}=(n+1)\exp\{-m^{1/3}/2\}<1/(r-1). \tag{3}$$

We shall derandomize the above probabilistic statement simultaneuosly for all sets N_i and for all partition sets R_j. We restrict ourselves to the case $r=2$ in the proof below for clarity of exposition. The proof for arbitrary r follows along the same lines; alternatively, the general case is implied by repeated applications of the case $r=2$ (at the cost of error accumulation of at most $rm^{2/3}$).

With $r=2$, $p=p_1$, and $R=R_1$, we sequentially construct a binary sequence $\{\xi_1,\dots,\xi_m\}$, where $\xi_l=1$ if and only if $v_l\in R$. Let R^l denote $R\cap\{v_1,\dots,v_l\}$; in particular, let $R^0=\emptyset$. Let Y be a random variable that counts how many of the $n+1$ inequalities are not satisfied. By (3), we know that $E(Y)<1$.

Assuming $\xi_1,\dots,\xi_{l-1}$ have been determined, we choose $\xi_l\in\{0,1\}$ to minimize the conditional expectation $E(Y|\xi_1,\dots,\xi_l)$, which is given by

$$\sum_{i=1}^{n+1}\sum_{x\notin[a_{i,l},b_{i,l}]}\binom{n_{i,l}}{x}p^x(1-p)^{n_{i,l}-x}, \tag{4}$$

where

$$a_{i,l} = p|N_i| - m^{2/3} - |R^l \cap N_i|, \qquad b_{i,l} = p|N_i| + m^{2/3} - |R^l \cap N_i|,$$

and

$$n_{i,l} = |N_i| - |R^l \cap N_i|.$$

It is straightforward to check that

$$E(Y|\xi_1, \ldots, \xi_{l-1}) = pE(Y|\xi_1, \ldots, \xi_{l-1}, 1) + (1-p)E(Y|\xi_1, \ldots, \xi_{l-1}, 0),$$

and thus, for each $l = 1, \ldots, m$,

$$E(Y|\xi_1, \ldots, \xi_l) \le E(Y|\xi_1, \ldots, \xi_{l-1}) < 1.$$

In particular, when $l = m$, we have

$$0 < Y(\xi_1, \ldots, \xi_m) = E(Y|\xi_1, \ldots, \xi_m) < 1,$$

which implies that the integer $Y(\xi_1, \ldots, \xi_m)$ equals zero.

Hence, we have for all $i = 1, \ldots, n+1$, $||R \cap N_i| - p|N_i|| \le \frac{1}{2}m^{2/3}$. To obtain the desired set R of order pm, the values of at most $\frac{1}{2}m^{2/3}$ elements ξ_i are interchanged (from 0 to 1 or vice versa) at the cost of doubling the errors in the approximations of $|R \cap N_i|$, $i = 1, 2, \ldots, n$. □

3. Derandomizing the complete catching lemma. In this section, we warm up by derandomizing Proposition 1, the special case of Theorem 2.2 in which $G = K_{n,n}$.

PROPOSITION 2. *Let n and k be positive integers, and let $0 < \lambda < 1$ such that $2k \exp\{-2\lambda^2 n\} < 1$. There exists a deterministic polynomial (in both n and k) time algorithm which, given a complete bipartite graph $K_{n,n}$ with vertex classes U and V, and k pairs of subsets $S_i \subset U$, $T_i \subset V$, $i = 1, 2, \ldots, k$, constructs a perfect matching of $K_{n,n}$ such that inequality (2) holds for all $i = 1, 2, \ldots, k$.*

Proof. To construct the desired perfect matching, we shall apply the Erdős–Selfridge method of conditional probabilities. However, unlike Lemma 2.3, the choices of edges in the matching are not independent, and hence the underlying probability space is not binomial. In this case, the method succeeds due to a combinatorial identity enjoyed by the hypergeometric distribution (cf. Claim 1 below).

Let $s_i = |S_i|$, $t_i = |T_i|$, $a_i = \lceil s_i t_i/n - \lambda n \rceil$, and $b_i = \lfloor s_i t_i/n + \lambda n \rfloor$, and let A_i denote the event that $a_i \le |\sigma(S_i) \cap T_i| \le b_i$, $i = 1, 2, \ldots, k$. Let X count the number of events A_i that do not hold. By applying Chernoff's inequality to the hypergeometric random variable X, we have

$$E(X) = \sum_i P(\overline{A_i}) = \sum_{i=1}^{k} \sum_{x \notin [a_i, b_i]} \frac{\binom{t_i}{x}\binom{n-t_i}{s_i-x}}{\binom{n}{s_i}} < 2k \exp\left\{-2\lambda^2 n\right\} < 1.$$

We shall sequentially select edges to form a perfect matching that satisfies (2). Having chosen one edge $e_1 = uv$, $u \in U$, $v \in V$, we restrict our subsequent search to the complete bipartite graph $K_{n-1,n-1}$ obtained by removing the two endpoints of e_1. Then we update the sets S_i and T_i by setting $S_{i,1} = S_i \setminus \{u\}$ and $T_{i,1} = T_i \setminus \{v\}$. We also update the intervals $[a_i, b_i]$ by shifting them to the left one unit if e_1 is "caught"

by the pair (S_i, T_i); that is, if $u \in S_i$ and $v \in T_i$. We denote the new interval limits by $a_{i,1}$ and $b_{i,1}$. If A_i^1 is the event (in the space of all perfect matchings of $K_{n-1,n-1}$) that $a_{i,1} \le |\sigma(S_{i,1}) \cap T_{i,1}| \le b_{i,1}$, $i = 1, 2, \ldots, k$, then

$$P(\overline{A_i}|e_1) = P(\overline{A_i^1}).$$

In general, we define the following function f on the set of all matchings of $K_{n,n}$. For each $l = 0, 1, 2, \ldots, n$, let $n_l = n - l$. Given a matching $\sigma = \{e_1, \ldots, e_l\}$ we set

$$f(\sigma) = \sum_{i=1}^{k} \sum_{x \notin [a_{i,\sigma}, b_{i,\sigma}]} \frac{\binom{t_{i,\sigma}}{x}\binom{n_l - t_{i,\sigma}}{s_{i,\sigma} - x}}{\binom{n_l}{s_{i,\sigma}}}, \tag{5}$$

where

$$s_{i,\sigma} = |S_i \setminus \{e_1 \cup \cdots \cup e_l\}|, \qquad t_{i,\sigma} = |T_i \setminus \{e_1 \cup \cdots \cup e_l\}|,$$

and where

$$a_i - a_{i,\sigma} = b_i - b_{i,\sigma} = |E(S_i, T_i) \cap \{e_1 \cup \cdots \cup e_l\}|.$$

Of course, for the empty matching $\sigma = \emptyset$, we have $f(\emptyset) = E(X)$, and thus $f(\emptyset) < 2ke^{-2\lambda^2 n} < 1$.

This function can also be defined recursively. For each $l = 1, 2, \ldots, n$, and each matching $\sigma = \{e_1, \ldots, e_{l-1}\}$, set

$$f(e_1, \ldots, e_l) = \sum_{i=1}^{k} \sum_{x \notin [a_{i,l}, b_{i,l}]} \frac{\binom{t_{i,l}}{x}\binom{n_l - t_{i,l}}{s_{i,l} - x}}{\binom{n_l}{s_{i,l}}}, \tag{6}$$

where $s_{i,l-1} = s_{i,\sigma}$, $t_{i,l-1} = t_{i,\sigma}$, $a_{i,l-1} = a_{i,\sigma}$, and $b_{i,l-1} = b_{i,\sigma}$. Then for $e_l = uv$ we have

$$s_{i,l} = \begin{cases} s_{i,l-1} - 1 & \text{if } u \in S_i, \\ s_{i,l-1} & \text{if } u \notin S_i, \end{cases} \qquad t_{i,l} = \begin{cases} t_{i,l-1} - 1 & \text{if } v \in T_i, \\ t_{i,l-1} & \text{if } v \notin T_i, \end{cases} \tag{7}$$

$$a_{i,l-1} - a_{i,l} = b_{i,l-1} - b_{i,l} = \begin{cases} 1 & \text{if } u \in S_i, v \in T_i, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $s_{i,0} = s_i$, $t_{i,0} = t_i$, $a_{i,0} = a_i$, $b_{i,0} = b_i$.

It is crucial that $f(e_1, \ldots, e_{l-1})$ is the average of $f(e_1, \ldots, e_l)$ over all $(n - l + 1)^2$ choices of e_l.

Claim 1. *Let E_0 be the set of all edges of $K_{n,n}$, and for each $l = 1, 2, \ldots, n$, let E_l be the set of all edges in the complete bipartite graph that remain after deleting the endpoints of the edges $e_1, \ldots, e_l$. Then*

$$\frac{1}{(n-l+1)^2} \sum_{e_l \in E_{l-1}} f(e_1, \ldots, e_l) = f(e_1, \ldots, e_{l-1}).$$

This claim shall be proved at the end of this section. Observe that as a consequence of Claim 1, there is always a choice of e_l for which

$$f(e_1, \ldots, e_l) \le f(e_1, \ldots, e_{l-1}). \tag{9}$$

Since $s_{i,l}$, $t_{i,l}$, $a_{i,l}$, and $b_{i,l}$ are updated by (7)–(8) after each edge of σ is chosen, for each $l = 1, 2, \ldots, n$ we have

$$E(X|e_1, \ldots, e_l) = \sum_i P(\overline{A_i}|e_1, \ldots, e_l) = f(e_1, \ldots, e_l).$$

In particular, when $l = n$ we may observe that $n_n = t_{i,n} = s_{i,n} = 0$, while $P(\overline{A_i}|e_1, \ldots, e_n) = 1$ if $0 \notin [a_{i,n}, b_{i,n}]$, and $P(\overline{A_i}|e_1, \ldots, e_n) = 0$ otherwise. Thus, for each event A_i that fails, there is a corresponding term in (6) that equals 1 (with $x = 0$). By (9),

$$E(X|e_1, \ldots, e_n) = f(e_1, \ldots, e_n) \leq E(X) < 1,$$

which means that $X(e_1, \ldots, e_n) = 0$. Hence, the perfect matching $\sigma = \{e_1, \ldots, e_n\}$ satisfies (2). □

Proof of Claim 1. For clarity, we shall prove only the claim for $l = 1$; that is, we shall prove that $\frac{1}{n^2} \sum_{e_1 \in E_0} f(e_1) = f(\emptyset) = E(X)$. The general case is similar. We begin by interchanging the first two summations so that

$$\frac{1}{n^2} \sum_{e_1 \in E_0} f(e_1) = \frac{1}{n^2} \sum_{i=1}^{k} \sum_{e_1 \in E_0} \Phi_i, \tag{10}$$

where

$$\Phi_i = \sum_{x \notin [a_{i,1}, b_{i,1}]} \frac{\binom{t_{i,1}}{x}\binom{n_1 - t_{i,1}}{s_{i,1} - x}}{\binom{n_1}{s_{i,1}}}.$$

Now observe that for each $i = 1, 2, \ldots, k$, the summand $\frac{\binom{t_{i,1}}{x}\binom{n_1 - t_{i,1}}{s_{i,1} - x}}{\binom{n_1}{s_{i,1}}}$ depends exclusively on which of the four positions, relative to the pair (S_i, T_i), the edge e_1 takes (see Figure 1).

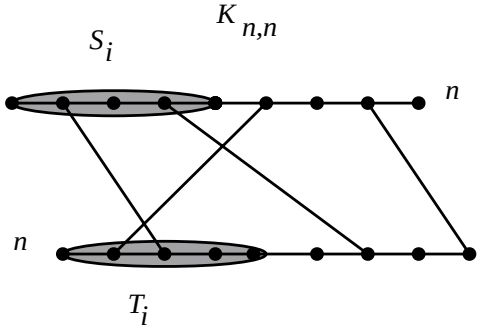


FIG. 1. *The four possible positions of* e_1.

Hence, we may write (10) as

$$\begin{aligned} \Phi_i = \frac{s_i t_i}{n^2} \sum_x p_{i,1} + \frac{s_i(n - t_i)}{n^2} \sum_x p_{i,2} \\ + \frac{(n - s_i)t_i}{n^2} \sum_x p_{i,3} + \frac{(n - s_i)(n - t_i)}{n^2} \sum_x p_{i,4}, \end{aligned} \tag{11}$$

where the four summations are taken over all $x \notin [a_{i,1}, b_{i,1}]$ and where

$$p_{i,1} = \frac{\binom{t_i-1}{x}\binom{n-t_i}{s_i-1-x}}{\binom{n-1}{s_i-1}}, \qquad p_{i,2} = \frac{\binom{t_i}{x}\binom{n-1-t_i}{s_i-1-x}}{\binom{n-1}{s_i-1}},$$

$$p_{i,3} = \frac{\binom{t_i-1}{x}\binom{n-t_i}{s_i-x}}{\binom{n-1}{s_i}}, \qquad p_{i,4} = \frac{\binom{t_i}{x}\binom{n-1-t_i}{s_i-x}}{\binom{n-1}{s_i}}.$$

Let us write $f(\emptyset) = \sum_{i=1}^{k} \Psi_i$, where

$$\Psi_i = \sum_{x \notin [a_i, b_i]} \frac{\binom{t_i}{x}\binom{n-t_i}{s_i-x}}{\binom{n}{s_i}}.$$

It can now be checked algebraically that for each $i = 1, 2, \ldots, k$, we have $\Phi_i = \Psi_i$.

Alternatively, for fixed i, imagine the following two-stage random experiment: an edge e_1 is first chosen uniformly at random; then a perfect matching is chosen randomly from the remaining subgraph $K_{n-1,n-1}$ to form a random perfect matching σ of $K_{n,n}$. Let B_j, $j = 1, 2, 3, 4$, denote the event that, relative to the pair (S_i, T_i), the edge e_1 takes a position that corresponds to the term $p_{i,j}$. Then we have $P(B_1) = \frac{s_i t_i}{n^2}$, $P(B_2) = \frac{s_i(n-t_i)}{n^2}$, $P(B_3) = \frac{(n-s_i)t_i}{n^2}$, and $P(B_4) = \frac{(n-s_i)(n-t_i)}{n^2}$. Moreover, $P(\overline{A_i}|B_j) = \sum_{x \notin [a_i, b_i]} p_{i,j}$. Hence, by the law of total probability, every term of (11) is equal to the corresponding term of the sum (6) with $l = 0$. □

Remark. One might think that it would be simpler to fix in each step an unmatched vertex u of U and choose the next edge only from among those incident with u. Unfortunately, as we shall see in section 4, this technique does not carry over to the general case of super-regular graphs since we have no control over the neighborhoods of individual vertices.

4. Derandomizing the catching lemma. Some say that super (d, ϵ)-regular graphs are like complete bipartite graphs, only sparser. In a super (d, ϵ)-regular graph, the number of edges between sets of sizes s and t is roughly dst, and the number of perfect matchings in such a graph is, very roughly, $d^n n!$. Hence, it might seem that derandomizing Theorem 2.2 involves a mere repetition of the above argument, only with the presence of a factor of d. However, we shall see that this is not so.

One difficulty we must cope with is that we can no longer enjoy the hereditary property possessed by complete bipartite graphs, namely, that every induced subgraph of a complete bipartite graph, no matter how large or how small, is itself a complete bipartite graph. This hereditary property was exploited at each step in the proof of Proposition 1 as we sequentially constructed a perfect matching of $K_{n,n}$.

By contrast, in a super (d, ϵ)-regular graph, we have no control over the regularity of subgraphs of size smaller than ϵn. Furthermore, even in large subgraphs, we cannot guarantee "high" minimum vertex degree, nor can we rule out the possibility of isolated vertices. Thus, it is possible that if edges are not chosen properly at each stage, then we may end up with a subgraph which contains no matching at all.

To circumvent this problem we shall construct the perfect matching as we did in the case of the complete graph, only now we shall rapidly change the strategy when we have but ϵn vertices left on each side. At this stage, and with no respect to the sets (S_i, T_i), we would be satisfied to find *any* perfect matching in the leftover graph.

After merging this smaller matching in the leftover subgraph with the larger matching already constructed, the error term in (2) is then enlarged only slightly.

One obstacle still remains: the leftover subgraph may have no perfect matching at all. To remedy this, we shall randomly enlarge the set of leftover unmatched vertices by selecting roughly $(\epsilon^{1/3}-\epsilon)n$ vertices from the set of matched vertices in one vertex class (along with their matched partners in the other vertex class). We shall then "unmatch" these vertices and add them to the leftover graph to form an induced subgraph. The randomness guarantees (by Chernoff's bound) that the resulting $\epsilon^{2/3}$-regular graph will have sufficiently large minimum degree, and hence will contain a perfect matching. Of course, this step must be derandomized as well, but this can easily be accomplished by applying Lemma 2.3. Once we construct a perfect matching in the enlarged leftover subgraph (using any algorithm we like), then the construction of σ is complete. The term $\epsilon^{1/3}n$ is then incorporated into the error term in (2).

THEOREM 4.1 (derandomized catching lemma). *Let $\epsilon > 0$, $1 > \gamma > 3\epsilon^{1/3}$, $\lambda > 6\epsilon^{1/4}$, and let k and n be positive integers such that*

$$2k \exp\left\{\sqrt{\epsilon}n - 2\lambda^2 n\right\} < \left(\frac{\epsilon}{e}\right)^{\epsilon n}.$$

There exists a deterministic polynomial time algorithm **CATCH** *which, given an ϵ-regular bipartite graph $G = (U, V; E)$, $|U| = |V| = n$, with minimum degree at least γn, and given k pairs of subsets $S_i \subset U$, $T_i \subset V$, $i = 1, 2, \ldots, k$, constructs a perfect matching in G such that the inequality*

$$|S_i||T_i|/n - 2\lambda n < |\sigma(S_i) \cap T_i| < |S_i||T_i|/n + 2\lambda n \tag{12}$$

holds for all $i = 1, 2, \ldots, k$.

Proof. Without loss of generality we may assume that for each $i = 1, 2, \ldots, k$, $\epsilon n < |S_i|, |T_i| < n - \epsilon n$. Otherwise the entire range of the random variable $|\sigma(S_i) \cap T_i|$ is contained in an interval of length ϵn, and since $\lambda > \epsilon$, inequality (12) would automatically be satisfied by any perfect matching σ of G. This assumption shall be used only in the proof of Claim 2 below.

Let $s_i = |S_i|$, $t_i = |T_i|$, $a_i = \lceil s_i t_i/n - \lambda n \rceil$, and $b_i = \lfloor s_i t_i/n + \lambda n \rfloor$, $i = 1, 2, \ldots, k$. First, as in the proof of Proposition 2, we shall construct a partial matching $\sigma' = \{e_1, \ldots, e_L\}$ of size $L = n - \lceil \epsilon n \rceil$, so that for all i,

$$a_i - \epsilon n \le |\sigma'(S_i) \cap T_i| \le b_i. \tag{13}$$

The obvious modification is that the next edge is chosen only from the set E_{l-1} of all edges of the subgraph of G resulting from deleting the endpoints of $e_1, \ldots, e_l$, $l = 1, \ldots, L$. By the ϵ-regularity of G, $|E_{l-1}| > (d - \epsilon)(n - l + 1)^2$, where d is the density of G.

Let $f(\sigma)$ be defined as in the proof of Proposition 2. While Claim 1 is no longer true, we shall instead prove that having chosen $e_1, \ldots, e_{l-1}$, there is always a choice of an edge e_l so that $f(e_1, \ldots, e_l) \le (1 + \sqrt{\epsilon}) f(e_1, \ldots, e_{l-1})$.

CLAIM 2. *Let E_0 be the set of all edges of G, and for each $l = 1, \ldots, L = n - \lceil \epsilon n \rceil$, let E_l be the set of all edges of the subgraph of G that remain after deleting the endpoints of $e_1, \ldots, e_l$. Then*

$$\frac{1}{|E_{l-1}|} \sum_{e_l \in E_{l-1}} f(e_1, \ldots, e_l) \le (1 + \sqrt{\epsilon}) f(e_1, \ldots, e_{l-1}).$$

We prove Claim 2 at the end of this section. As a consequence, if σ' is the partial matching of order L constructed by the algorithm described in the proof of Proposition 2 (with the obvious modification; cf. Claims 1 and 2), then

$$f(\sigma') \le (1+\sqrt{\epsilon})^L f(\emptyset) < 2k\exp\{\sqrt{\epsilon}n - 2\lambda^2 n\}. \tag{14}$$

Now we prove that (13) holds for all $i = 1, 2, \dots, k$. Let us set $\sigma_i = |\sigma'(S_i) \cap T_i|$ for convenience. Thus, $a_{i,L} = a_i - \sigma_i$ and $b_{i,L} = b_i - \sigma_i$. Suppose that for some i, (13) does not hold. If $\sigma_i > b_i$, then $b_{i,L} < 0$, and thus there exists an integer $x \notin [a_{i,L}, b_{i,L}]$ such that

$$\max\{0, s_{i,L} + t_{i,L} - \epsilon n\} \le x \le \min\{s_{i,L}, t_{i,L}\}. \tag{15}$$

Consequently, at least one term of (5) is positive, and hence

$$f(\sigma') \ge \frac{1}{\binom{\epsilon n}{s_{i,L}}} \ge 2^{-\epsilon n},$$

which contradicts (14). If, on the other hand, $\sigma_i < a_i - \epsilon n$, then $a_{i,L} > \epsilon n \ge s_{i,L} + t_{i,L} - \epsilon n$, and again there is an integer $x \notin [a_{i,L}, b_{i,L}]$ that satisfies (15). This leads to a contradiction to (14).

If we let $U' \subset U$ and $V' \subset V$ be the two sets matched by σ', then we may view σ' as a bijection $\sigma' : U' \to V'$. Let $N_1, \dots, N_n$ and $M_1, \dots, M_n$ be the neighborhoods of the vertices of U and V restricted to V' and U', respectively; that is,

$$N_j = N_G(u_j) \cap V',\ u_j \in U, \qquad M_j = N_G(v_j) \cap U',\ v_j \in V. \tag{16}$$

Now we apply algorithm **PARTITION** from Lemma 2.3 to the set V' and the input sets $N_1, \dots, N_n, \sigma'(M_1), \dots, \sigma'(M_n)$ (with $r = 2$, $m = L$, and $p = p_1 = \frac{\epsilon^{1/3} - \epsilon}{1-\epsilon}$). As a result, **PARTITION** constructs a subset R of V' of order $|R| = (\epsilon^{1/3} - \epsilon)n$ such that for all $i = 1, 2, \dots, n$,

$$|N_i \cap R| > p|N_i| - 2L^{2/3} > \frac{\epsilon^{1/3} - \epsilon}{1-\epsilon}(\gamma - \epsilon)n - 2n^{2/3} > \epsilon^{2/3} n,$$

and the same inequality holds for the sets $\sigma'(M_i)$, $i = 1, 2, \dots, n$. Thus, every vertex of G has at least $\epsilon^{2/3}n$ neighbors in R or $(\sigma')^{-1}(R)$, respectively.

Finally, consider the subgraph H of G induced by the vertex sets $R \cup (V \setminus V')$ and $(\sigma')^{-1}(R) \cup (U \setminus U')$. It is $\epsilon^{2/3}$-regular and has minimum degree at least $\epsilon^{2/3}n$. Consequently, by Fact 1, it contains a perfect matching, and we may apply, say, the Hopcroft–Karp algorithm [6] to find such a perfect matching σ'' in H. Now if σ''' is the matching obtained from σ' by removing the edges incident to the set R, then by (13), and since $\lambda > \epsilon^{1/3} + \epsilon$, the combined perfect matching $\sigma'' \cup \sigma'''$ satisfies inequality (12) for all $i = 1, 2, \dots, k$. □

Proof of Claim 2. We apply induction on l. For $l = 1$ we basically follow the proof of Claim 1, writing $e(S_i, T_i)$ instead of $s_i t_i$, and so on. However, because $\min\{s_i, t_i, n - s_i, n - t_i\} > \epsilon n$, by the ϵ-regularity of G, we obtain the bound $e(S_i, T_i) < (d + \epsilon)s_i t_i$, and so on. Thus, with the notation from the proof of Claim 1, we have

$$\frac{1}{|E_0|} \sum_{e_1 \in E_0} f(e_1) \le \frac{d+\epsilon}{d-\epsilon} \sum_{i=1}^{k} \Phi_i \le (1+\sqrt{\epsilon}) f(\emptyset),$$

where

$$\Phi_i = \frac{s_i t_i}{n^2} \sum_x p_{i,1} + \frac{s_i(n-t_i)}{n^2} \sum_x p_{i,2} + \frac{(n-s_i)t_i}{n^2} \sum_x p_{i,3} + \frac{(n-s_i)(n-t_i)}{n^2} \sum_x p_{i,4}.$$

The last inequality results from the fact that $d \geq \gamma \geq 2\sqrt{\epsilon}$. Now assume that the claim is true for all $j = 1, 2, \ldots, l-1$, and let

$$\Phi_i = \sum_{e_l \in E_{l-1}} \sum_{x \notin [a_{i,l}, b_{i,l}]} \frac{\binom{t_{i,l}}{x}\binom{n_l - t_{i,l}}{s_{i,l} - x}}{\binom{n_l}{s_{i,l}}}, \qquad \Psi_i = \sum_{x \notin [a_{i,l-1}, b_{i,l-1}]} \frac{\binom{t_{i,l-1}}{x}\binom{n_{l-1} - t_{i,l-1}}{s_{i,l-1} - x}}{\binom{n_{l-1}}{s_{i,l-1}}}.$$

In this setting we shall prove that $\Phi_i \leq (1+\sqrt{\epsilon})|E_{l-1}|\Psi_i$.

Towards this end, note that $\Psi_i = 0$ if and only if

$$a_{i,l-1} \leq \max\{0, s_{i,l-1} + t_{i,l-1} - n_{l-1}\} \leq \min\{s_{i,l-1}, t_{i,l-1}\} \leq b_{i,l-1} \ .$$

However, then for every $e_l \in E_{l-1}$, we also have

$$a_{i,l} \leq \max\{0, s_{i,l} + t_{i,l} - n_l\} \leq \min\{s_{i,l}, t_{i,l}\} \leq b_{i,l} \ ,$$

and so $\Phi_i = 0$. In other words, if at some point the above inequality is satisfied for a particular pair (S_i, T_i), then it is irrelevant with respect to that pair how the construction of σ proceeds in the future. Inequality (2) is guaranteed to hold in such a case since at any given time l the number of edges of σ that could be caught by the pair (S_i, T_i) lies between $s_{i,l} + t_{i,l} - n_l$ and $\min\{s_{i,l}, t_{i,l}\}$.

If $\Psi_i > 0$, then a straightforward calculation shows that

$$\min\{s_{i,l-1}, t_{i,l-1},\ n - s_{i,l-1}, n - t_{i,l-1}\} > \epsilon n;$$

hence, we can essentially repeat the proof of the case $l = 1$ (or Claim 1 for that matter). Indeed, there exists $x \notin [a_{i,l}, b_{i,l}]$ such that $\binom{t_{i,l-1}}{x}\binom{n_{l-1}-t_{i,l-1}}{s_{i,l-1}-x} \geq 1$. Now suppose that $s_{i,l-1} < \epsilon n$. Then

$$f(e_1, \ldots, e_{l-1}) \geq \frac{1}{\binom{n_{l-1}}{s_{i,l-1}}} \geq \frac{1}{\binom{n}{s_{i,l-1}}} > \left(\frac{\epsilon}{e}\right)^{\epsilon n}.$$

Clearly, the same inequality is obtained if $n_{l-1} - s_{i,l-1} < \epsilon n$ (by the symmetry of the binomial coefficients) and also if $t_{i,l-1} < \epsilon n$ or $n_{l-1} - t_{i,l-1} < \epsilon n$ (by the symmetry of the hypergeometric distribution). However, by the induction assumption we have

$$f(e_1, \ldots, e_{l-1}) < (1+\sqrt{\epsilon})^{l-1} f(\emptyset) < 2k \exp\{\sqrt{\epsilon} n - 2\lambda^2 n\},$$

which, by our assumption, is less than $\left(\frac{\epsilon}{e}\right)^{\epsilon n}$. This completes the proof. □

We wrap up this section with a more formal description of algorithm **CATCH**; a different version of this algorithm was outlined in [14].

Algorithm **CATCH.**
Input: An ϵ-regular bipartite graph $G = (U, V; E)$, $|U| = |V| = n$, with minimum degree at least γn and subsets $S_i \subset U$, $T_i \subset V$, $i = 1, 2, \ldots, k$, as defined in Theorem 4.1.
Output: A perfect matching $\sigma : U \to V$ such that for each $i = 1, 2, \ldots, k$, inequality (12) holds.

1. Let $\sigma' = \emptyset$, $L = n - \lceil \epsilon n \rceil$.
2. For $l = 1, 2, \ldots, L$, and given a partial matching $\sigma' = \{e_1, \ldots, e_{l-1}\}$,
   (i) choose an edge e_l so that the function $f(e_1, \ldots, e_l)$ defined in (6) is minimized;
   (ii) replace σ' by $\sigma' \cup \{e_l\}$;
   (iii) let U' and V' be the sets matched by σ'.
3. Apply **PARTITION** to the set V' and the input sets $N_1, \ldots, N_n, \sigma'(M_1), \ldots, \sigma'(M_n)$ defined in (16) and denote the output by R.
4. Apply the Hopcroft–Karp algorithm to the subgraph induced by the sets $R \cup (V \setminus V')$ and $(\sigma')^{-1}(R) \cup (U \setminus U')$ and denote the output by σ''.
5. Let $\sigma = \sigma'' \cup (\sigma' \setminus \{e : e \text{ is incident with } R\})$.

5. An algorithmic version of the blow-up lemma. In [8], Komlós, Sárközy, and Szemerédi proved a striking result called the blow-up lemma that, loosely speaking, enables one to embed any bounded degree graph H as a spanning subgraph of a large super-regular graph G. By an *embedding*, we mean a bijection $f : V(H) \to V(G)$ such that if $x_1x_2 \in E(H)$, then $f(x_1)f(x_2) \in E(G)$. Their proof was based on a probabilistic argument and, subsequently, they derandomized their approach to provide a deterministic embedding [9]. It is worth noting that the blow-up lemma has been used to affirmatively answer two well-known conjectures, the approximate form of the Pósa–Seymour conjecture on squares of Hamiltonian cycles and the Alon–Yuster conjecture on graph packings (see [10] and [11], respectively).

In [13], an alternative probabilistic proof of the blow-up lemma was presented. In this probabilistic version, first the vertex set of H is partitioned using the Hajnal–Szemerédi theorem (see [5]). Then, a corresponding partition of $V(G)$ is achieved with a random partition. Finally, a version of the catching lemma (Theorem 2.2 above) is used to embed H into G. In this section, we give a constructive proof of the blow-up lemma by derandomizing the approach just described. First, instead of using the Hajnal–Szemerédi theorem to obtain the partition of $V(H)$, we shall use the constructive Sauer–Spencer packing lemma (see [15]). Then to construct the partition of $V(G)$, we shall use a deterministic polynomial time procedure based on Lemma 2.3 (see section 2). Finally, with the derandomized catching lemma (Theorem 4.1) in hand, we shall then embed H into G by finding suitable perfect matchings in super-regular bipartite graphs. Consequently, we prove the following theorem.

THEOREM 5.1 (algorithmic blow-up lemma). *Given positive integers r and Δ, and $0 < d < 1$, let $\delta = \delta(r, \Delta, d) > 0$ be a sufficiently small real number and $n_0 = n_0(\delta)$ a sufficiently large integer. There exists an algorithm* **EMBED** *which, in time polynomial in n, does the following. Let G be an r-partite graph with partition sets $V_1, \ldots, V_r$ such that*

(i) $|V_i| = n \geq n_0$, $i = 1, \ldots, r$,
(ii) *all* $\binom{r}{2}$ *bipartite subgraphs $G[V_i, V_j]$ are super (d, δ)-regular.*

Let H be an r-partite graph with partition sets $X_1, \ldots, X_r$ such that

(iii) $|X_i| = n$, $i = 1, \ldots, r$,
(iv) $\Delta(H) \leq \Delta$.

Then **EMBED** *constructs an embedding f of H into G that maps X_i onto V_i, $i = 1, \ldots, r$.*

Algorithm **EMBED** consists of two phases, which we shall first outline here and then describe in more detail later in this section.

In the preliminary phase (Phase I), the partitions of $V(H)$ and $V(G)$ are refined, and then edges are added to H and G. For clarity, we assume that n is divisible by $2\Delta^2$; otherwise, a simple adjustment is necessary (see [13]). Then with $t = 2\Delta^2$, $m = n/t$, $s = rt$, and $\epsilon = t\delta$, Phase I results in graphs H' and G' that satisfy the following conditions. G' is an s-partite graph with partition sets $W_1, \ldots, W_s$ such that

(i′) $|W_i| = m$, $i = 1, \ldots, s$,
(ii′) all $\binom{s}{2}$ bipartite subgraphs $G'[W_i, W_j]$ are super (d, ϵ)-regular.

H' is an s-partite graph with partition sets $Y_1, \ldots, Y_s$ such that

(iii′) $|Y_i| = m$, $i = 1, \ldots, s$,
(iv′) all $\binom{s}{2}$ bipartite subgraphs $H'[Y_i, Y_j]$ are perfect matchings.

In Phase II, **EMBED** recursively constructs an embedding f of H' into G' that maps Y_i onto W_i, $i = 1, \ldots, s$, so that the edges added to G in Phase I can be only images of the edges that were added to H. Hence, f is the desired embedding.

To execute Phase I, we invoke four subroutines: **PACK**, **PARTITION** (from Lemma 2.3), **SATURATE**, and **AUGMENT**. The last two algorithms are simple and shall be described below as we go.

Procedure **PACK** is an algorithmic version of a graph packing result by Sauer and Spencer presented in [15]. Given two graphs Γ and Γ' on the same vertex set V, we say that a bijection $\pi : V \to V$ is a *packing of* Γ *and* Γ' if $E(\Gamma) \cap \pi(E(\Gamma')) = \emptyset$, where $\pi(E(\Gamma')) = \{\pi(u)\pi(v) : uv \in E(\Gamma')\}$. It is straightforward to show that Sauer and Spencer's original existence proof yields the following result (see [17] for the details).

LEMMA 5.2. *There exists a polynomial time algorithm* **PACK** *that finds a packing of two given n-vertex graphs, Γ and Γ', provided $2\Delta(\Gamma)\Delta(\Gamma') < n$.* □

In Phase I, **PACK** shall be used to pack vertex disjoint cliques onto the squares of certain graphs. Recall that given a graph F, the *square of* F, denoted by F^2, is the graph obtained from F by joining with an edge each pair of distinct vertices of F whose distance is at most two. By definition of F^2, any pair of vertices in an independent set I of F^2, are in F at distance at least three from each other. Therefore the subgraph of F induced by two such sets is a matching.

PHASE I.

Phase I takes as its input the graphs H and G which satisfy properties (i)–(iv) in Theorem 5.1. First, we use algorithm **PACK** to partition each set $X_i \subset V(H)$, $i = 1, 2, \ldots, r$, into $t = 2\Delta^2$ sets $X_{i,j}$, $j = 1, \ldots, t$, of size m, where $m = n/t$, so that each pair of distinct sets $(X_{i_1,j_1}, X_{i_2,j_2})$ induces a (possibly empty) matching in H. For a fixed set X_i, this is achieved by applying **PACK** to the graphs $\Gamma = H^2[X_i]$ and Γ', where $\Gamma = H^2[X_i]$ is the subgraph of H^2 induced by X_i and where Γ' denotes the vertex-disjoint union of t cliques of order m. Note that since $\Delta(\Gamma) \leq \Delta(\Delta - 1)$ and $\Delta(\Gamma') \leq m - 1$, the hypothesis $2\Delta(\Gamma)\Delta(\Gamma') < n$ of Lemma 5.2 indeed holds. As a result, the procedure packs Γ' onto Γ and thus splits each X_i into t subsets $X_{i,j}$, each of which has size m, and more importantly, each of which is independent in H^2. Since the vertices of an independent set in H^2 are at a distance of at least three from each

other in H, any subgraph of H induced by two such sets X_{i_1,j_1}, X_{i_2,j_2} is a (possibly empty) matching.

In the next step of Phase I, we construct a corresponding finer partition of G. For this, we use algorithm **PARTITION** from Lemma 2.3 to partition each set V_i, $i = 1, \dots, r$, into t sets $V_{i,j}$, $j = 1, 2, \dots, t$, of size $m = n/t$, and so that each pair of sets $(V_{i_1,j_1}, V_{i_2,j_2})$ with $i_1 \neq i_2$ induces a super $(d, t\delta)$-regular subgraph. More precisely, to each set V_i, we apply **PARTITION** with $p_j = t^{-1}$, $j = 1, \dots, t$, and with the sets N_v where, for each $v \notin V_i$, $N_v = N_G(v) \cap V_i$ (cf. Lemma 2.3 above). As a result, for each $i = 1, \dots, r$ we obtain a partition $V_i = \bigcup_{j=1}^{t} R_{i,j}$ with all sets $R_{i,j}$ of size m such that all induced subgraphs $G[R_{i_1,j_1}, R_{i_2,j_2}]$, $i_1 \neq i_2$, have minimum and maximum degree bounded between $(d - \delta)m - n^{2/3}$ and $(d + \delta)m + n^{2/3}$.

By the δ-regularity of G, all resulting pairs $(V_{i_1,j_1}, V_{i_2,j_2})$ with $i_1 \neq i_2$ induce in G $t\delta$-regular subgraphs. Since for large n, we have $\delta + o(1) < t\delta$, we conclude that all pairs $(V_{i_1,j_1}, V_{i_2,j_2})$ in G induce super $(d, t\delta)$-regular subgraphs.

The last two stages of Phase I involve two simple procedures called **SATURATE** and **AUGMENT**, which are performed solely to unify all steps of the recursive embedding in Phase II. For each pair $(X_{i_1,j_1}, X_{i_2,j_2})$ of sets in H that does not induce a perfect matching, procedure **SATURATE** adds edges between the pair so that a perfect matching is induced. The resulting graph is denoted by H'.

Similarly, between each pair of sets $(V_{i_1,j_1}, V_{i_2,j_2})$ in G that induces a subgraph with density 0 (i.e., for which $i_1 = i_2$), procedure **AUGMENT** inserts edges so that the pair induces a super $(d, t\delta)$-regular subgraph. For this purpose, we use the same (our favorite) super $(d, t\delta)$-regular graph on $2m$ vertices. The resulting graph is denoted by G'.

For convenience, we denote the new partition sets by $Y_1, \dots, Y_s$ and $W_1, \dots, W_s$, with $s = rt$, respectively. Hence, at the conclusion of Phase I we have two graphs H' and G', and partitions $V(H') = Y_1 \cup \cdots \cup Y_s$ and $V(G') = W_1 \cup \cdots \cup W_s$, which satisfy conditions (i′)–(iv′).

PHASE II.

This phase takes as its input the graphs H' and G' and their partitions that satisfy conditions (i′)–(iv′). In this phase, algorithm **EMBED** recursively constructs an embedding of H' into G' so that each Y_j is mapped onto the set W_j. Note that every such embedding yields the desired embedding of H into G. Indeed, every edge from $E(G') \setminus E(G)$ connects two sets $(V_{i_1,j_1}, V_{i_2,j_2})$ with $i_1 = i_2$, and thus it can only be the image of an edge from $E(H') \setminus E(H)$.

The main procedure involved in constructing the embedding is algorithm **CATCH** from Theorem 4.1; for technical reasons which shall be described later, another subroutine called **DELETE** is also used.

Before describing how the embedding is constructed, we introduce some definitions and notation. For convenience, we relabel H' and G' as H and G, respectively. For every $1 \leq j \leq s - 1$ and each vertex $x \in Y_{j+1} \cup \cdots \cup Y_s$, let $N_j(x)$ denote the set of precisely j neighbors of x which belong to $Y_1 \cup \cdots \cup Y_j$. Given a bijection f_j between $Y_1 \cup \cdots \cup Y_j$ and $W_1 \cup \cdots \cup W_j$, let $M_j(x) = f_j(N_j(x))$. Given f_j, for each $l = j+1, \dots, s$, we define a bipartite auxiliary graph A_j^l with bipartition (Y_l, W_l) and edge set

$$E(A_j^l) = \{xw : x \in Y_l, w \in W_l \text{ and } uw \in E(G) \text{ for each } u \in M_j(x)\}. \tag{17}$$

We call the graphs A_j^l *candidacy graphs* because the edges of A_j^l join a vertex $x \in Y_l$ to all vertices of W_l which, after f_j embeds $Y_1 \cup \cdots \cup Y_j$ onto $W_1 \cup \cdots \cup W_j$, are still

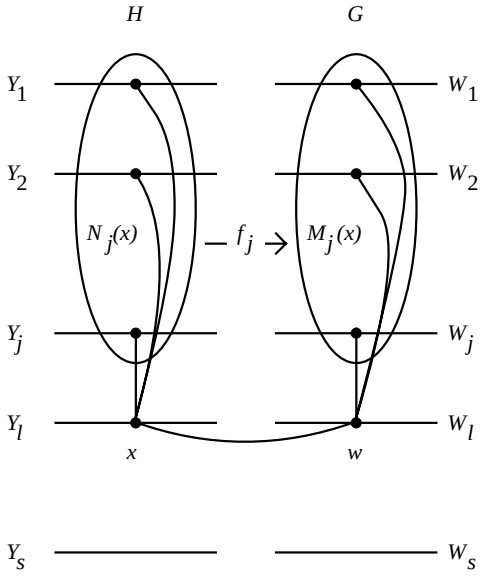


FIG. 2. $xw \in E(A_j^l)$.

good candidates for the image of x (see Figure 2).

Said another way, every neighbor w of x in A_j^l is joined in G to the images of the already embedded neighbors of x, and thus $f_l(x)$ could be set to w if the lth step (i.e., Y_l being mapped onto W_l) were next.

The construction of the embedding proceeds as follows. Let f_1 be any bijection between Y_1 and W_1. At this initial stage the graph A_1^l is isomorphic to $G[W_1, W_l]$ for all $l = 2, \ldots, s$, and thus it is super (d, ϵ)-regular. We set $\epsilon_1 = \epsilon$.

Assume that there exists an embedding f_j of $H[Y_1 \cup \cdots \cup Y_j]$ into $G[W_1 \cup \cdots \cup W_j]$ such that the candidacy graphs A_j^l, $l = j+1, \ldots, s$, are super (d^j, ϵ_j)-regular. Our goal is to extend f_j to f_{j+1} by constructing a perfect matching $\sigma_{j+1} : Y_{j+1} \to W_{j+1}$ in A_j^{j+1} such that all graphs A_{j+1}^l, $l = j+2, \ldots, s$, are super $(d^{j+1}, \epsilon_{j+1})$-regular for some ϵ_{j+1} to be fixed later. Then we shall set $f_{j+1}(x) = f_j(x)$ for $x \in Y_1 \cup \cdots \cup Y_j$ and $f_{j+1}(x) = \sigma_{j+1}(x)$ for $x \in Y_{j+1}$. The requirement that all graphs A_{j+1}^l are super $(d^{j+1}, \epsilon_{j+1})$-regular serves to carry the recursion through.

When this procedure reaches the stage $j = s-1$, then $j+2 > s$ and thus there are no candidacy graphs whose super-regularity must be maintained in the future. At this point, all that is needed is *any* perfect matching σ_s in A_{s-1}^s, and we use our favorite algorithm from [6] to find one. Then $f = f_s$ is the desired embedding of H into G.

To guarantee that there is at least one perfect matching in A_{s-1}^s, we use the fact that, by our construction, A_{s-1}^s is super $(d^{s-1}, \epsilon_{s-1})$-regular. A sufficiently small choice of δ (cf. Theorem 5.1) ensures that $d^{s-1} > 2\epsilon_{s-1}$. Thus, the minimum degree in A_{s-1}^s is at least $d^{s-1}m - \epsilon_{s-1}m > \epsilon_{s-1}m$, and by Fact 1, this last candidacy graph contains a perfect matching.

In light of the preceding discussion, it suffices to show how to extend f_j to an embedding f_{j+1} such that all graphs A_{j+1}^l, $l = j+2, \ldots, s$, will be super $(d^{j+1}, \epsilon_{j+1})$-regular. Let us fix $2 \le j < s-1$. For every $j+1 < l \le s$ let B^l be the bipartite graph

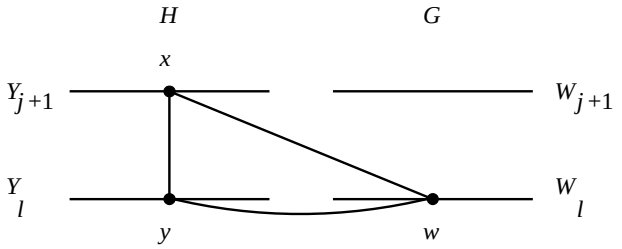


FIG. 3. $xw \in E(B^l)$.

with bipartition (Y_{j+1}, W_l) such that $xw \in E(B^l)$ if and only if $yw \in E(A_j^l)$, where y is the unique neighbor of x in Y_l (see Figure 3).
Observe that B^l is isomorphic to A_j^l. Also, set $G^l = G[W_{j+1}, W_l]$.

In preparation for finding a perfect matching in the candidacy graph A_j^{j+1} which would ensure the super-regularity of all future candidacy graphs, we apply procedure **DELETE** to remove every edge $e = xw$ of A_j^{j+1} for which the inequality

$$(d^{j+1} - 2\epsilon_j)m < |N_{B^l}(x) \cap N_{G^l}(w)| < (d^{j+1} + 2\epsilon_j)m \tag{18}$$

is violated for some $l = j+2, \ldots, s$. (We omit a formal description of this procedure.) As shown in [13], the resulting subgraph, which we denote by $\bar{A}_j^{j+1}$, is super $(d^j, 2s\sqrt{\epsilon_j})$-regular.

Note that the minimum degree in $\bar{A}_j^{j+1}$ is at least γm, where $\gamma = d^j - 2s\sqrt{\epsilon_j}$. We choose δ in Theorem 5.1 so small that $\gamma > 3(2s\sqrt{\epsilon_j})^{1/3}$, and we set $\lambda = \lambda_j - 7(2s\sqrt{\epsilon_j})^{1/4}$. These choices guarantee that the hypotheses of Theorem 4.1 are satisfied.

At this point, the stage is almost set for algorithm **CATCH** to find the desired perfect matching σ_{j+1} in $\bar{A}_j^{j+1}$.

We apply **CATCH** to the graph $\bar{A}_j^{j+1}$, where the sets S_i are the sets $N_{B^l}(w)$ and $N_{B^l}(w_1) \cap N_{B^l}(w_2)$, and the sets T_i are, respectively, the sets $N_{G^l}(w)$ and $N_{G^l}(w_1) \cap N_{G^l}(w_2)$ for all $w \in W_l$ and all pairs w_1, w_2 of vertices of W_l, $l = j+2, \ldots, s$. (Note that $k \le m + m^2$.)

As a result, a perfect matching $\sigma = \sigma_{j+1}$ of $\bar{A}_j^{j+1}$ is constructed so that for all $l = j+2, \ldots, s$, all $w \in W_l$, and all pairs w_1, w_2 of vertices of W_l,

$$\begin{aligned} |N_{B^l}(w)||N_{G^l}(w)|/m - 2\lambda m &< |\sigma(N_{B^l}(w)) \cap N_{G^l}(w)| \\ &< |N_{B^l}(w)||N_{G^l}(w)|/m + 2\lambda m \end{aligned} \tag{19}$$

and

$$\begin{aligned} &|N_{B^l}(w_1) \cap N_{B^l}(w_2)||N_{G^l}(w_1) \cap N_{G^l}(w_2)|/m - 2\lambda m \\ &< |\sigma(N_{B^l}(w_1) \cap N_{B^l}(w_2)) \cap (N_{G^l}(w_1) \cap N_{G^l}(w_2))| \\ &< |N_{B^l}(w_1) \cap N_{B^l}(w_2)||N_{G^l}(w_1) \cap N_{G^l}(w_2)|/m + 2\lambda m, \end{aligned} \tag{20}$$

where $|\sigma(S) \cap T|$ stands for the number of edges of σ that connect a vertex in S with a vertex in T.

It remains to be shown that for each $l = j+2, \ldots, s$, the graph A_{j+1}^l is super $(d^{j+1}, \epsilon_{j+1})$-regular, where $\epsilon_{j+1} = (64\lambda)^{1/5}$. Let B_σ^l be the subgraph of B_l such that $xw \in E(B_\sigma^l)$ if and only if $xw \in E(B^l)$ and $\sigma(x)w \in E(G^l)$. By definition, the graph B_σ^l is isomorphic to A_{j+1}^l (see Figure 4).

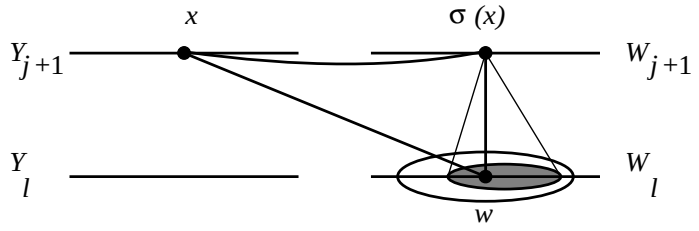


FIG. 4. $xw \in E(B^l_\sigma)$.

We now make three crucial observations.

(a) For every vertex $x \in Y_{j+1}$,

$$deg_{B^l_\sigma}(x) = |N_{G^l}(\sigma(x)) \cap N_{B^l}(x)|.$$

(b) For every vertex $w \in W_l$,

$$deg_{B^l_\sigma}(w) = |\sigma(N_{B^l}(w)) \cap N_{G^l}(w)|.$$

(c) For every pair of vertices $w_1, w_2 \in W_l$,

$$|N_{B^l_\sigma}(w_1) \cap N_{B^l_\sigma}(w_2)| = |\sigma(N_{B^l}(w_1) \cap N_{B^l}(w_2)) \cap (N_{G^l}(w_1) \cap N_{G^l}(w_2))|.$$

These observations shall be used in conjunction with Lemma 2.1 to verify the super-regularity of the graphs A^l_{j+1}.

Towards this end, inequality (18) guarantees that for every perfect matching σ of $\bar{A}^{j+1}_j$, and for each l, the degree $deg_{B^l_\sigma}(x)$ of each vertex $x \in Y_{j+1}$ satisfies

$$(d^{j+1} - 2\epsilon_j)m < deg_{B^l_\sigma}(x) < (d^{j+1} + 2\epsilon_j)m. \tag{21}$$

Also, since G^l is super (d, ϵ)-regular, the following inequality holds for all but at most ϵm^2 pairs $w_1, w_2 \in W_l$:

$$|N_{G^l}(w_1) \cap N_{G^l}(w_2)| \leq (d^2 + \epsilon)m. \tag{22}$$

Similarly, since B^l is super (d^j, ϵ_j)-regular, it follows that for all but $\epsilon_j m^2$ pairs $w_1, w_2 \in W_l$, we have

$$|N_{B^l}(w_1) \cap N_{B^l}(w_2)| \leq (d^{2j} + \epsilon_j)m. \tag{23}$$

We call a pair $w_1, w_2 \in W_l$ *good* if both (22) and (23) hold. Hence, in total there are at least $(1 - 2\epsilon_j)m^2$ good pairs in W_l.

By the super (d, ϵ)-regularity of G^l and the super (d^j, ϵ_j)-regularity of B^l, we have $|N_{G^l}(w)| > (d - \epsilon)m$ and $|N_{B^l}(w)| > (d^j - \epsilon_j)m$. Hence, conditions (b) and (19) imply that for each $w \in W_l$

$$\begin{aligned} deg_{B^l_\sigma}(w) &= |\sigma(N_{B^l}(w)) \cap N_{G^l}(w)| \\ &> |N_{B^l}(w)||N_{G^l}(w)|/m - 2\lambda m \\ &> (d^{j+1} - 4\lambda)m. \end{aligned}$$

On the other hand, by (c) and (20), for all good pairs $w_1, w_2 \in W_l$ we have

$$\begin{aligned}|N_{B^l_\sigma}(w_1) \cap N_{B^l_\sigma}(w_2)| &= |\sigma(N_{B^l}(w_1) \cap N_{B^l}(w_2)) \cap (N_{G^l}(w_1) \cap N_{G^l}(w_2))| \\ &< (d^{j+1} + 4\lambda)^2 m.\end{aligned}$$

It follows from Lemma 2.1 that for all $l = j+2, \ldots, s$, the graph B^l_σ, and thus the graph A^l_{j+1} (since they are isomorphic), is super $(d^{j+1}, (64\lambda)^{1/5})$-regular. We conclude this section with a more formal description of algorithm **EMBED**.

Algorithm **EMBED.**
Input: r-partite graphs H and G as in Theorem 5.1.
Output: An embedding f of H into G so that each X_i is mapped onto V_i, $i = 1, \ldots, r$.
Phase I:

1. Apply **PACK** to $V(H)$ and **PARTITION** to $V(G)$ and denote the output by $V(H) = Y_1 \cup \cdots \cup Y_s$ and $V(G) = W_1 \cup \cdots \cup W_s$, where $|Y_1| = |W_1| = \cdots = |Y_s| = |W_s| = m$.
2. Apply **SATURATE** to H and **AUGMENT** to G and denote the output by H' and G'.
3. Reset H and G to H' and G', respectively.

Phase II:

1. Let f_1 be any bijection from Y_1 to W_1.
2. For $j = 1, 2, \ldots, s-2$,
   (a) apply **DELETE** to A^{j+1}_j and denote the output by $\bar{A}^{j+1}_j$;
   (b) apply **CATCH** to the graph $\bar{A}^{j+1}_j$ and to the pairs $(N_{B^l}(w),\ N_{G^l}(w))$, $w \in W_l$, $l = j+2, \ldots, s$, $(N_{B^l}(w_1) \cap N_{B^l}(w_2),\ N_{G^l}(w_1) \cap N_{G^l}(w_2))$ for all good pairs $w_1, w_2 \in W_l$, $l = j+2, \ldots, s$, and denote the output by σ_{j+1};
   (c) set $f_{j+1}(x) = \begin{cases} f_j(x) & \text{if } x \in Y_1 \cup \cdots \cup Y_j, \\ \sigma_{j+1}(x) & \text{if } x \in Y_{j+1}. \end{cases}$
3. Use the Hopcroft–Karp algorithm to construct a perfect matching σ_s in A^s_{s-1} and set
   $f(x) = \begin{cases} f_{s-1}(x) & \text{if } x \in Y_1 \cup \cdots \cup Y_{s-1}, \\ \sigma_s(x) & \text{if } x \in Y_s. \end{cases}$

Acknowledgments. The authors extend thanks to Alan Frieze and an anonymous referee for their valuable comments and suggestions. The authors also thank Dwight Duffus and Michal Karoński for their generous readings of earlier versions of this manuscript.

REFERENCES

[1] N. Alon, R. Duke, H. Leffman, V. Rödl, and R. Yuster, *The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109.
[2] N. Alon, V. Rödl, and A. Ruciński, *Perfect matchings in ϵ-regular graphs*, Electronic J. Combin., 5 (1998).
[3] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley, New York, 1992.
[4] P. Erdős and J. L. Selfridge, *On a combinatorial game*, J. Combin. Theory Ser. A, 14 (1973), pp. 298–301.
[5] A. Hajnal and E. Szemerédi, *Proof of a conjecture of Erdős*, in Combinatorial Theory and Its Applications, Vol. II, P. Erdős, A. Rényi, and V.T. Sós eds., Colloq. Math. Soc. János Bolyai 4, North-Holland, Amsterdam, 1970, pp. 601-623.

[6] J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225–231.
[7] S. Janson, T. Łuczak, and A. Ruciński, *Random Graphs*, Wiley, New York, 2000.
[8] J. Komlós, G. N. Sárközy, and E. Szemerédi, *Blow-up lemma*, Combinatorica, 17 (1997), pp. 109–123.
[9] J. Komlós, G. N. Sárközy, and E. Szemerédi, *An algorithmic version of the blow-up lemma*, Random Structures Algorithms, 12 (1998), pp. 297–312.
[10] J. Komlós, G. N. Sárközy, and E. Szemerédi, *On the Pósa-Seymour conjecture*, J. Graph Theory, 29 (1998), pp. 167–176.
[11] J. Komlós, G. N. Sárközy, and E. Szemerédi, *Proof of the Alon-Yuster conjecture*, submitted.
[12] J. Komlós, G. N. Sárközy, and E. Szemerédi, *Proof of the Seymour conjecture for large graphs*, Ann. Combin., 2 (1998), pp. 43–60.
[13] V. Rödl and A. Ruciński, *Perfect matchings in ϵ-regular graphs and the blow-up lemma*, Combinatorica, 19 (1999), pp. 437–452.
[14] V. Rödl, A. Ruciński, and M. Wagner, *An algorithmic embedding of graphs via* 1*-factors*, in Randomization and Approximation Techniques in Computer Science, M. Luby, J. Rolim, and M. Serna, eds., Lecture Notes in Comput. Sci. 1518, Springer, Berlin, 1998, pp. 25–34.
[15] N. Sauer and J. Spencer, *Edge disjoint placement of graphs*, J. Combin. Theory Ser. B, 25 (1978), pp. 295–302.
[16] E. Szemerédi, *Partitions of graphs*, in Problèmes Combinatoires et theorie des graphes, Colloq. Internat. CNRS 260, 1978, pp. 399–401.
[17] M. Wagner, *The Derandomization of the Blow-up Lemma*, Ph.D. thesis, Emory University, Atlanta, GA, 1999.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 447–459

OPTIMAL BUY-AND-HOLD STRATEGIES FOR FINANCIAL MARKETS WITH BOUNDED DAILY RETURNS*

GEN-HUEY CHEN†, MING-YANG KAO‡, YUH-DAUH LYUU†, AND HSING-KUO WONG†

Abstract. In the context of investment analysis, we formulate an abstract online computing problem called a *planning game* and develop general tools for solving such a game. We then use the tools to investigate a practical *buy-and-hold trading* problem faced by long-term investors in stocks. We obtain the unique optimal static online algorithm for the problem and determine its exact competitive ratio. We also compare this algorithm with the popular dollar averaging strategy using actual market data.

Key words. buy-and-hold trading problems, balanced strategy, dollar averaging strategy, online algorithms, competitive analysis, planning games, minimax theorem, linear programming, zero-sum two-person games

AMS subject classifications. 05A15, 15A09, 15A23, 05A99, 60C05, 68R05, 90A09, 90A12, 90D10, 90D13

PII. S0097539799358847

1. Introduction.

In an *online* problem, an *online* algorithm $\mathcal{B}$ is given one input at a time from a sequence of inputs. $\mathcal{B}$ takes an action on each input before seeing any remaining input. In contrast, an *offline* algorithm sees the entire input sequence before it takes any action. Each action yields a positive *accumulation*. Let E denote the set of all admissible input sequences. Let $\mathcal{C}(\vec{e})$ denote the (expected) total accumulation of an online or offline algorithm $\mathcal{C}$ on $\vec{e} \in E$. Let $\mathcal{A}$ denote the *optimal* offline algorithm, i.e., one that produces the largest total accumulation on each admissible input sequence. In competitive analysis [4, 25, 27], $\mathcal{B}$'s performance is measured by its *competitive ratio*

$$\Upsilon_{\mathcal{B}} = \sup_{\vec{e} \in E} \frac{\mathcal{A}(\vec{e})}{\mathcal{B}(\vec{e})}. \tag{1.1}$$

The *online player* seeks to minimize this ratio by choosing a suitable $\mathcal{B}$, while the *adversary* attempts to maximize it by picking $\vec{e}$ after examining $\mathcal{B}$. This paper assumes that the adversary is *oblivious*, i.e., it fixes the input sequence before $\mathcal{B}$ performs any computation such as generating random bits.

A *planning game* is an abstract online problem where the length of the input sequence is fixed and known a priori to $\mathcal{B}$. This time horizon feature captures many important online problems including those for portfolio rebalancing [7, 8, 21], asset trading [2, 5, 11, 12], secretary selection [1, 6, 13, 16], and bipartite matching [15, 17]. A planning game is *finite* if the numbers of admissible sequences of actions and inputs are both finite; otherwise, it is *infinite*. A finite planning game corresponds to a linear programming problem, where an optimal randomized online algorithm corresponds to

*Received by the editors July 20, 1999; accepted for publication (in revised form) July 11, 2000; published electronically August 22, 2001. A preliminary version appeared *in Proceedings of the* 31*st Annual ACM Symposium on Theory of Computing*, Atlanta, GA, 1999, pp. 119–128.

http://www.siam.org/journals/sicomp/31-2/35884.html

†Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Republic of China. The research of the third and fourth authors was supported in part by NSC grant 87-2416-H-002-031.

‡Department of Computer Science, Yale University, New Haven, CT 06520 (kao-ming-yang@cs.yale.edu). The research of this author was supported in part by NSF grant CCR-9531028.

Exchange	Circuit Breaker	
	α^{-1}	β
Amsterdam	90%	110%
Bangkok	90%	110%
Paris	95%	110%
Taipei	93%	107%
Tel-Aviv	95%	110%
Tokyo	95%	130%
Vienna	95%	105%

FIG. 1. *Circuit breaker rules in various exchanges.*

an optimal feasible solution. Consequently, we can show that the smallest competitive ratio of any randomized online algorithm for such a game is the reciprocal of the value of the game as a zero-sum two-person game.

In this general optimization framework, we investigate the *buy-and-hold trading problem* defined as follows. An investor starts with some capital, which is normalized to one dollar, and trades it for a certain security over n days, which is referred to as the *investment horizon*. To avoid triviality, we assume $n \geq 2$. On each day, the security has only one *exchange rate*, i.e., the number of shares of the security which one unit of capital can buy. Upon seeing the exchange rate, the investor executes one transaction for that day and may trade all or part of the remaining capital. All the capital must be traded by the nth day, and converting the acquired security back to capital is prohibited. The total *accumulation* of the investor is the number of shares of the security she accumulates at the end of the investment horizon. Note that the competitive ratio between the adversary's and the investor's accumulations is exactly the competitive ratio between the dollar values of the accumulations. This problem is faced by millions of investors who save for retirement purposes on a long-term basis; for instance, a widely popular security for today's investors would be a stock index fund.

We employ the *bounded daily return model*, in which the next day's exchange rate e' depends on the current day's exchange rate e with $e/\beta \leq e' \leq e\alpha$ for some fixed $\alpha, \beta > 1$. The values n, α, and $1/\beta$ are known a priori to the investor. We call α and $1/\beta$ the *daily return bounds*. Figure 1 gives some stock markets which enforce such ratios through circuit breakers. This model can also be regarded as an approximation to the geometric Brownian motion model used extensively in the finance community [3, 9, 14, 18, 20, 24, 26].

A *static* algorithm is an online algorithm for the buy-and-hold trading problem such that for $1 \leq i \leq n$, the (expected) amount of dollars invested by the algorithm on the ith day is the same for all exchange rate sequences. A *dynamic* algorithm refers to any online algorithm for the problem which is not necessarily static. The *static* buy-and-hold trading problem refers to the case of the problem where the investor can use only a static algorithm.

We prove that the smallest possible competitive ratio for any randomized or deterministic static algorithm is $\frac{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}{\alpha\beta-1}$. We also obtain a deterministic static algorithm with this competitive ratio, called the *balanced strategy*, and prove that it is the only optimal deterministic static algorithm. In comparison, the popular dollar averaging strategy has a strictly greater competitive ratio and thus is not optimal. The balanced strategy is so simple that it can be executed even by those who

are not mathematically sophisticated. Starting with one dollar initially, the algorithm invests $\frac{\alpha(\beta-1)}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}$ dollar on the first day, $\frac{(\alpha-1)\beta}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}$ dollar on the last day, and $\frac{(\alpha-1)(\beta-1)}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}$ dollar on each of the other days.

Previously, El-Yaniv [10] and El-Yaniv et al. [11, 12] obtained optimal online algorithms for this unidirectional trading problem under the assumption that the daily exchange rates, instead of the daily returns, are between a pair of upper and lower bounds. Al-Binali [2] further studied the same setting in a framework of risk and reward [19]. Our model and that of El-Yaniv et al. [11, 12] are each formulated for real but different regulations of stock and foreign currency markets. A subtle difference between these models is that their model fixes a upper bound and a lower bound on the daily exchange rates globally for the entire investment horizon, while our model sets new bounds dynamically every day. Interestingly, although this difference might seem minor, they give rise to mathematical results of very distinct flavors using significantly different techniques.

Section 2 discusses how to compute optimal randomized online algorithms for finite planning games. Section 3 uses the general analysis in section 2 to derive the balanced strategy and compare it with the dollar averaging strategy. Section 4 concludes the paper with some open problems.

2. General analysis of finite planning games. A finite planning game G can be regarded as a finite zero-sum two-person game $\Gamma_H(m,n)$ defined as follows. For any integer $k>0$, let $Z_k=\{1,2,\ldots,k\}$. The maximizing player is the online player, whose pure strategies are the deterministic online algorithms $\mathcal{B}_i$ of G indexed with $i\in Z_m$. The minimizing player is the adversary of G, whose pure strategies are the input sequences $\vec{\sigma}_j$ of G indexed with $j\in Z_n$. The payoff matrix[1] H of $\Gamma_H(m,n)$ is defined by

$$H(i,j)=\frac{\mathcal{B}_i(\vec{\sigma}_j)}{\mathcal{A}(\vec{\sigma}_j)}>0,\quad i\in Z_m \text{ and } j\in Z_n. \tag{2.1}$$

Let $\Phi(Z_k)$ be the set of all probability density functions defined on Z_k. For $k=n$ or m, each $h\in\Phi(Z_k)$ is regarded as a point in the k-dimensional Euclidean space and represents a mixed strategy that applies the ℓth pure strategy indexed by Z_k with probability $h(\ell)$. By von Neumann's minimax theorem [22],

$$\begin{aligned}\max_{f\in\Phi(Z_m)}\min_{g\in\Phi(Z_n)}\sum_{i=1}^{m}\sum_{j=1}^{n}f(i)g(j)H(i,j) &= \min_{g\in\Phi(Z_n)}\max_{f\in\Phi(Z_m)}\sum_{i=1}^{m}\sum_{j=1}^{n}f(i)g(j)H(i,j)\\ &= \max_{f\in\Phi(Z_m)}\min_{j\in Z_n}\sum_{i=1}^{m}f(i)H(i,j)\\ &= \min_{g\in\Phi(Z_n)}\max_{i\in Z_m}\sum_{j=1}^{n}g(j)H(i,j),\end{aligned} \tag{2.2}$$

which is called the *value* v^* of $\Gamma_H(m,n)$.

[1] In (1.1), we use $\frac{\mathcal{A}(\vec{\sigma})}{\mathcal{B}(\vec{\sigma})}$ instead of $\frac{\mathcal{B}(\vec{\sigma})}{\mathcal{A}(\vec{\sigma})}$ so that the competitive ratios of different online algorithms are greater than 1 and therefore are easier to distinguish visually in Figures 6 and 9. In contrast, in (2.1), we choose $\frac{\mathcal{B}_i(\vec{\sigma}_j)}{\mathcal{A}(\vec{\sigma}_j)}$ instead of $\frac{\mathcal{A}(\vec{\sigma}_j)}{\mathcal{B}_i(\vec{\sigma}_j)}$ in order to simplify the linear algebra involved.

Let r^* be the smallest possible competitive ratio of any randomized online algorithm for G; i.e.,

$$r^* = \min_{f \in \Phi(Z_m)} \max_{j \in Z_n} \frac{\mathcal{A}(\vec{\sigma}_j)}{\sum_{i=1}^{m} f(i)\mathcal{B}_i(\vec{\sigma}_j)}.$$

A randomized online algorithm is *optimal* if its competitive ratio is r^*.

The next theorem relates G and $\Gamma_H(m,n)$.

THEOREM 2.1.

1. $r^* = \frac{1}{v^*}$.
2. *An optimal mixed strategy of the online player of* $\Gamma_H(m,n)$ *induces an optimal randomized online algorithm for* G *and vice versa.*

Proof. This theorem follows from (2.2). □

In light of Theorem 2.1, we use G and $\Gamma_H(m,n)$ interchangeably. A main purpose of this paper is to derive the exact value of r^* and an optimal randomized online algorithm for G. To do so by means of Theorem 2.1, the *primal* and *dual* problems of $\Gamma_H(m,n)$ or G are defined as follows:

Primal:		Dual:	
minimize	$x^T u_m$	maximize	$y^T u_n$
subject to	$x^T H \geq u_n^T$,	subject to	$Hy \leq u_m$,
	$x \geq 0$;		$y \geq 0$,

where u_k is the column vector of k copies of 1.

For each $j \in Z_n$, let H^j denote the jth column of H. Moreover, let X and Y be the sets of feasible solutions to the primal and dual problems of G, respectively. Let $\bar{X}$ and $\bar{Y}$ be the sets of optimal feasible solutions to these problems. Let X^* and Y^* be the sets of optimal mixed strategies of the online player and the adversary, respectively.

The next lemma is useful for computing an optimal randomized online algorithm for G and its competitive ratio via linear programming.

LEMMA 2.2.

1. *For all nonzero* $x \in X$ *and* $y \in Y$, $\frac{x}{x^T u_m}$ *and* $\frac{y}{y^T u_n}$ *are mixed strategies for the online player and the adversary, respectively.*
2. $\min_{x \in X} x^T u_m = r^* = \max_{y \in Y} y^T u_n$.
3. $X^* = \frac{1}{r^*} \cdot \bar{X} \neq \emptyset$, *and* $Y^* = \frac{1}{r^*} \cdot \bar{Y} \neq \emptyset$.
4. *For each nonzero* $x \in X$, *if* $j \in Z_n$ *satisfies* $x^T H^j = \min_{\ell \in Z_n} x^T H^\ell$, *then* $\vec{\sigma}_j$ *is a worst-case input sequence for the online player's mixed strategy* $\frac{x}{x^T u_m}$.

Proof. This lemma follows from Theorem 2.1 and the basics of linear programming [22]. □

The next fact is useful for analyzing the uniqueness of an optimal randomized online algorithm for G.

FACT 2.3 (see [23]). *For any* $x \in \bar{X}$ *and* $y \in \bar{Y}$, x *and* y *are extreme points of the convex polyhedra* $\bar{X}$ *and* $\bar{Y}$ *if and only if there is a square submatrix* $H' = (h_{ij})_{i \in I, j \in J}$ *of* H *for some* $I \subseteq Z_m$ *and* $J \subseteq Z_n$ *with the following properties:*

1. H' *is nonsingular.*
2. $\sum_{i \in I} h_{ij} x_i = 1$ *for all* $j \in J$.
3. $\sum_{j \in J} h_{ij} y_j = 1$ *for all* $i \in I$.
4. *For all* $i \notin I$, $x_i = 0$.
5. *For all* $j \notin J$, $y_j = 0$.

The next theorem combines Lemma 2.2 and Fact 2.3 for the case $m = n$.

THEOREM 2.4. *Assume that $m = n$ and H^{-1} exists. Let $x = (u_n^T H^{-1})^T$ and $y = H^{-1}u_n$. Further assume $x \geq 0$ and $y \geq 0$. Let $b = \frac{x}{x^T u_n}$.*

1. *Then, x and y are optimal feasible solutions to the primal and dual problems of G, respectively.*
2. *$\Upsilon_{\mathcal{B}} = r^* = x^T u_n$, where $\mathcal{B}$ is the randomized online algorithm corresponding to the online player's mixed strategy b; in other words, $\mathcal{B}$ is optimal for G.*
3. *For all $j = 1, \ldots, n$, $\frac{\mathcal{A}(\vec{\sigma}_j)}{\mathcal{B}(\vec{\sigma}_j)} = \Upsilon_{\mathcal{B}}$; i.e., $\mathcal{B}$ has the same performance relative to the adversary's on every input sequence.*
4. *If every component of x and y is strictly greater than 0, then x and y are the only optimal feasible solutions to the primal and dual problems of G, and, consequently, $\mathcal{B}$ is the only optimal randomized online algorithm.*

Proof.

Statement 1. By direct verification, $x \in X$ and $y \in Y$. Then, since $x^T u_n = (y^T u_n)^T = y^T u_n$, by Lemma 2.2(2) $x \in \bar{X}$ and $y \in \bar{Y}$.

Statement 2. Note that $x^T u_n = r^*$ by Statement 1 and Lemma 2.2(2). Then, by Statement 1 and Lemma 2.2(3), b is an optimal mixed strategy of the online player. Thus, this statement follows from Theorem 2.1.

Statement 3. As pointed out in Statement 2, $x^T u_n = r^*$. By direct evaluation and Statement 2 $b^T H = \frac{1}{r^*} u_n = \frac{1}{\Upsilon_{\mathcal{B}}} u_n$. Then this statement follows from the fact that by definition, the jth component of $b^T H$ equals $\frac{\mathcal{B}(\vec{\sigma}_j)}{\mathcal{A}(\vec{\sigma}_j)}$.

Statement 4. To prove the uniqueness of $\mathcal{B}$, by Theorem 2.1(2) and Lemma 2.2 (3), it suffices to show that $\bar{X}$ has a unique element. By basics of linear programming [22], $\bar{X}$ has only a finite number of extreme points, and any element in X is a finite convex combination of these extreme points. Thus, it suffices to show that x is the only extreme point of $\bar{X}$ as follows. Since H^{-1} exists, x and y are extreme points of $\bar{X}$ and $\bar{Y}$ by Fact 2.3 with $I = J = Z_n$. On the other hand, let z be any extreme point of $\bar{X}$. Since y is an extreme point of $\bar{Y}$, there is a square submatrix $H' = (h_{ij})_{i \in I, j \in J}$ of H such that z and y satisfy the five conditions in Fact 2.3. Since $y_j > 0$ for $j \in Z_n$, $J = Z_n$ by Condition 5. Since H' is square, $I = Z_n$ and $H' = H$. Then, by Condition 2, $z^T H = u_n^T$. Since $x^T H = u_n^T$, we have $z = x$ as desired. □

3. Optimal static algorithms. This section applies the general tools in section 2 to the static buy-and-hold trading problem to derive the smallest possible competitive ratio for static algorithms.

3.1. Notations. As specified in section 1, the investor in the buy-and-hold trading problem is given α, β, and n prior to an n-day investment horizon.

For $i \in Z_n$, let e_i be the given security's exchange rate on the ith day of the investment horizon. Let e_0 be the exchange rate on the 0th day, i.e., the day right before the investment horizon. Without loss of generality, we normalize e_0 to 1 to simplify the discussion. An *admissible* exchange rate sequence is any $\vec{e} = \langle e_1, e_2, \ldots, e_n \rangle$ where $e_i \in [e_{i-1}\beta^{-1}, e_{i-1}\alpha]$.

As in section 1, let E denote the set of all admissible exchange rate sequences. Let $\mathcal{A}$ denote the optimal offline trading algorithm. Let $\mathcal{B}$ be the investor's online trading algorithm. After the adversary examines $\mathcal{B}$ but before the investor starts executing $\mathcal{B}$, the adversary picks and fixes some $\vec{e} \in E$. On the ith day for $i \in Z_n$, upon seeing e_i, $\mathcal{B}$ decides the amount of remaining capital to be traded for shares of the security without knowing any future exchange rate, i.e., e_j with $j > i$. Note that $\mathcal{A}(\vec{e}) = \max_{1 \leq i \leq n} e_i$, and $\mathcal{B}(\vec{e}) = \sum_{i=1}^{n} a_i e_i$, where a_i is the (expected) amount

of dollars invested by $\mathcal{B}$ on the ith day and depends only on the current and past exchange rate $e_1, e_2, \ldots, e_i$.

For $i \in Z_n$, the algorithm $\mathcal{S}_i$ which trades the entire initial capital of one dollar on the ith day is called the *trade-once algorithm on the ith day.* Note that $\mathcal{S}_i$ is static and $\mathcal{S}_i(\vec{e}) = e_i$.

Let $\mathcal{S}$ be a randomized static algorithm. Let s_i be the expected amount of dollars invested by $\mathcal{S}$ on the ith day. Note that $s_i \geq 0$ for all i and $\sum_{i=1}^{n} s_i = 1$. Thus, let $\mathcal{S}'$ be the deterministic static algorithm that invests s_i on the ith day. Also, since the amounts $s_1, \ldots, s_n$ define a probability density function in $\Phi(Z_n)$, let $\mathcal{S}''$ be the randomized static algorithm that applies $\mathcal{S}_i$ with probability s_i.

LEMMA 3.1. *$\mathcal{S}$, $\mathcal{S}'$, and $\mathcal{S}''$ are equivalent in the sense that for all $\vec{e} \in E$, $\mathcal{S}(\vec{e}) = \mathcal{S}'(\vec{e}) = \mathcal{S}''(\vec{e})$.*

Proof. The proof is straightforward. □

By Lemma 3.1, we identify $\mathcal{S}, \mathcal{S}'$, and $\mathcal{S}''$. Also, let r_{s}^* be the smallest competitive ratio for the static algorithms; then by Lemma 3.1,

$$r_{\mathrm{s}}^* = \inf_{f \in \Phi(Z_n)} \sup_{\vec{e} \in E} \frac{\mathcal{A}(\vec{e})}{\sum_{i=1}^{n} f(i)\, \mathcal{S}_i(\vec{e})}. \tag{3.1}$$

3.2. Reduction to finite games. The static buy-and-hold trading problem is an infinite planning game because the adversary has an infinite number of pure strategies, while by Lemma 3.1 the online player has n pure strategies $\mathcal{S}_i$. In order to use the tools in section 2, we need to reduce the game to a finite one by eliminating the adversary's dominated pure strategies, i.e., non-worst-case exchange rate sequences, so that the remaining exchange rate sequences are finite in number.

For $j = 1, \ldots, n$, let

$$\vec{e}_j = \langle \overbrace{\alpha, \alpha^2, \ldots, \alpha^j}^{j}, \overbrace{\alpha^j \beta^{-1}, \alpha^j \beta^{-2}, \ldots, \alpha^j \beta^{j-n}}^{n-j} \rangle.$$

We call these n exchange rate sequences the *downturns*; see Figure 2 for an illustration.

LEMMA 3.2.

1. *Given a static algorithm $\mathcal{S}$, each $\vec{e} \in E$ is dominated by downturn $\vec{e}_j$, i.e., $\frac{\mathcal{A}(\vec{e})}{\mathcal{S}(\vec{e})} \leq \frac{\mathcal{A}(\vec{e}_j)}{\mathcal{S}(\vec{e}_j)}$, where $e_j = \max_{i=1}^{n} e_i$.*
2. *The smallest competitive ratio for the static algorithms is*

$$r_{\mathrm{s}}^* = \inf_{f \in \Phi(Z_n)} \max_{1 \leq j \leq n} \frac{\mathcal{A}(\vec{e}_j)}{\sum_{i=1}^{n} f(i)\, \mathcal{S}_i(\vec{e}_j)}.$$

3. *The static buy-and-hold trading problem can be regarded as a finite zero-sum two-person game $\Gamma_K(n, n)$ with the payoff matrix K defined by $K(i, j) = \alpha^{i-j}$ if $i \leq j$ or β^{j-i} if $i > j$, i.e.,*

$$K = \begin{pmatrix} 1 & \alpha^{-1} & \alpha^{-2} & \cdots & \alpha^{1-n} \\ \beta^{-1} & 1 & \alpha^{-1} & \cdots & \alpha^{2-n} \\ \beta^{-2} & \beta^{-1} & 1 & \cdots & \alpha^{3-n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta^{1-n} & \beta^{2-n} & \beta^{3-n} & \cdots & 1 \end{pmatrix}.$$

Proof. Statement 1 follows from the fact that $\frac{e_j}{e_i} \leq \alpha^{j-i}$ if $i \leq j$ and $\frac{e_j}{e_i} \leq \beta^{i-j}$ otherwise. Statement 2 follows from (3.1) and Statement 1. For Statement 3, we

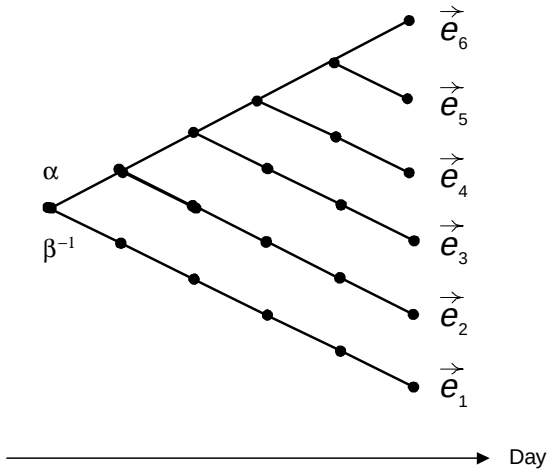


FIG. 2. *The downturns.*

let $\mathcal{S}_i$ be the online player's ith pure strategy and let $\vec{e}_j$ be the adversary's jth pure strategy. As in section 2, the payoff matrix K is defined by $K(i,j) = \frac{\mathcal{S}_i(\vec{e}_j)}{\mathcal{A}(\vec{e}_j)}$. The statement then follows from the facts that $\mathcal{A}(\vec{e}_j) = \alpha^j$ and that $\mathcal{S}_i(\vec{e}_j) = \alpha^i$ if $i \le j$ or $\alpha^j\beta^{j-i}$ otherwise. □

In light of Lemma 3.2, an optimal mixed strategy of the online player of $\Gamma_K(n,n)$ corresponds to an optimal static algorithm. Thus, we next solve $\Gamma_K(n,n)$ to derive an optimal static algorithm.

3.3. Deriving an optimal static algorithm.

LEMMA 3.3. *For $n \ge 2$, $\det(K) = (1-\alpha^{-1}\beta^{-1})^{n-1} > 0$.*

Proof. We use K_n to emphasize the dimension n of K. Let A_{ij} be the submatrix of K_n obtained by deleting row i and column j. To expand $\det(K_n)$ along the first row of K_n, observe that $A_{11} = K_{n-1}$. Furthermore, the first column of A_{12} equals β^{-1} times that of A_{11}, while the other columns of A_{12} equal the corresponding ones of A_{11}; thus $\det(A_{12}) = \beta^{-1}\det(A_{11})$. For $j = 3,\dots,n$, $\det(A_{1j}) = 0$ because in A_{1j}, the first column equals β^{-1} times the second column. Hence, $\det(K_n) = \det(A_{11}) - \alpha^{-1}\det(A_{12}) = \det(K_{n-1}) - \alpha^{-1}\beta^{-1}\det(K_{n-1}) = (1-\alpha^{-1}\beta^{-1})\det(K_{n-1})$. The lemma immediately follows by induction on n. □

Let b^* be the column vector of n components defined by

$$b_i^* = \begin{cases} \frac{\alpha(\beta-1)}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}, & i = 1; \\ \frac{(\alpha-1)(\beta-1)}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}, & 1 < i < n; \\ \frac{(\alpha-1)\beta}{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}, & i = n. \end{cases} \tag{3.2}$$

Since $b^* > 0$ and ${b^*}^T u_n = 1$, b^* represents a mixed strategy of the online player. Therefore let BAL denote the static algorithm which applies $\mathcal{S}_i$ with probability b_i^*; note that by Lemma 3.1, BAL is equivalent to the deterministic static algorithm which invests b_i^* dollars on the ith day. Let c^* be the vector obtained by swapping the first and nth components of b^*. Similarly, $c^* > 0$ and ${c^*}^T u_n = 1$, and we intend c^* to represent a mixed strategy of the adversary.

The next theorem analyzes BAL. In light of Statement 3 of the theorem, we call BAL the *balanced strategy.*

THEOREM 3.4. *Let* $r = \frac{n\alpha\beta-(n-1)(\alpha+\beta)+(n-2)}{\alpha\beta-1}$.

1. BAL *is an optimal static algorithm, and* $\Upsilon_{\mathrm{BAL}} = r_{\mathrm{s}}^* = r$.
2. BAL *is the only optimal static algorithm subject to the equivalence stated in Lemma* 3.1.
3. *For* $j = 1, \ldots, n$, $\frac{\mathcal{A}(\vec{e}_j)}{\mathrm{BAL}(\vec{e}_j)} = \Upsilon_{\mathrm{BAL}}$*; in other words,* BAL *has the same performance relative to the adversary's on every downturn.*

Proof. Let $\bar{b} = rb^*$, and $\bar{c} = rc^*$. By Lemma 3.3, K^{-1} exists. Below we prove $\bar{b}^T = u_n^T K^{-1}$ and $\bar{c} = K^{-1}u_n$. Then, Statement 1 follows from Theorem 2.4(2) and the fact that $\bar{b} \geq 0$, $\bar{c} \geq 0$, and $\bar{b}^T u_n = r$. Statement 2 follows from Theorem 2.4(4) and the fact that every component of $\bar{b}$ and $\bar{c}$ is greater than 0. Statement 3 follows from Theorem 2.4(3)

To prove $\bar{b}^T = u_n^T K^{-1}$ and $\bar{c} = K^{-1}u_n$, observe that $\bar{c}$ can be obtained by swapping α and β in $\bar{b}$, and the jth column K^j of K can be obtained from the jth row of K by the same operation. Therefore, $\bar{b}^T K = u_n^T$ if and only if $K\bar{c} = u_n$, and we need only to establish $\bar{b}^T K = u_n^T$. Since $\bar{b}^T K^1 = 1$ if and only if $\bar{b}^T K^n = 1$, we show only $\bar{b}^T K^j = 1$ for $1 \leq j < n$ as follows:

$$\begin{aligned}
\bar{b}^T K^j &= \sum_{i=1}^{n} \bar{b}_i K(i,j) \\
&= \frac{1}{\alpha\beta-1}\left[\alpha(\beta-1)\alpha^{1-j} + \sum_{1<i\leq j}(\alpha-1)(\beta-1)\alpha^{i-j}\right. \\
&\qquad \left. + \sum_{j<i<n}(\alpha-1)(\beta-1)\beta^{j-i} + (\alpha-1)\beta\beta^{j-n}\right] \\
&= \frac{1}{\alpha\beta-1}\left[\alpha^{2-j}(\beta-1) + (\alpha-\alpha^{2-j})(\beta-1)\right. \\
&\qquad \left. +(\alpha-1)(1-\beta^{j-n+1}) + (\alpha-1)\beta^{j-n+1}\right] \\
&= \frac{1}{\alpha\beta-1}(\alpha\beta-1) \\
&= 1. \qquad \square
\end{aligned}$$

3.4. Comparison with the dollar averaging strategy. The *dollar averaging strategy* (DA) is the static algorithm which invests an equal amount of capital, i.e., $1/n$ dollars, on each trading day. Thus, by Lemma 3.1, DA is the uniformly mixed strategy for the online player in the game $\Gamma_K(n,n)$. By Theorem 3.4, DA is not an optimal static algorithm, and $\Upsilon_{\mathrm{BAL}} < \Upsilon_{\mathrm{DA}}$. The next lemma gives a closed-form formula of Υ_{DA}. Figure 3 plots the relationship between Υ_{DA} and Υ_{BAL} for $2 \leq n \leq 100$.

LEMMA 3.5. $\Upsilon_{\mathrm{DA}} = \max\{\frac{n(1-\alpha^{-1})}{1-\alpha^{-n}}, \frac{n(1-\beta^{-1})}{1-\beta^{-n}}\}$.

Proof. Let $B_j = \sum_{i=1}^{n} K(i,j)$. By Lemma 3.2, $\Upsilon_{\mathrm{DA}} = \max_{1\leq j\leq n} \frac{n}{B_j}$. By algebra, $B_{j+1} - B_j$ is a decreasing function of j. Thus, B_j is a function of j whose minimum occurs at one end of the domain $\{1, \ldots, n\}$. The lemma follows from this concavity. $\square$

We have also experimented with BAL and DA using Taiwan's market data. As shown in Figure 1, the Taipei Stock Exchange (TSE) adopts $\alpha = 1/0.93$ and $\beta = 1.07$. We select the Taiwan Semiconductor Manufacturing Company (TSMC) and Acer Computer Company (Acer) for experimental analysis. TSMC is the largest foundry of wafer manufacturing in the world and is listed on both TSE and the New York

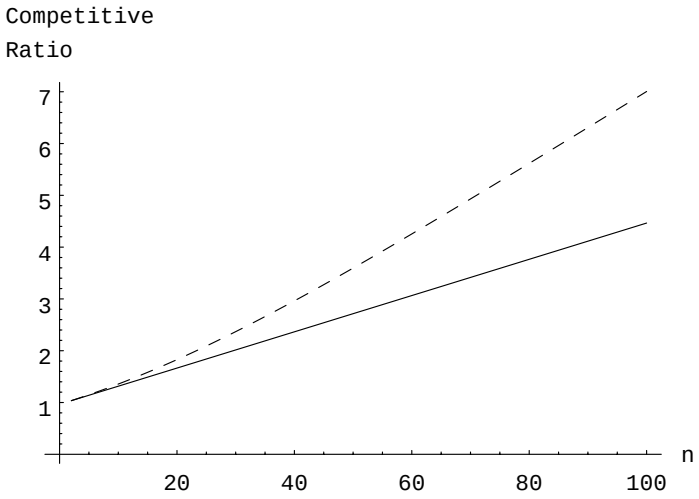


FIG. 3. *The dashed and solid dotted lines denote* Υ_{DA} *and* Υ_{BAL}*, respectively, with* $\alpha = 1/0.93$ *and* $\beta = 1.07$.

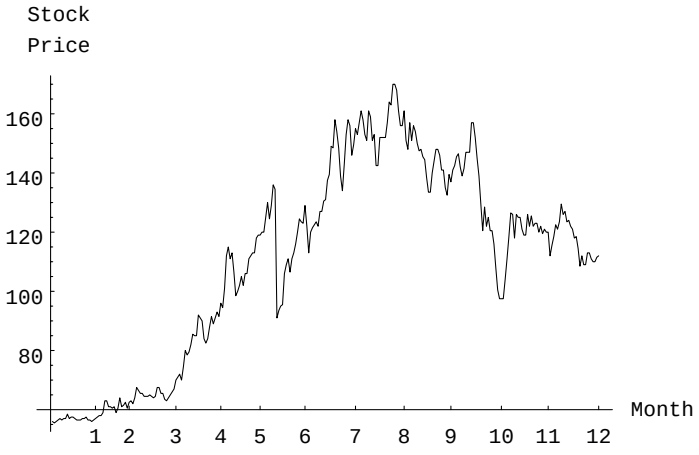


FIG. 4. *TSMC's daily closing stock prices in* 1997.

Stock Exchange (NYSE) under the symbol TSM. Acer is the world's third largest PC manufacturer as well as the fifth largest mobile PC manufacturer.

Figure 4 shows the daily closing prices of TSMC in 1997. All stock prices are quoted in the New Taiwan dollar (NT dollar). One investment plan is executed each month. Each plan buys shares of TSMC with an initial capital of one NT dollar as in section 3; however, the exchange rate of a day is the reciprocal of that day's share price without an initial normalization to one. A *monthly* accumulation is the

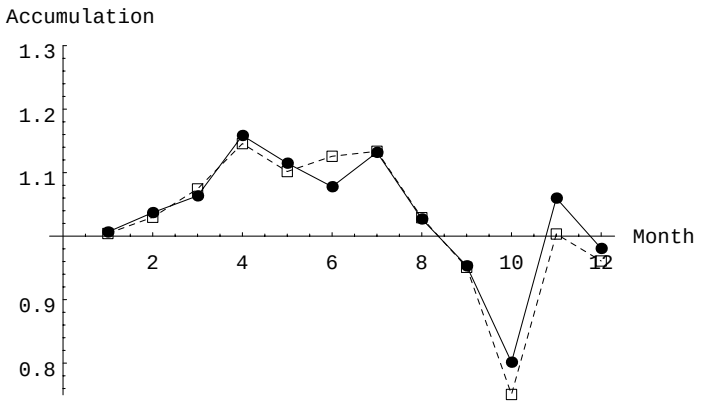


FIG. 5. *Accumulations of* BAL *and* DA *on TSMC.*

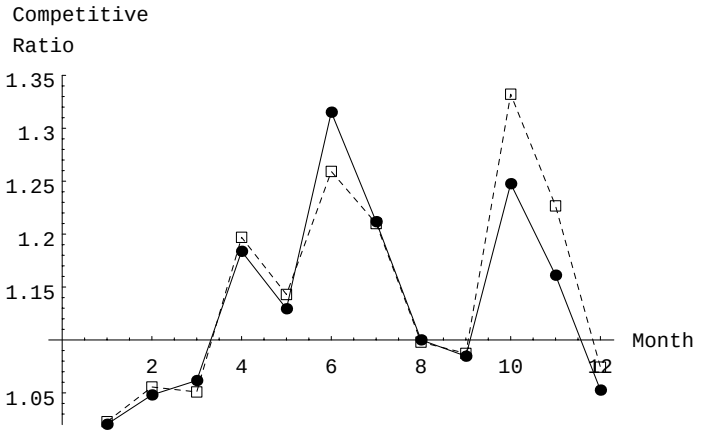


FIG. 6. *Realized competitive ratios of* BAL *and* DA *on TSMC.*

total number of shares acquired over a month. For ease of comparison, a monthly accumulation is expressed in NT dollar by converting the acquired shares into NT dollars at the price of the last trading day of each month. Figure 5 shows the monthly accumulations of BAL and DA on TSMC for each month of 1997. Notice that BAL and DA are money-making except in September, October, and December. Figure 6 shows the *realized* competitive ratios of BAL and DA, which are the performance ratios as defined in (1.1) but with $\vec{e}$ set to the actual exchange rate sequences. Note that for all 12 months, these ratios are less than 1.35. For visual clarity, we join the monthly accumulations and competitive ratios by line segments and use the solid and dotted lines to denote the graphs of BAL and DA, respectively. Observe that, overall, BAL outperforms DA.

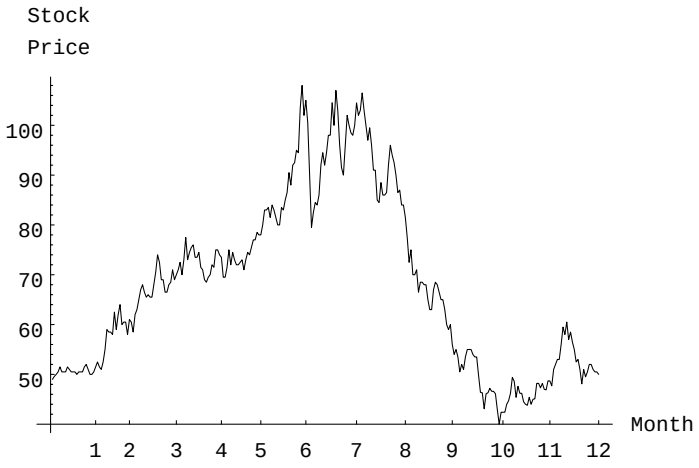


FIG. 7. *Acer's daily closing stock prices in* 1997.

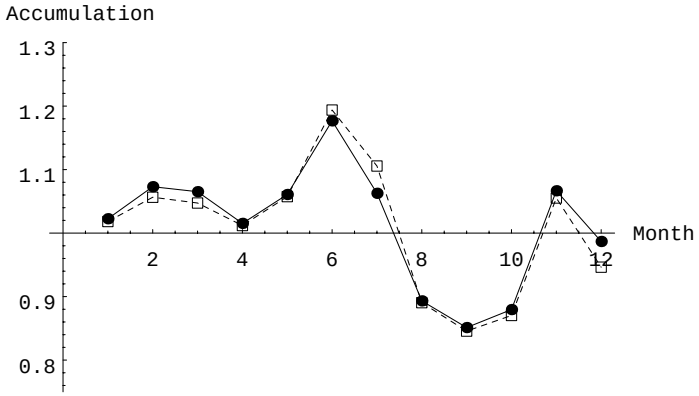


FIG. 8. *Accumulations of* BAL *and* DA *on Acer.*

Figure 7 shows the daily closing prices of Acer in 1997. Figures 8 and 9 show the monthly accumulations and realized competitive ratios of BAL and DA, respectively. The experimental results for Acer lead to similar conclusions to those for TSMC.

4. Open problems. We have presented the balanced strategy BAL and proved its unique optimality among the static algorithms. Furthermore, each of its exact competitive ratio and daily investment amounts has a closed-form expression which takes $O(1)$ time to evaluate. In light of these results, an immediate open problem is whether there are similar results for dynamic online trading algorithms. There are two orthogonal directions for further research as follows.

One direction is to change the assumption that the time horizon is fixed and

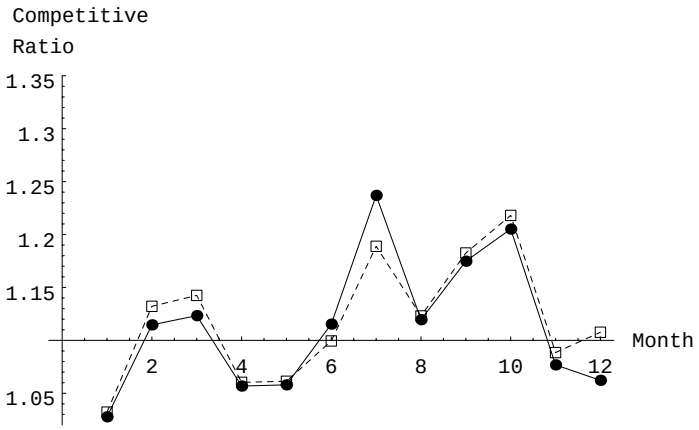


FIG. 9. *Realized competitive ratios of* BAL *and* DA *on Acer.*

known a priori to $\mathcal{B}$. For instance, it would be meaningful to consider the scenario that there is a cash stream instead of a one-time capital at the beginning of the investment horizon. For this scenario, an investor might need to guess when the cash stream will end.

The other direction is to replace α and β with a known probability distribution of the ratio $\frac{e'}{e}$. This would be an example of the standard approach in finance of considering the average-case performance under an assumed probabilistic model. While the worst-case approach in computer science is unnecessarily pessimistic, the average-case approach in finance is overly dependent on the chosen model. In general, it would be of interest to combine these two approaches to formulate more informative computational problems than either approach could.

Acknowledgment. We wish to thank the anonymous referees for very thoughtful comments. Some of the comments have resulted in open problems in section 4.

REFERENCES

[1] M. AJTAI, N. MEGIDDO, AND O. WAARTS, *Improved algorithms and analysis for secretary problems and generalizations*, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 473–482.

[2] S. AL-BINALI, *The competitive analysis of risk taking with application to online trading*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 336–344.

[3] M. BAXTER AND A. RENNIE, *Financial Calculus: An Introduction to Derivative Pricing*, Cambridge University Press, Cambridge, UK, 1996.

[4] A. BORODIN AND R. EL-YANIV, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[5] A. CHOU, J. COOPERSTOCK, R. EL-YANIV, M. KLUGERMAN, AND T. LEIGHTON, *The statistical adversary allows optimal money-making trading strategies*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, pp. 467–476.

[6] Y. S. CHOW, S. MORIGUTI, H. ROBBINS, AND S. M. SAMUELS, *Optimal selection based on relative rank* (*the secretary problem*), Israel J. Math., 2 (1964), pp. 81–90.

[7] T. Cover and E. Ordentlich, *Universal portfolios with side information*, IEEE Trans. Inform. Theory, 42 (1996), pp. 348–363.
[8] T. M. Cover, *Universal portfolio*, Math. Finance, 1 (1991), pp. 1–29.
[9] D. Duffie, *Dynamic Asset Pricing Theory*, Princeton University Press, Princeton, NJ, 1996.
[10] R. El-Yaniv, *Competitive solutions for online financial problems*, ACM Comput. Surveys, 30 (1998), pp. 28–69.
[11] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin, *Competitive analysis of financial games*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 327–333.
[12] R. El-Yaniv, A. Fiat, R. M. Karp, and G. Turpin, *Optimal search and one-way trading online algorithms*, Algorithmica, 30 (2001), pp. 101–139.
[13] P. Freeman, *The secretary problem and its extensions*, Internat. Statist. Rev., 51 (1983), pp. 189–206.
[14] J. Hull, *Options, Futures, and Other Derivatives*, 3rd ed., Prentice-Hall, Upper Saddle River, NJ, 1997.
[15] M. Y. Kao and S. R. Tate, *Online matching with blocked input*, Inform. Process. Lett., 38 (1991), pp. 113–116.
[16] M. Y. Kao and S. R. Tate, *On-line difference maximization*, in Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 1997, pp. 175–182.
[17] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, *An optimal algorithm for on-line bipartite matching*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 352–358.
[18] Y. D. Lyuu, *Financial Engineering and Computation: Principles, Mathematics, Algorithms*, Cambridge University Press, Cambridge, UK, 2001.
[19] K. R. MacCrimmon and D. A. Wehrung, *Taking Risks: The Management of Uncertainty*, Free Press, New York, 1986.
[20] S. N. Neftci, *An Introduction to the Mathematics of Financial Derivatives*, Academic Press, New York, 1996.
[21] E. Ordentlich and T. Cover, *On-line portfolio selection*, in Proceedings of the Ninth Conference on Computational Learning Theory, Desenzano del Garda, Italy, 1996, pp. 310–313.
[22] L. A. Petrosjan and N. A. Zenkevich, *Game Theory*, World Scientific, Singapore, 1996.
[23] T. Raghavan, *Zero-sum two-person games*, in Handbook of Game Theory, Vol. 2, R. Aumann and S. Hart, eds., North-Holland, Amsterdam, 1994, pp. 735–768.
[24] W. F. Sharpe, G. J. Alexander, and J. V. Bailey, *Investments*, 5th ed., Prentice-Hall, Upper Saddle River, NJ, 1995.
[25] D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.
[26] P. Wilmott, S. Howison, and J. Dewynne, *The Mathematics of Financial Derivatives*, Cambridge University Press, Cambridge, UK, 1995.
[27] A. C. C. Yao, *New algorithms for bin packing*, J. ACM, 27 (1980), pp. 207–227.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 460–476

QUANTUM FORMULAS: A LOWER BOUND AND SIMULATION*

VWANI P. ROYCHOWDHURY† AND FARROKH VATAN†‡

Abstract. We show that Nechiporuk's method [I. Wegener, *The Complexity of Boolean Functions*, Teubner-Wiley, New York, 1987] for proving lower bounds for Boolean formulas can be extended to the quantum case. This leads to an $\Omega(n^2/\log^2 n)$ lower bound for quantum formulas computing an explicit function. The only known previous explicit lower bound for quantum formulas [A. Yao, *Proceedings of 34th IEEE Symposium on Foundations of Computer Science,* IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 352–361] states that the majority function does not have a linear-size quantum formula. We also show that quantum formulas can be simulated by Boolean circuits of almost the same size.

Key words. quantum formula, lower bound, mixed state, density matrix

AMS subject classifications. 81P68, 68Q10, 68Q05, 03D10

PII. S0097539700370965

1. Introduction.

Computational devices based on quantum physics have attracted much attention lately, and quantum algorithms that perform much faster than their classical counterparts have been developed [12, 21, 22]. To provide a systematic study of the computational power of quantum devices, models similar to those for classical computational devices have been proposed. Deutsch [9] formulated the notion of quantum Turing machine. This approach was further developed by Bernstein and Vazirani [5], and the concept of an efficient universal quantum Turing machine was introduced. As in the case of classical Boolean computation, there is also a quantum model of computation based on circuits (or networks). Yao [27] proved that the quantum circuit model, first introduced by Deutsch [10], is equivalent to the quantum Turing machine model.

Since every Boolean circuit can be simulated by a quantum circuit, with at most a polynomial factor increase in its size, any nontrivial lower bound for quantum circuits could have far-reaching consequences. In classical Boolean circuit theory, all nontrivial lower bounds are for proper subclasses of Boolean circuits such as monotone circuits, formulas, bounded-depth circuits, etc. In the quantum case it also seems that the only hope to prove nontrivial lower bounds is for proper subclasses of quantum circuits. So far the only such known lower bound has been derived by Yao [27] for quantum formulas.[1] The quantum formula is a straightforward generalization of the classical Boolean formula: in both cases, the graph of the circuit is a tree. Yao has proved that the quantum formula size of the majority function MAJ_n is not linear;[2] i.e., if $L(\mathrm{MAJ}_n)$ denotes the minimum quantum formula size of MAJ_n, then

*Received by the editors April 26, 2000; accepted for publication (in revised form) April 10, 2001; published electronically August 22, 2001. This work was supported in part by grants from the Revolutionary Computing group at JPL (contract 961360) and from the DARPA Ultra program (subcontract from Purdue University 530-1415-01).

http://www.siam.org/journals/sicomp/31-2/37096.html

†Electrical Engineering Department, UCLA, Los Angeles, CA 90095 (vwani@ee.ucla.edu, vatan@ee.ucla.edu).

‡Current address: Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109 (Farrokh.Vatan@jpl.nasa.gov).

[1] There are exponential lower bounds on the time of quantum computation for the black-box model (see, e.g., [3]), but they do not apply to the size of quantum circuits.

[2] The value of $\mathrm{MAJ}_n(x_1, \ldots, x_n)$ is 1 if at least $\lceil n/2 \rceil$ of inputs are 1.

$\lim_{n\longrightarrow\infty} L(\mathrm{MAJ}_n)/n = \infty$. This bound is derived from a bound on the quantum communication complexity of Boolean functions.

In this paper, we prove an almost quadratic lower bound for quantum formula size. The key step in the derivation of this lower bound is the extension of Nechiporuk's method to quantum formulas; for a detailed discussion of Nechiporuk's method in the Boolean setting, see [11, 26]. Nechiporuk's method has been used in several different areas of Boolean complexity (see, e.g., [11] for details). It has also been applied to models where the gates do not take on binary or discrete values, but the input/output map still corresponds to a Boolean function. For example, in [23] this method has been used to get a lower bound for arithmetic and threshold formulas. The challenging part of this method is a step that we shall refer to as "path squeezing" (see section 4 for the exact meaning of it). Although in the case of Boolean gates this part can be solved easily, in the case of analog circuits it is far from obvious (see [23]). For the quantum formulas "path squeezing" becomes even more complicated, because here we should take care of any *quantum entanglement* and interference phenomena. We show that it is still possible to squeeze a path with an arbitrary number of constant inputs to a path with a fixed number of inputs. This leads to a lower bound of $\Omega(n^2/\log^2 n)$ on the size of quantum formulas computing a class of explicit functions. For example, we get such a bound for the element distinctness function ED_n. The input of ED_n, for $n = 2\ell\log\ell$, is of the form $(z_1,\ldots,z_\ell)$, where each z_j is a string of $2\log\ell$ bits. Then $\mathrm{ED}_n(z_1,\ldots,z_\ell) = 1$ if and only if all these strings are pairwise distinct.

At the end of the paper we compare the powers of quantum formulas and Boolean *circuits*. Surprisingly, in some sense quantum formulas are not more powerful than Boolean circuits. Any quantum formula of size s and depth d can be approximated by a Boolean circuit of size $O(s\log s\log\log s)$ and depth $O(d\log\log s)$. Similar results are not known, and most probably are not true, for quantum circuits and other models which are depending on real number parameters (like arithmetic circuits [23]). The key idea for this simulation is that the computation of a quantum formula on an input (which is a pure state in the Hilbert space) can be described as performing a sequence of unitary operations on 4×4 density matrices of mixed states.

In this paper we use the notation $|\cdot|$ for two different purposes. When α is a complex number, $|\alpha|$ denotes the absolute value of α; i.e., $|\alpha| = \sqrt{\alpha\cdot\alpha^*}$. While if X is a set, then $|X|$ denotes the cardinality of X.

2. Preliminaries. A *quantum circuit* is defined as a straightforward generalization of an acyclic classical (Boolean) circuit (see [10]). For constructing a quantum circuit, we begin with a *basis* of quantum gates as elementary gates. Each elementary gate g with d inputs represents a unitary operation $U_g \in \mathbf{U}(2^d)$, where $\mathbf{U}(m)$ denotes the group of $m\times m$ unitary complex matrices. The gates are interconnected by quantum "wires." Each wire represents a quantum bit, *qubit*, which is a 2-state quantum system represented by a unit vector in $\mathbb{C}^2$. Let $\{|0\rangle, |1\rangle\}$ be the standard orthonormal basis of $\mathbb{C}^2$. The $|0\rangle$ and $|1\rangle$ values of a qubit correspond to the classical Boolean 0 and 1 values, but a qubit can also be in a superposition of the form $\alpha|0\rangle + \beta|1\rangle$, where $\alpha,\beta\in\mathbb{C}$ and $|\alpha|^2+|\beta|^2 = 1$. Note that the output of such a gate, in general, is not a tensor product of its inputs but an *entangled state,* e.g., a state like $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ which cannot be written as a tensor product.

If the circuit has m inputs, then for each d-input gate g, the unitary operation $U_g \in \mathbf{U}(2^d)$ can be considered in a natural way as an operator in $\mathbf{U}(2^m)$ by acting as the identity operator on the other $(m-d)$ qubits. Hence, a quantum circuit with m inputs computes a unitary operator in $\mathbf{U}(2^m)$, which is the product of successive

unitary operators defined by successive gates.

The *size* of a quantum circuit C, denoted by $\mathsf{size}(C)$, is the number of gates occurring in C. The *depth* of C, denoted by $\mathsf{depth}(C)$, is the length of the longest path in C from an input to an output gate.

In this paper, we consider quantum circuits that compute Boolean functions. Consider a quantum circuit C with m inputs. Suppose that C computes the unitary operator $U_C \in \mathbf{U}(2^m)$. We say C computes the Boolean function $f\colon \{0,1\}^n \longrightarrow \{0,1\}$ if the following holds. The inputs are labeled by the variables $x_1, x_2, \ldots, x_n$ or the constants $|0\rangle$ or $|1\rangle$. (Different inputs may be labeled by the same variable x_j.) We consider one of the output wires, say, the first one, as the output of the circuit. To compute the value of the circuit at $(a_1, \ldots, a_n) \in \{0,1\}^n$, let the value of each input wire with label x_j be $|a_j\rangle$. These inputs, along with the constant inputs to the circuit, define a unit vector $|\alpha\rangle$ in $\mathbb{C}^{2^m}$. In fact, this vector is a standard basis vector of the following form (up to some repetitions and a permutation):

$$|\alpha\rangle = |a_1\rangle \otimes \cdots \otimes |a_n\rangle \otimes |0\rangle \otimes \cdots \otimes |1\rangle\,.$$

The action of the circuit C on the input $|\alpha\rangle$ is the same as $U_C(|\alpha\rangle)$. Note that since U_C is unitary, $\|U_C(|\alpha\rangle)\| = 1$. We decompose the vector $U_C(|\alpha\rangle) \in \mathbb{C}^{2^m}$ with respect to the output qubit. Let the result be

$$U_C(|\alpha\rangle) = |0\rangle \otimes |A_{0,\alpha}\rangle + |1\rangle \otimes |A_{1,\alpha}\rangle\,.$$

Then we define the *probability* that C outputs 1 (on the input α) as $p_\alpha = \||A_{1,\alpha}\rangle\|^2$, i.e., the square of the length of $|A_{1,\alpha}\rangle \in \mathbb{C}^{2^{m-1}}$. Finally, we say that the quantum circuit C computes the Boolean function f if for every $\alpha \in \{0,1\}^n$, if $f(\alpha) = 1$, then $p_\alpha > 2/3$; and if $f(\alpha) = 0$, then $p_\alpha < 1/3$.

Following Yao [27], we define quantum formulas as a subclass of quantum circuits. A quantum circuit C is a *formula* if for every input there is a unique path that connects it to the output qubit. To make this definition more clear we define the *computation graph* of C, denoted by G_C. The nodes of G_C correspond to a subset of the gates of C. We start with the output gate of C, i.e., the gate which provides the output qubit, and let it be a node of G_C. Once a node v belongs to G_C, then all gates in C that provide inputs to v are considered as adjacent nodes of v in G_C. Then C is a formula if the graph G_C is a tree. Figure 2.1 provides examples of quantum circuits of both kinds, i.e., circuits that are also quantum formulas and circuits that are not formulas.

All circuits that we consider are over some fixed quantum basis. The lower bound does not depend on the basis; the only condition is that the number of inputs (and therefore the number of outputs) of each gate be bounded by some fixed constant number. (This condition is usually considered as part of the definition of a quantum basis.) For example, this basis can be the set of all 2-input 2-output quantum gates, and, as is shown in [2], this basis is universal for computation with quantum circuits.

It is well known that any Boolean circuit can be efficiently simulated by a quantum circuit over a universal basis. Indeed, for this purpose, the 3-bit *Toffoli gate* is enough (see, e.g., [4, 17]). Similarly, any Boolean formula can be efficiently simulated by a quantum formula using only a Toffoli gate or a basis universal for classical computation. In the special case, from [24] it follows that there is a polynomial-size log-depth quantum *formula* computing the majority function MAJ_n. This fact implies that for quantum formulas over reasonable bases (i.e., universal for classical computation) the threshold probability of the correct answer ($\frac{2}{3}$ in the above definition) can be efficiently boosted to a number arbitrarily close to one.

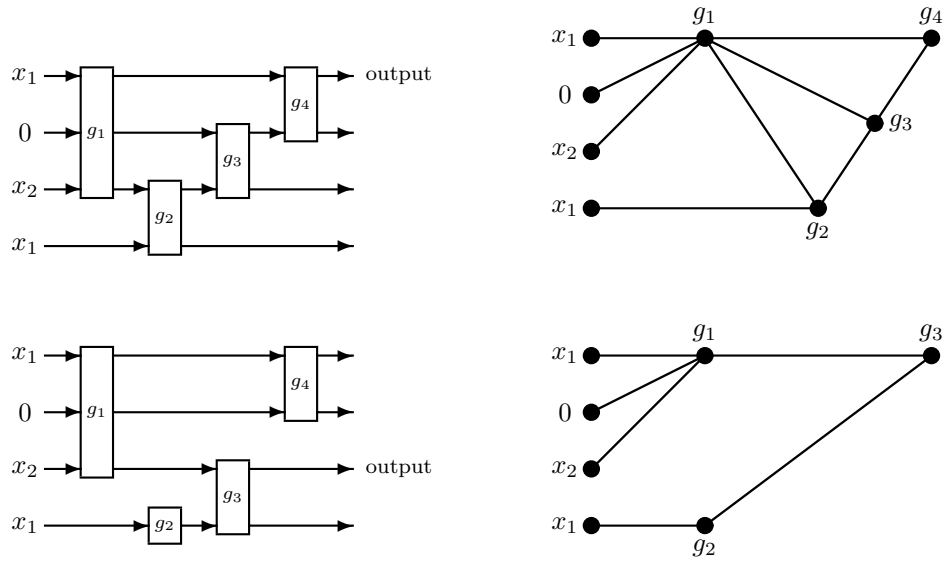


FIG. 2.1. *Quantum circuits and their computation graphs; the top circuit is not a formula, while the bottom one is a formula.*

For our proof we also need a Shannon-type result for quantum circuits. Knill [15] has proved several theorems about the quantum circuit complexity of almost all Boolean functions. We will use the following theorem.

THEOREM 2.1 (see [15]). *The number of different n-variable Boolean functions that can be computed by size N quantum circuits ($n \leq N$) with d-input d-output elementary gates is at most $2^{cN \log N}$, where c depends only on d.*

For the sake of completeness, in the appendix we have provided a proof for a slightly weaker bound. Our approach is different from that in [15], and it seems that it is shorter and simpler than the proof in [15]. Although the bound that we get is a little weaker than the bound provided by the above theorem (it is of the form $2^{O(nN)}$), our bound results in the same bound of Theorem 2.1 if $\log(N) = \Omega(n)$, which is true for almost all Boolean functions. Thus our result provides the same bound for the complexity of almost all functions and is sufficient for the bound we get in this paper.

We also need to consider general orthonormal bases in the space $\mathbb{C}^{2^n}$ other than the standard basis. In the context of quantum physics, we identify the Hilbert space $\mathbb{C}^{2^n}$ as the tensor product space $\bigotimes_{j=1}^{n} \mathbb{C}^2$, and the standard basis consists of the vectors

$$|c_1\rangle \otimes \cdots \otimes |c_n\rangle = |c_1 \cdots c_n\rangle\,,\ c_j \in \{0,1\}.$$

FACT 2.2. *Let $|A_j\rangle \in \mathbb{C}^{2^k}$ and $|B_\ell\rangle \in \mathbb{C}^{2^m}$ be unit vectors (for j and ℓ in some index sets). If $|A_j\rangle$ are pairwise orthogonal and $|B_\ell\rangle$ are pairwise orthogonal, then the family*

$$\left\{ |A_j\rangle \otimes |B_\ell\rangle \in \mathbb{C}^{2^{k+m}} : j, \ell \right\}$$

is an orthonormal set.

The following lemma, although seemingly obvious, is crucial for the "path squeezing" technique in the proof of the lower bound.

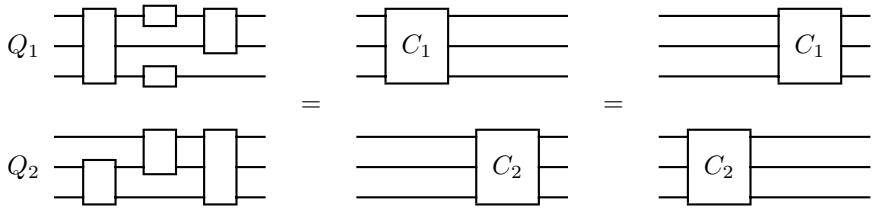


FIG. 2.2. *Decomposition of a quantum subcircuit acting on disjoint sets of qubits* (*Lemma* 2.3(a)).

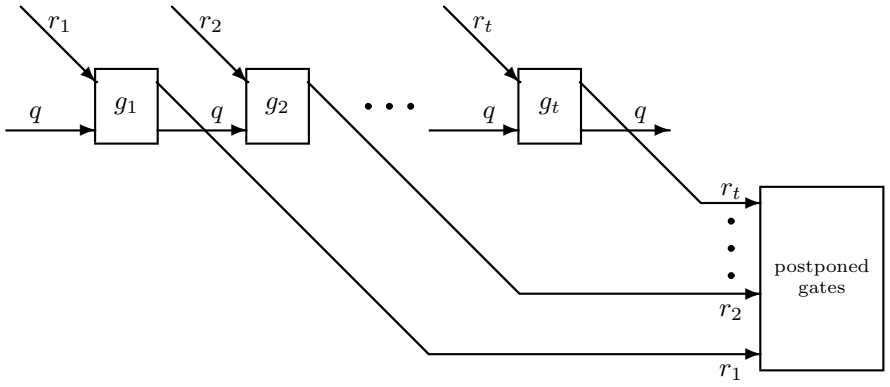


FIG. 2.3. *Postponing the gates* (*Lemma* 2.3(b)).

LEMMA 2.3. (a) *Suppose that C is a subcircuit of a quantum circuit. Let the inputs of C be divided into two disjoint sets of qubits Q_1 and Q_2. Suppose that each gate of C either acts only on qubits from Q_1 or only on qubits from Q_2. Then there are subcircuits C_1 and C_2 such that C_j acts only on qubits from Q_j, and the operation of C is the composition of operations of C_1 and C_2 no matter in which order they act; i.e., $C = C_1 \circ C_2 = C_2 \circ C_1$. Therefore the subcircuit C can be substituted by C_1 and C_2 (see Figure* 2.2*).*

(b) *Let C be a subcircuit of a quantum circuit with distinct input qubits q and $r_1, \ldots, r_t$. Suppose that only t gates $g_1, \ldots, g_t$ in C act on q. Moreover, suppose that each g_j acts on q and r_j. Then, without loss of generality (w.l.o.g.), we can assume that each qubit r_j after entering the gate g_j will not interact with any other qubit until the gate g_t is performed (see Figure* 2.3*).*

Proof. Part (a) is based on the following simple observation. If $M \in \mathbf{U}(2^m)$ and $N \in \mathbf{U}(2^n)$, then

$$\begin{aligned} M \otimes N &= (M \otimes I_n) \circ (I_m \otimes N) \\ &= (I_m \otimes N) \circ (M \otimes I_n), \end{aligned}$$

where I_t is the identity map in $\mathbf{U}(2^t)$. Note that the inputs of the subcircuit C may be in an entangled state; but to see that the equality $C = C_1 \circ C_2 = C_2 \circ C_1$ holds, it is enough to check this equality for the standard basis and extend it to the whole space by linearity.

Part (b) follows simply from part (a); as in Figure 2.4, part (a) can be applied on the subcircuit consisting of gates h_2 and h_3. Note that in this case input qubits r_j

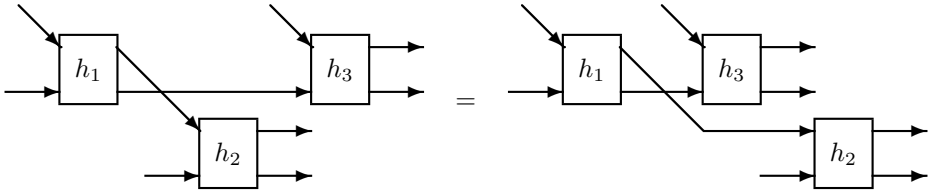


FIG. 2.4. *Changing the order of gates* (*Lemma* 2.3(b)).

of g_j's may also be in an entangled state. Again a linearity argument shows that we have to consider only the case in which r_j's are in a product state. □

The above lemma is a special case of a more general fact that operations on one part of a bipartite quantum system do not affect the result of operations on the other part (for more details see, e.g., [18]).

3. A new equivalent definition for quantum formulas. Kitaev [14] has brought to our attention that quantum formulas are equivalent to a model that is very similar to the classical formulas. In this model the inputs and the intermediate results are density matrices. Each gate is a completely positive trace-preserving super-operator, which maps density matrices of d-qubit systems to one-qubit density matrices. The underlying graph, like a classical formula, is a directed tree; i.e., from each input there is a unique path to the output gate. Thus the output of such a circuit is a density matrix of a single qubit which provides the probability of the output "0" or "1." To make the paper self-contained, we first present the definitions of the notions mentioned in this new definition.

By a *pure state* $|\alpha\rangle$ we mean a unit vector in some Hilbert space $\mathbb{C}^{2^n}$. A *mixed state* $\{\psi\}$ in $\mathbb{C}^{2^n}$ is a probability distribution on pure states in this Hilbert space. We denote such a mixed state as $\{\psi\} = \{p_k, |\psi_k\rangle\}$, where $p_k \geq 0$ and $\sum_k p_k = 1$. Then $\{\psi\}$ picks the pure state $|\psi_k\rangle$ with probability p_k.

The *density matrix* of a pure state $|\alpha\rangle$ is the matrix $\rho_{|\alpha\rangle}$ of the linear mapping $|\alpha\rangle\langle\alpha|$; i.e, the mapping $|x\rangle \longrightarrow \langle\alpha|x\rangle\,|\alpha\rangle$. Therefore, if $|0\rangle, |1\rangle, \ldots, |2^n-1\rangle$ represent the standard computational basis of $\mathbb{C}^{2^n}$ and $|\alpha\rangle = \sum_k \lambda_k |k\rangle$, then the (i,j) entry of $\rho_{|\alpha\rangle}$ is $\lambda_i \lambda_j^\star$. The importance of the density matrix is that it suffices to characterize the quantum state of the system. Specially, this matrix is enough to find the probabilities of measurements. In general, the result of each measurement can be represented by action of a projection operator $\mathcal{P}$ on the given state $|\alpha\rangle$, where $\mathcal{P}$ is a projection onto some subspace $\mathcal{E}$. Then the probability that the result of the measurement is in the subspace $\mathcal{E}$ is equal to $\mathrm{Tr}(\mathcal{P}\,\rho_{|\alpha\rangle})$.

The density matrix of a mixed state $\{\psi\} = \{p_k, |\psi_k\rangle\}$ is defined as

$$\rho_{\{\psi\}} = \sum_k p_k \rho_{|\psi_k\rangle} = \sum_k p_k\, |\psi_k\rangle\langle\psi_k|\,.$$

Like the case of pure states, the probability that the result of the measurement is in the subspace $\mathcal{E}$ is equal to $\mathrm{Tr}(\mathcal{P}\,\rho_{\{\psi\}})$.

If the (pure or mixed) state $|\psi\rangle$ can be written as the tensor product $|\phi\rangle \otimes |\chi\rangle$, then the density matrix $\rho_{|\psi\rangle}$ is equal to the tensor (Hadamard) product $\rho_{|\phi\rangle} \otimes \rho_{|\chi\rangle}$.

The next important notion is *partial trace*. Consider the Hilbert spaces $\mathcal{H}_1 = \mathbb{C}^{2^n}$ and $\mathcal{H}_1 = \mathbb{C}^{2^m}$, and $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$; thus $\mathcal{H}$ is isomorphic with $\mathbb{C}^{2^{n+m}}$. Let

$$\mathcal{B}_1 = \{\, |u_i\rangle : i = 1, \ldots, 2^n \,\} \quad \text{and} \quad \mathcal{B}_2 = \{\, |v_j\rangle : j = 1, \ldots, 2^m \,\}$$

be orthonormal bases for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. Then

$$\mathcal{B}_1 \otimes \mathcal{B}_2 = \{\, |u_i\rangle \otimes |v_j\rangle : i = 1, \ldots, 2^n,\ j = 1, \ldots, 2^m \,\}$$

is a basis for $\mathcal{H}$. Let ρ be the density matrix of a mixed state $|\psi\rangle$ in the space $\mathcal{H}$. It is possible to restrict the state $|\psi\rangle$ to the subspace $\mathcal{H}_1$. The result is a *partial trace* $\rho|_{\mathcal{H}_1} = \mathrm{Tr}_{\mathcal{H}_2}\, \rho$, which is the density matrix of some mixed state in the subspace $\mathcal{H}_1$. We also say that the subspace $\mathcal{H}_2$ is *traced out*. The partial trace $\rho|_{\mathcal{H}_1}$ enables us to calculate probabilities of the results of the measurements bearing only on the subspace $\mathcal{H}_1$. We assume that the rows and columns of the matrices ρ and $\rho|_{\mathcal{H}_1}$ are labeled by the vectors in the basis $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{B}_1$, respectively. For example, $\rho\left(|u_{i_1}\rangle\,|v_{j_1}\rangle\,, |u_{i_2}\rangle\,|v_{j_2}\rangle\right)$ is the entry of ρ at the row labeled by $|u_{i_1}\rangle \otimes |v_{j_1}\rangle$ and the column labeled by $|u_{i_2}\rangle \otimes |v_{j_2}\rangle$. With this notation, the partial trace $\rho|_{\mathcal{H}_1}$ is defined as follows:

$$\rho|_{\mathcal{H}_1}\left(|u_{i_1}\rangle\,, |u_{i_2}\rangle\right) = \sum_{j=1}^{2^m} \rho\left(|u_{i_1}\rangle\,|v_j\rangle\,, |u_{i_2}\rangle\,|v_j\rangle\right).$$

Again let $\mathcal{E}$ be a subspace of $\mathcal{H}_1$. We can identify it with subspace $\mathcal{E} \otimes \mathcal{H}_2$ of $\mathcal{H}$. Let $\mathcal{P}\colon \mathcal{H}_1 \longrightarrow \mathcal{E}$ be the projection operator associated with $\mathcal{E}$. The operator $\mathcal{P}$ can be extended to the whole space $\mathcal{H}$ in a natural way as the operator $\mathcal{P} \otimes \mathrm{Id}_{\mathcal{H}_2}$, where $\mathrm{Id}_{\mathcal{H}_2}$ is the identity operator on $\mathcal{H}_2$. Then the probability that the result of the measurement is in the subspace $\mathcal{E}$ is equal to $\mathrm{Tr}(\mathcal{P}\,\rho|_{\mathcal{H}_1})$. (For more details on density matrices of mixed states and the partial trace, see, e.g., [8].)

Let C be a quantum circuit. For inputs of C it is possible to consider mixed states along with pure states. Toward this end, each input is substituted by its density matrix and each gate g of C by a *superoperator* $\widetilde{g}$ that maps density matrices to density matrices. In fact, if the unitary operator of the gate g is U, then the action of $\widetilde{g}$ on the density matrix ρ is as follows:

$$\widetilde{g}(\rho) = g \circ \rho = U \rho\, U^{\dagger}. \tag{3.1}$$

LEMMA 3.1 (see [1]). *If the gates g_1 and g_2 operate on disjoint sets of qubits, then for any density matrix ρ we have $g_1 \circ g_2 \circ \rho = g_2 \circ g_1 \circ \rho$.*

First we show that every quantum formula is equivalent to a circuit based on this new definition. Let $\mathcal{F}$ be a quantum formula on a basis of d-bit gates. Construct a circuit $\mathcal{C}$ from $\mathcal{F}$ by the following transformations. In each gate g, performing the unitary operation $U \in \mathbf{U}\left(2^d\right)$, keep the only output which is connected to the output and substitute the operator U by the superoperator $[g] = \mathrm{Tr}_{\mathcal{H}} \circ \widetilde{g}$, where $\mathcal{H}$ is the $(d-1)$-dimensional subspace spanned by the qubits removed from the output of this gate. The fact that the circuit $\mathcal{C}$ computes the same function as the formula $\mathcal{F}$ follows from Lemma 2.3. Thus the underlying graph of the circuit $\mathcal{C}$ is the same as the computation tree of the formula $\mathcal{F}$, where the node corresponding with the gate g computes the superoperator $[g]$.

Now let $\mathcal{C}$ be a circuit based on this new definition. We construct a quantum formula $\mathcal{F}$ from $\mathcal{C}$ by simply substituting each gate of $\mathcal{C}$, computing the superoperator T, by a $(d+2)$-input $(d+2)$-output unitary gate U; only one output of this gate is connected to the next gate and the other outputs never interact with any other qubit. Therefore $\mathcal{F}$ satisfies our original definition of quantum formula. The only thing that remains is to show how we can choose the unitary operators U such that

the formula $\mathcal{F}$ computes the same Boolean function as $\mathcal{C}$. The following theorem guarantees the existence of the correct operator U for each gate of $\mathcal{C}$. Here $\mathbf{L}(\mathcal{H})$ is the space of linear operators on the Hilbert space $\mathcal{H}$ and for unitary operator U on $\mathcal{H}$, the operator $\mathcal{O}_U \in \mathbf{L}(\mathcal{H})$ is defined as $\mathcal{O}_U(M) = U\,M\,U^\dagger$.

THEOREM 3.2 (see [7, 13, 16, 20]). *Suppose that $T : \mathbf{L}(\mathcal{H}_1) \longrightarrow \mathbf{L}(\mathcal{H}_2)$ is a trace-preserving and completely positive superoperator. Then there are Hilbert spaces $\mathcal{G}_1$ and $\mathcal{G}_2$, where $\dim(\mathcal{G}_1) = (\dim(\mathcal{H}_2))^2$ and $\dim(\mathcal{G}_2) = \dim(\mathcal{H}_1) \cdot \dim(\mathcal{H}_2)$, and there is a unitary operator $U : \mathcal{H}_1 \otimes \mathcal{G}_1 \longrightarrow \mathcal{H}_2 \otimes \mathcal{G}_2$ such that $T = \mathrm{Tr}_{\mathcal{G}_2} \circ \mathcal{O}_U$.*

We would like to mention that from now on it might be more useful to accept the new modified definition as the standard one for quantum formulas in the literature.

4. The lower bound. Let $f(x_1, \ldots, x_n)$ be a Boolean function. Let $X = \{x_1, \ldots, x_n\}$ be the set of the input variables. Consider a partition $\{S_1, \ldots, S_k\}$ of X; i.e.,

$$X = \bigcup_{j=1}^{k} S_j \quad \text{and} \quad S_{j_1} \cap S_{j_2} = \emptyset \quad \text{for} \quad j_1 \neq j_2.$$

Let $n_j = |S_j|$ for $j = 1, \ldots, k$. Let $\mathcal{F}_j$ be the set of all subfunctions of f on S_j obtained by fixing the variables outside S_j in all possible ways. We denote the cardinality of $\mathcal{F}_j$ by σ_j.

As an example, we compute the above parameters for the element distinctness function ED_n (see [6]). Let $n = 2\ell \log \ell$ (so $\ell = \Omega(n/\log n)$) and divide the n inputs of the function into ℓ strings each of $2\log\ell$ bits. Then the value of ED_n is 1 if and only if these ℓ strings are pairwise distinct. We consider the partition $(S_1, \ldots, S_\ell)$ such that each S_j contains all variables of the same string. Thus $n_j = |S_j| = 2\log\ell$. Each string in S_j represents an integer from the set $\{0, 1, \ldots, \ell^2 - 1\}$. The function ED_n is symmetric with respect to S_j's; so $|\mathcal{F}_j| = |\mathcal{F}_{j'}|$. To estimate $|\mathcal{F}_1|$, note that if the strings $(z_2, \ldots, z_\ell)$ in $S_2, \ldots, S_\ell$ represent distinct integers, then the corresponding subfunction is different from any subfunction corresponding to any other string. Therefore $\sigma_j = |\mathcal{F}_1| \geq \binom{\ell^2}{\ell-1} > \ell^{\ell-1}$.

THEOREM 4.1. *Every quantum formula computing f has size*

$$\Omega\left(\sum_{1\leq j\leq k} \frac{\log(\sigma_j)}{\log\log(\sigma_j)}\right).$$

Proof. We give a proof for any basis consisting of 2-input 2-output quantum gates. The proof for bases with more than two inputs is a simple generalization of this proof.

Let F be a formula computing f. Let Σ_j be the set of input wires of F labeled by a variable from S_j, and let $s_j = |\Sigma_j|$. Then

$$\mathsf{size}(F) = \Omega\left(\sum_{1\leq j\leq k} s_j\right). \tag{4.1}$$

We want to consider the formulas obtained from F by letting the input variables not in Σ_j to be some constant value $|0\rangle$ or $|1\rangle$. In this regard, let P_j be the set of all paths from an input wire in Σ_j to the output of F. Finally, let G_j be the set of gates of F where two paths from P_j intersect. Then $|G_j| \leq s_j$.

Let τ be an assignment of $|0\rangle$ or $|1\rangle$ to the input variable wires *not* in Σ_j. We denote the resulting formula by F_τ. Thus F_τ computes a Boolean function $f_\tau\colon \{0,1\}^{n_j} \longrightarrow \{0,1\}$ which is a subfunction of f and a member of $\mathcal{F}_j$. Consider a path

$$\pi = (g_1, g_2, \dots, g_m), \qquad m > 2, \tag{4.2}$$

in F_τ, where g_1 is an input wire or a gate in G_j, g_m is a gate in G_j or the output wire of F, and $g_\ell \notin G_j$ for $1 < \ell < m$.

To show how we can squeeze paths like (4.2) (this is the essence of the Nechiporuk's method), we introduce the following notations. We consider a natural ordering $\gamma_1, \gamma_2, \dots, \gamma_t$ on the gates of the formula F_τ, and regard F_τ as a computation in t steps, where at step ℓ the corresponding gate γ_ℓ is performed. We say two qubits q_1 and q_2 are *strong companions* of each other at step ℓ if there is a gate γ_j such that $j \le \ell$ and q_1 and q_2 are inputs of γ_j. We say qubits q_1 and q_2 are *companions* of each other at step ℓ if there exists a sequence $r_1, r_2, \dots, r_p$ of qubits such that $r_1 = q_1$, $r_p = q_2$, and r_j and r_{j+1} (for $1 \le j \le p-1$) are strong companions of each other at step ℓ (see Figure 4.1). If q_1 and q_2 are companions at step ℓ, then they are also companions at any step after ℓ. For a gate $g = \gamma_k$, we define the *set of companions* of g as the union of all companions of input qubits of g at step k.

Suppose that in the path (4.2), $g_1 = \gamma_{j_0}$, $g_m = \gamma_{j_1}$, the inputs of g_1 are q_0 and q_1, the output of γ_{j_0} from the path (4.2) is the qubit q_0, and the input of γ_{j_1} not from the path (4.2) is the qubit q_2 (see Figure 4.2). Note that q_0 is the companion of q_2 at step j_1. Let Q_π be the union of all sets of companions of $g_2, \dots, g_{m-1}$ minus q_0 and q_1 and their companions at step j_1. Let C_0 be the circuit defined by the gates $g_1, \dots, g_{m-1}$ from the path (4.2). Suppose that $|Q_\pi| = v$ and consider C_0 as an operation acting on $\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^{2^v}$. To study the action of the subcircuit C_0, it is enough to consider the action of C_0 on the computational basis vectors of the space $\mathcal{H}$. Therefore, while the inputs of C_0 as a subcircuit of F_τ are in general entangled states, we have to study only the action of C_0 on the computational basis vectors which are product states. We label the inputs $|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |\alpha\rangle \in \mathcal{H}$ of C_0 in such a way that when C_0 acts as a subformula of F_τ, then $|\alpha_0\rangle$, $|\alpha_1\rangle$, and $|\alpha\rangle$ are replaced by q_0, q_1, and the companion qubits in Q_π, respectively. Note that because F_τ is a formula, all qubits in Q_π are constant inputs of F_τ and do not intersect any other path like (4.2). Therefore, when C_0 acts as a subformula of F_τ, the input $|\alpha\rangle$ of the subcircuit C_0 is the same for all possible inputs for $|\alpha_0\rangle$ and $|\alpha_1\rangle$. Therefore, let $\widetilde{C_0}$ be a circuit such that on input $|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |0\cdots 0\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^{2^v}$, it first computes $|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |\alpha\rangle$ then performs the action of C_0 on $|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |\alpha\rangle$. Then if we replace C_0 by $\widetilde{C_0}$ and assign the value $|0\rangle$ to the qubits in Q_π, the result is a circuit equivalent to F_τ. Suppose that the action of $\widetilde{C_0}$ is defined as follows:

$$|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |0\cdots 0\rangle \longrightarrow \sum_{c_0,c_1\in\{0,1\}} |c_0\rangle \otimes |c_1\rangle \otimes \left|A_{c_0,c_1}^{\alpha_0,\alpha_1}\right\rangle, \tag{4.3}$$

where $\alpha_0, \alpha_1 \in \{0,1\}$, and $\left|A_{c_0,c_1}^{\alpha_0,\alpha_1}\right\rangle \in \mathbb{C}^{2^v}$ may be not a unit vector. Let $\mathcal{A}_\pi \subseteq \mathbb{C}^{2^v}$ be the vector space spanned by $\left|A_{c_0,c_1}^{\alpha_0,\alpha_1}\right\rangle$ for $\alpha_0, \alpha_1, c_0, c_1 \in \{0,1\}$ and $d = \dim(\mathcal{A}_\pi)$. Then $1 \le d \le 16$. Let $|A_1^\pi\rangle, \dots, |A_d^\pi\rangle$ be an orthonormal basis for $\mathcal{A}_\pi$. Then we can rewrite (4.3) as follows:

$$|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |0\cdots 0\rangle \longrightarrow \sum_{c_0,c_1\in\{0,1\}} \sum_{1\le j\le d} \lambda_{j,c_0,c_1}^{\alpha_0,\alpha_1} |c_0\rangle \otimes |c_1\rangle \otimes \left|A_j^\pi\right\rangle. \tag{4.4}$$

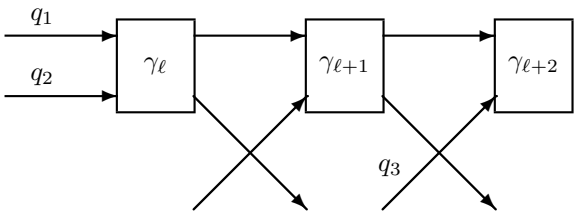


FIG. 4.1. *The qubits q_1 and q_2 are strong companions at step ℓ; the qubits q_2 and q_3 are companions at step $\ell + 2$.*

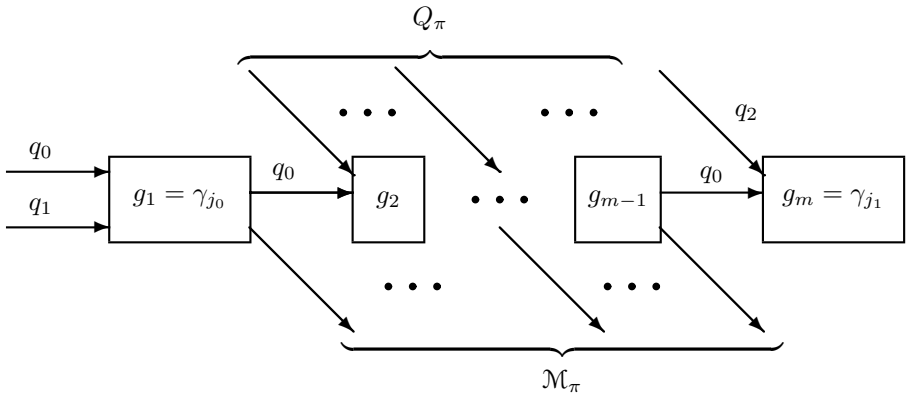


FIG. 4.2. *Squeezing a path.*

Let $\mathcal{M}_\pi$ be the set of those unitary operations that are performed after one of the gates $g_1, \ldots, g_{m-1}$ on some qubits in Q_π before the step j_1. Since qubits in Q_π do not interact with any other path of the form (4.2), by Lemma 2.3(b) we can postpone all operations in $\mathcal{M}_\pi$ after we have computed the output of g_m. Let $\pi_1, \ldots, \pi_k$ be a natural ordering on the paths like (4.2) on all paths in P_j (i.e., the last gate of π_{j+1} is not performed before the last gate of π_j). Consider the sets of postponed operations $\mathcal{M}_{\pi_1}, \ldots, \mathcal{M}_{\pi_k}$. Once again Lemma 2.3 implies that we can postpone operations in $\mathcal{M}_{\pi_1}$ after the last gate of π_2; then we can postpone operations in $\mathcal{M}_{\pi_1}$ and $\mathcal{M}_{\pi_2}$ after the last gate of π_3, and so on. Repeating this argument shows that we can postpone all operations in $\mathcal{M}_{\pi_1}, \ldots, \mathcal{M}_{\pi_k}$ after we compute the output qubit. In this way, the state of the output qubit, before the postponed operations $\mathcal{M}_{\pi_1}, \ldots, \mathcal{M}_{\pi_k}$ are applied, is of the form

$$|0\rangle \otimes |M\rangle + |1\rangle \otimes |N\rangle\,, \tag{4.5}$$

where the first qubit is the output qubit, and $|M\rangle$ and $|N\rangle$ are superpositions of tensor products of orthonormal vectors $\left|A_k^{\pi_j}\right\rangle$ used in (4.4). By Fact 2.2, these tensor products of the vectors $\left|A_k^{\pi_j}\right\rangle$ are unit vectors and pairwise orthogonal. The unitary operations in the sets $\mathcal{M}_{\pi_j}$ (for paths π_j of the form (4.2)), which are postponed to the end, do not change the lengths of $|M\rangle$ and $|N\rangle$. Thus, as far as the computation of the Boolean function f_τ is concerned, we can ignore all the postponed unitary operations. For this reason we construct the circuit $\overline{F_\tau}$ from the formula F_τ by eliminating all postponed operations in $\mathcal{M}_{\pi_j}$, substituting for each path π_j of the form (4.2) the companion qubits in Q_{π_j} by four new qubits, and the unitary operation (4.4) by the

operation defined as

$$|\alpha_0\rangle \otimes |\alpha_1\rangle \otimes |0000\rangle \longrightarrow \sum_{c_0,c_1\in\{0,1\}} \sum_{0\le j\le 15} \lambda_{j,c_0,c_1}^{\alpha_0,\alpha_1} |c_0\rangle \otimes |c_1\rangle \otimes |j\rangle\,. \tag{4.6}$$

The output of the circuit $\overline{F_\tau}$, instead of (4.5), is of the form

$$|0\rangle \otimes |M'\rangle + |1\rangle \otimes |N'\rangle\,, \tag{4.7}$$

where $\||M\rangle\| = \||M'\rangle\|$ and $\||N\rangle\| = \||N'\rangle\|$. Therefore the circuit $\overline{F_\tau}$ computes f_τ. Moreover,

$$\mathsf{size}(\overline{F_\tau}) = O(s_j),$$

and for another assignment τ', the corresponding circuit $\overline{F_{\tau'}}$ differs from $\overline{F_\tau}$ only at unitary operations defined by (4.6).

The above discussion implies that σ_j, the number of subfunctions on S_j, is at most the number of different Boolean functions computed by size $O(s_j)$ quantum circuits. Therefore, by Theorem 2.1, we get

$$\sigma_j \le 2^{O(s_j \log s_j)}.$$

Therefore $s_j = \Omega(\log(\sigma_j)/\log\log(\sigma_j))$. Now the theorem follows from (4.1). □

We would like to mention that the fact that a path like (4.2) can be squeezed to a path of constant length is a special case of the general property of the superoperators stated in Theorem 3.2.

To apply the general bound of the above theorem, we could consider any of the several explicit functions used in the case of Boolean formulas (see [11, 26]). As we mentioned in the beginning of this section, we consider the element distinctness function ED_n. For this function, $\sigma_j > \ell^{\ell-1}$, where $\ell = \Omega(n/\log n)$. Therefore, we get the lower bound $\Omega(\ell^2) = \Omega(n^2/\log^2 n)$ for the formula size.

THEOREM 4.2. *Any quantum formula computing* ED_n *has size* $\Omega(n^2/\log^2 n)$.

5. Quantum formulas vs. Boolean circuits. In this section we show that quantum formulas are not more powerful than Boolean circuits. Therefore as a model of computation, their strength lies between Boolean formulas and Boolean circuits.

Following the idea developed in section 3, we consider a quantum formula as a quantum circuit operating on *mixed states*. For the details of quantum circuits with mixed states, see [1, 13]. Before we start the proof of the main result of this section, we need to see how we can bound errors in quantum circuits with mixed states. Toward this end we need a suitable norm on superoperators. Each superoperator T which maps density matrices to density matrices is a linear mapping of the form $\mathbf{L}(\mathcal{H}_1) \longrightarrow \mathbf{L}(\mathcal{H}_2)$, where $\mathcal{H}_1$ and $\mathcal{H}_2$ are finite-dimensional Hilbert spaces and $\mathbf{L}(\mathcal{H}_j)$ is the set of linear operators on $\mathcal{H}_j$. Note that $\mathbf{L}(\mathcal{H}_j)$ itself is a linear space. Let $\mathcal{H}$ be an m-dimensional Hilbert space. There are several norms on the space $\mathbf{L}(\mathcal{H})$, of which we need the following ones. Let $A \in \mathbf{L}(\mathcal{H})$. We identify A with its $m \times m$ matrix (a_{ij}). The first norm is

$$M(A) = m \max_{i,j} |a_{ij}|.$$

The usual norm is defined as

$$\|A\| = \sup_{|x\rangle\neq 0} \frac{\|A\,|x\rangle\|}{\||x\rangle\|} = \max\left\{ \sqrt{\lambda} \colon \lambda \in \mathrm{Spec}(A^\dagger A) \right\},$$

where $\mathrm{Spec}(M)$ is the *spectrum* of the matrix M, i.e., the set of the eigenvalues of M. The other norm is the *trace norm*

$$\|A\|_{\mathrm{Tr}} = \sum_{\lambda \in \mathrm{Spec}(A^\dagger A)} \sqrt{\lambda}.$$

We need the next norm $\|\cdot\|_\star$ to define another norm: let T be a linear operator that maps matrices to matrices; i.e., $T \in \mathbf{L}(\mathbf{L}(\mathcal{H}))$; then

$$\|T\|_\star = \sup_{A \neq 0} \frac{\|TA\|_{\mathrm{Tr}}}{\|A\|_{\mathrm{Tr}}}.$$

The last norm we consider is the *diamond norm*, defined in [13] and also in [1]. To define this norm, we consider a Hilbert space $\mathcal{G}$ such that $\dim(\mathcal{G}) \geq \dim(\mathcal{H})$ and we let

$$\|T\|_\diamond = \|T \otimes I_{\mathcal{G}}\|_\star,$$

where $I_{\mathcal{G}}$ is the identity operator on $\mathcal{G}$. The following are the basic properties of these norms:

(i) $\frac{1}{m} M(A) \leq \|A\| \leq M(A)$.
(ii) $\|A\|_{\mathrm{Tr}} \leq m \|A\|$.
(iii) $\|T(\rho)\|_{\mathrm{Tr}} \leq \|T\|_\diamond \|\rho\|_{\mathrm{Tr}}$, for the density matrix ρ.
(iv) $\|TR\|_\diamond \leq \|T\|_\diamond \|R\|_\diamond$.
(v) $\|T \otimes R\|_\diamond = \|T\|_\diamond \|R\|_\diamond$.
(vi) If $T = \widetilde{g}$, for some quantum gate g, or $T = \mathrm{Tr}_{\mathcal{F}}$, then $\|T\|_\diamond = 1$.

The properties (iii)–(vi) are proved in [1, 13].

For any operator $V \in \mathbf{L}(\mathcal{H})$ we define the operator $\mathcal{O}_V \in \mathbf{L}(\mathbf{L}(\mathcal{H}))$ as

$$\mathcal{O}_V(M) = V M V^\dagger, \qquad M \in \mathbf{L}(\mathcal{H}). \tag{5.1}$$

In [1, 13] it is proved that $\|\mathcal{O}_V - \mathcal{O}_W\|_\diamond \leq 2\|V - W\|$ if $\|V\| \leq 1$ and $\|W\| \leq 1$. We need the following general form of this inequality.

LEMMA 5.1. *Let* $\dim(\mathcal{H}) = m$. *For any* $V, W \in \mathbf{L}(\mathcal{H})$ *we have*

$$\|\mathcal{O}_V - \mathcal{O}_W\|_\diamond \leq 2m \|V - W\| \min(\|V\|, \|W\|) + m\|V - W\|^2.$$

Proof. We have (for $A \in \mathbf{L}(\mathcal{H} \otimes \mathcal{H})$)

$$\begin{aligned}
\|\mathcal{O}_V - \mathcal{O}_W\|_\diamond &= \sup_{A \neq 0} \|(\mathcal{O}_V \otimes I_{\mathcal{H}})A - (\mathcal{O}_W \otimes I_{\mathcal{H}})A\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}} \\
&= \sup_{A \neq 0} \left\|(V \otimes I_{\mathcal{H}})\, A\, (V^\dagger \otimes I_{\mathcal{H}}) - (W \otimes I_{\mathcal{H}})\, A\, (W^\dagger \otimes I_{\mathcal{H}})\right\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}} \\
&= \sup_{A \neq 0} \|(V \otimes I_{\mathcal{H}})\, A\, (V^\dagger \otimes I_{\mathcal{H}}) \\
&\quad - ((V + (W - V)) \otimes I_{\mathcal{H}})\, A\, ((V^\dagger + (W^\dagger - V^\dagger)) \otimes I_{\mathcal{H}})\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}} \\
&\leq \sup_{A \neq 0} \left\|(V \otimes I_{\mathcal{H}})\, A\, ((W^\dagger - V^\dagger) \otimes I_{\mathcal{H}})\right\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}} \\
&\quad + \sup_{A \neq 0} \left\|((W - V) \otimes I_{\mathcal{H}})\, A\, (V^\dagger \otimes I_{\mathcal{H}})\right\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}} \qquad (5.2) \\
&\quad + \sup_{A \neq 0} \left\|((W - V) \otimes I_{\mathcal{H}})\, A\, ((W^\dagger - V^\dagger) \otimes I_{\mathcal{H}})\right\|_{\mathrm{Tr}} / \|A\|_{\mathrm{Tr}}.
\end{aligned}$$

Since $\|MN\| \leq \|M\| \cdot \|N\|$, $\|M\| \leq \|M\|_{\mathrm{Tr}} \leq m\,\|M\|$, $\|M \otimes N\| = \|M\| \cdot \|N\|$, $\left\|M^\dagger\right\| = \|M\|$, and $\|I_{\mathcal{H}}\| = 1$, it follows that

$$\begin{aligned}\left\|(V \otimes I_{\mathcal{H}})\, A\, ((W^\dagger - V^\dagger) \otimes I_{\mathcal{H}})\right\|_{\mathrm{Tr}} &\leq m \left\|(V \otimes I_{\mathcal{H}})\, A\, ((W^\dagger - V^\dagger) \otimes I_{\mathcal{H}})\right\| \\ &\leq m\,\|V \otimes I_{\mathcal{H}}\| \cdot \|A\| \cdot \left\|(W^\dagger - V^\dagger) \otimes I_{\mathcal{H}}\right\| \\ &\leq m\,\|V\| \cdot \|A\|_{\mathrm{Tr}} \cdot \|W - V\|\,.\end{aligned}$$

By applying a similar reduction to the other terms of (5.2), we derive the following inequality:

$$\|\mathcal{O}_V - \mathcal{O}_W\|_\diamond \leq 2m\,\|V - W\| \cdot \|V\| + m\,\|V - W\|^2\,.$$

We can also derive a similar inequality with $\|V\|$ substituted by $\|W\|$. This completes the proof. □

We say two $n \times m$ matrices $A = (a_{ij})$ and $B = (b_{ij})$ are *δ-close* to each other if $|a_{ij} - b_{ij}| \leq \delta$ for every $1 \leq i \leq n$ and $1 \leq j \leq m$. If the $m \times m$ matrices A and B are δ-close to each other, then

$$\|A - B\| \leq M(A - B) \leq m\,\delta. \tag{5.3}$$

The following theorem formalizes the general form of the error bound for quantum circuits when approximating the unitary operator of each gate. This theorem is actually a generalization of a weaker theorem which has appeared in several papers (see, e.g., [1, 5, 13]). We need this generalization because once we substitute any unitary gate S of the original quantum circuit by some approximated gate T, in general we do not know whether $\|T\| \leq 1$ or not. (This is the assumption of the weaker version of this theorem.)

THEOREM 5.2. *Let $S_j, T_j \in \mathbf{L}(\mathbf{L}(\mathbb{C}^{2^d}))$, $1 \leq j \leq \ell$, be defined as $S_j = \mathcal{O}_{U_j}$ and $T_j = \mathcal{O}_{V_j}$, where $U_j \in \mathbf{U}(2^d)$ is unitary and V_j is δ-close to U_j. Then*

$$\|S_\ell \cdots S_3 S_2 S_1 - T_\ell \cdots T_3 T_2 T_1\|_\diamond \leq e^{\eta(d,\delta)\ell} - 1,$$

where $\eta(d, \delta) = 2^{2d+1}\delta\left(1 + 2^d\delta\right)$.

Proof. First note that

$$\begin{aligned}\|S_j - T_j\|_\diamond &= \left\|\mathcal{O}_{U_j} - \mathcal{O}_{V_j}\right\|_\diamond \\ &\leq 2^{d+1}\,\|U_j - V_j\|\,(1 + \|U_j - V_j\|) && \text{by Lemma 5.1} \\ &\leq 2^{2d+1}\delta\left(1 + 2^d\delta\right) && \text{by (5.3)} \\ &= \eta(d, \delta);\end{aligned}$$

and, by (vi),

$$\|T_j\|_\diamond \leq \|S_j\|_\diamond + \|S_j - T_j\|_\diamond \leq 1 + \eta(d, \delta). \tag{5.4}$$

We also have the following simple inequality:

$$\begin{aligned}\|M_2 M_1 - N_2 N_1\|_\diamond &= \|M_2(M_1 - N_1) - (M_2 - N_2)N_1\|_\diamond \\ &\leq \|M_2\|_\diamond\,\|M_1 - N_1\|_\diamond + \|N_1\|_\diamond\,\|M_2 - N_2\|_\diamond\,.\end{aligned} \tag{5.5}$$

Now, by repeated applications of (5.4) and (5.5), we have

$$\begin{aligned}
\|S_\ell \cdots S_3 S_2 S_1 - T_\ell \cdots T_3 T_2 T_1\|_\diamond &\le \|S_\ell \cdots S_3 S_2\|_\diamond \|S_1 - T_1\|_\diamond \\
&\quad + \|T_1\|_\diamond \|S_\ell \cdots S_3 S_2 - T_\ell \cdots T_3 T_2\|_\diamond \\
&\le \eta(d,\delta) + (1+\eta(d,\delta)) \|S_\ell \cdots S_3 S_2 - T_\ell \cdots T_3 T_2\|_\diamond \\
&\le \eta(d,\delta) + \eta(d,\delta)(1+\eta(d,\delta)) \\
&\quad + (1+\eta(d,\delta))^2 \|S_\ell \cdots S_3 - T_\ell \cdots T_3\|_\diamond \\
&\vdots \quad \vdots \\
&\le \eta(d,\delta) \sum_{j=0}^{\ell-1} \big(1+\eta(d,\delta)\big)^j \\
&= (1+\eta(d,\delta))^\ell - 1 \\
&\le e^{\eta(d,\delta)\ell} - 1. \qquad \square
\end{aligned}$$

The following theorem is the immediate consequence of the above theorem. Note that for a gate g the superoperator $\widetilde{g}$ is defined by (3.1).

THEOREM 5.3. *Let C be a quantum circuit composed of the gates $g_1, \ldots, g_s$. Suppose that each g_j is a d-bit gate computing the unitary operator $U_j \in \mathbf{U}(2^d)$. For each $1 \le j \le s$, let $V_j \in \mathbf{L}(\mathbb{C}^{2^d})$ be a δ-close matrix to U_j. Let $T_j \in \mathbf{L}(\mathbf{L}(\mathbb{C}^{2^d}))$ be defined as $T_j = \mathcal{O}_{V_j}$. For any input density matrix ρ_0, let*

$$\psi = (\widetilde{g}_s \otimes I_{\mathcal{H}_s}) \circ \cdots \circ (\widetilde{g}_2 \otimes I_{\mathcal{H}_2}) \circ (\widetilde{g}_1 \otimes I_{\mathcal{H}_1})\, \rho_0$$

be the output of C, where $\mathcal{H}_j$ is the Hilbert space generated by the qubits not involved with the gate g_j. Also, let

$$\zeta = (T_s \otimes I_{\mathcal{H}_s}) \cdots (T_2 \otimes I_{\mathcal{H}_2})(T_1 \otimes I_{\mathcal{H}_1})\, \rho_0$$

be the approximated output of C. Then

$$\|\psi - \zeta\|_{\mathrm{Tr}} \le \left(e^{\eta(d,\delta)s} - 1\right) \|\rho_0\|_{\mathrm{Tr}},$$

where $\eta(d,\delta) = 2^{2d+1}\delta\left(1 + 2^d\delta\right)$.

THEOREM 5.4. *Let $\mathcal{B}$ be a quantum basis. Then each quantum formula of size ℓ and depth d over the basis $\mathcal{B}$ can be simulated with error at most ε by a Boolean* circuit *of size $O(\ell\,\mu\,\log\mu\log\log\mu)$ and depth $O(d\log\mu)$, where $\mu = \lceil \log\ell - \log\varepsilon \rceil$.*

Proof. The basic idea of the simulation is to look at the behavior of a quantum formula as a quantum circuit acting on density matrices of mixed states. We assume, w.l.o.g., that each gate in the basis $\mathcal{B}$ is a 2-bit gate.

Consider a quantum formula $\mathcal{F}$ over the basis $\mathcal{B}$; suppose that $\mathcal{F}$ has t inputs (constant or variable) and computes the Boolean function $f\colon \{0,1\}^n \longrightarrow \{0,1\}$. We show that there is a Boolean circuit $\mathcal{C}$ that for any input $\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$ simulates the action of $\mathcal{F}$ on $\boldsymbol{a}$. Let $|\alpha\rangle = |0\rangle \otimes |A_0\rangle + |1\rangle \otimes |A_1\rangle$ be the output of $\mathcal{F}$ on the input $\boldsymbol{a}$. Suppose that the first qubit is the output bit. If we *trace out* the nonoutput bits of $|\alpha\rangle$, the result is a 2×2 density matrix $\rho_{\mathrm{final}} = \rho|_{|\alpha\rangle}$. From ρ_{final} it is easy to calculate the probability of acceptance of $\mathcal{F}$. The formula structure of $\mathcal{F}$ allows us to calculate the density matrix ρ_{final} without going to the 2^t-dimensional space. The Boolean circuit $\mathcal{C}$ finds the density matrix ρ_{final} by simulating the gates of $\mathcal{F}$ step by step.

Since the trace norm of a density matrix is equal to its trace, it follows that $\|\rho_0\|_{\mathrm{Tr}} = 1$, where ρ_0 is the density matrix of the input.

Now the gates of $\mathcal{F}$ are no longer acting on pure states, but they are acting on mixed states. If the input of a gate g_j (performing the unitary operation U_j) is the 4×4 density matrix ρ, then the output is the density matrix $\rho' = U_j\, \rho\, {U_j}^\dagger$. Of the two output bits q_1 and q_2 of this gate, only one, say, q_1, is connected to the output bit of $\mathcal{F}$. Therefore we trace out the system representing q_2 and consider the new density matrix $\rho|_{q_1} = \mathrm{Tr}_{q_2}\rho'$ for q_1. By repeating this process for each gate of $\mathcal{F}$ we finally get the desired density matrix ρ_{final}. The correctness of this process follows from Lemma 3.1.

The Boolean circuit $\mathcal{C}$ can simulate the calculations of these density matrices ρ_{q_1}. The only problem for this simulation is the proper approximation of the entries of unitary matrices U_j. If we substitute each entry of U_j by its first $\mu = -\lceil \log_2 \delta \rceil$ bits, then we get a matrix that is δ-close to U_j. Let $\widetilde{\mathcal{F}}$ be the resulting formula and $\widetilde{\rho}_{\mathrm{final}}$ be the output of $\widetilde{\mathcal{F}}$. Then, by Theorem 5.3, $\|\rho_{\mathrm{final}} - \widetilde{\rho}_{\mathrm{final}}\|_{\mathrm{Tr}} \leq e^{\eta(d,\delta)\ell} - 1$. Therefore if $\delta = O\left(\frac{\varepsilon}{\ell}\right)$, i.e., $\mu = O\left(\log \ell - \log \varepsilon\right)$, then the simulation of $\mathcal{F}$ by $\widetilde{\mathcal{F}}$ has at most ε error. The theorem now follows from the fact that addition and multiplication of m bits numbers can be carried out by Boolean circuits of size $O(m \log m \log\log m)$ and depth $O(\log m)$ (see [19, 26]). □

Why does this proof not provide a Boolean *formula* instead of a Boolean circuit? The reason is that to calculate $\rho' = U_j\, \rho\, {U_j}^\dagger$, we need four copies of each entry of ρ. Thus the fan-out of the gates in the Boolean circuit obtained from the formula $\widetilde{\mathcal{F}}$ is 4. This means that the Boolean formula equivalent to this Boolean circuit, in general, has size exponential in ℓ; this size is at least $\Omega\left(\ell^3\right)$ if the graph of $\mathcal{F}$ is a full binary tree.

6. Concluding remarks. We have extended a classical technique for proving lower bound for Boolean formula size to quantum formulas. The difficult part was to effectively deal with the phenomenon of entanglement of qubits. While we have been successful in extending a classical technique to the quantum case, the challenges encountered indicate that in general the problem of extending methods of Boolean case to the quantum case may not have simple solutions. For example, even the seemingly simple issue of the exact relationship between quantum formulas and quantum circuits has not been resolved. In the Boolean case, simulation of circuits by formulas is a simple fact, but in the quantum case it is not clear whether every quantum circuit can be simulated by a quantum formula. In particular, it is not clear in the process of going from quantum circuits to formulas how we can modify the underlying entanglement of qubits while keeping the probability of reaching the final answer the same. We were also able to show that it is possible to simulate quantum formulas with Boolean circuits of almost the same size. It does not seem that Boolean formulas could efficiently simulate their quantum counterparts. Therefore evidently quantum formulas, as a model of computation, are more powerful than Boolean formulas and less powerful than Boolean circuits. A better understanding of the relations between these models remains a challenging problem.

Appendix. Counting the number of Boolean functions computed by quantum circuits of a given size.

In this appendix we prove the following upper bound.

THEOREM A.1. *The number of different n-variable Boolean functions that can be computed by size N quantum circuits ($n \leq N$) with d-input d-output elementary*

gates (for some constant d) is at most $2^{O(nN)+O(N \log N)}$.

Proof. Our proof is based on Warren's bound on the number of different sign-assignments to real polynomials [25]. We begin with some necessary notations.

Let $P_1(x_1, \ldots, x_t), \ldots, P_m(x_1, \ldots, x_t)$ be real polynomials. A *sign-assignment* to these polynomials is a system of inequalities

$$P_1(x_1, \ldots, x_t) \,\Delta_1\, 0, \ldots, \quad P_m(x_1, \ldots, x_t) \,\Delta_m\, 0, \tag{A.1}$$

where each Δ_j is either "$<$" or "$>$." The sign-assignment (A.1) is called *consistent* if this system has a solution in $\mathbb{R}^t$.

THEOREM A.2 (Warren [25]). *Let $P_1(x_1, \ldots, x_t), \ldots, P_m(x_1, \ldots, x_t)$ be real polynomials, each of degree at most d. Then there are at most $(4edm/t)^t$ consistent sign-assignments of the form* (A.1).

We consider the class of quantum circuits of size N with d-bit gates computing n-variable Boolean functions. W.l.o.g., we can assume that n', the number of input wires of such circuits, is at most $d \cdot N$. We define an equivalence relation $\bowtie$ on such circuits: we write $C_1 \bowtie C_2$ if and only if C_1 and C_2 differ only in the label of their gates; in other words, C_1 and C_2 have the same underlying graph, but the corresponding gates in these circuits may compute different unitary operations. The number of different equivalence classes is at most

$$\binom{n'}{d}^N \leq (dN)^{dN} = 2^{O(N \log N)}.$$

Now we find an upper bound for the number of different Boolean functions that can be computed by circuits in the same equivalence class. Fix an equivalence class $\mathcal{E}$. We use the variables $a_1 + ib_1, a_2 + ib_2, \ldots, a_\mu + ib_\mu$, where $\mu = d^2N$, to denote the entries of the matrices of the gates of a circuit C in $\mathcal{E}$. By substituting appropriate values to the variables $a_1, \ldots, a_\mu, b_1, \ldots, b_\mu$, we get all circuits in $\mathcal{E}$. On input $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$, the probability that C outputs 1 can be represented by a *real* polynomial

$$P_\alpha(a_1, \ldots, a_\mu, b_1, \ldots, b_\mu).$$

The degree of P_α is at most $2N$. There are 2^n polynomials P_α, and the number of different Boolean functions which can be computed by C by changing the unitary operators of its gates is at most the number of different consistent sign-assignments to the following system:

$$P_\alpha(a_1, \ldots, a_\mu, b_1, \ldots, b_\mu) - \tfrac{2}{3},$$

$$P_\alpha(a_1, \ldots, a_\mu, b_1, \ldots, b_\mu) - \tfrac{1}{3},$$

for $\alpha \in \{0,1\}^n$. By Theorem A.2 this number is bounded from the above by

$$\left(\frac{4e(2N)2^{n+1}}{2\mu}\right)^{2\mu} = 2^{O(nN)+O(N \log N)}. \qquad \square$$

Acknowledgments. We thank Alexei Kitaev for helpful discussions on the early version of this paper. We also wish to thank the reviewer for the careful reading and useful comments.

REFERENCES

[1] D. Aharonov, A. Kitaev, and N. Nisan, *Quantum circuits with mixed states*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1998, pp. 20–30.

[2] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A (3), 52 (1995), pp. 3457–3467.

[3] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, in Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science, 1998, IEEE Computer Society Press, Los Alamitos, CA, pp. 352–361.

[4] C. H. Bennett, *Time/space trade-offs for reversible computation*, SIAM J. Comput., 18 (1989), pp. 766–776.

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

[6] R. B. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, J. van Leeuwen, ed., Elsevier Science, New York, MIT Press, Cambridge, MA, 1990, pp. 757–804.

[7] M.- D. Choi, *Completely positive linear maps on complex matrices*, Linear Algebra Appl., 10 (1975), pp. 285–290.

[8] C. Cohen-Tannoudji, B. Diu, and F. Laloë, *Quantum Mechanics.* I, John Wiley and Sons, New York, 1977.

[9] D. Deutsch, *Quantum theory, the Church–Turing principle and the universal quantum computer*, Proc. Roy. Soc. London Ser. A, 400 (1985), pp. 97–117.

[10] D. Deutsch, *Quantum computational networks*, Proc. Roy. Soc. London Ser. A, 425 (1989), pp. 73–90.

[11] P. E. Dunne, *The Complexity of Boolean Networks*, Academic Press, London, 1988.

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

[13] A. Kitaev, *Quantum computations: Algorithms and error correction*, Russian Math. Surveys, 52 (1997), pp. 1191–1249.

[14] A. Kitaev, *private communication,* 2000.

[15] E. Knill, *Approximating by Quantum Circuits*, Los Alamos National Laboratory e-print quant-ph/9508006, 1995.

[16] K. Kraus, *States, Effects, and Operations: Fundamental Notions of Quantum Theory*, Springer-Verlag, Berlin, New York, 1983.

[17] R. Y. Levine and A. T. Sherman, *A note on Bennett's time-space tradeoff for reversible computation*, SIAM J. Comput., 19 (1990), pp. 673–677.

[18] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, New York, 2000.

[19] A. Schönhage and V. Strassen, *Schnelle multiplikation grosser zahlen*, Computing, 7 (1971), pp. 281–292.

[20] B. Schumacher, *Sending entanglement through noisy quantum channels*, Phys. Rev. A (3), 54 (1996), pp. 2614–2628.

[21] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

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

[23] Gy. Turán and F. Vatan, *On the computation of Boolean functions by analog circuits of bounded fan-in*, J. Comput. System Sci., 54 (1997), pp. 199–212.

[24] L. G. Valiant, *Short monotone formulae for the majority function*, J. Algorithms, 5 (1984), pp. 363–366.

[25] H. E. Warren, *Lower bounds for approximation by nonlinear manifolds*, Trans. Amer. Math. Soc., 133 (1968), pp. 167–178.

[26] I. Wegener, *The Complexity of Boolean Functions*, Teubner-Wiley, New York, 1987.

[27] A. Yao, *Quantum circuit complexity*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 352–361.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 477–482

A 2-APPROXIMATION ALGORITHM FOR THE DIRECTED MULTIWAY CUT PROBLEM*

JOSEPH (SEFFI) NAOR† AND LEONID ZOSIN‡

Abstract. A directed multiway cut separates a set of terminals $T = \{s_1, \ldots, s_k\}$ in a directed capacitated graph $G = (V, E)$. Finding a minimum directed multiway cut is an NP-hard problem. We give a polynomial-time algorithm that achieves an approximation factor of 2 for this problem. This improves the result of Garg, Vazirani, and Yannakakis [*Proceedings of the 21st International Colloquium on Automata, Languages, and Programming*, Jerusalem, Israel, 1994, pp. 487–498], who gave an algorithm that achieves an approximation factor of $2 \log k$. Our approximation algorithm uses a novel technique for relaxing a multiway flow function in order to find a directed multiway cut. It also implies that the integrality gap of the linear program for the directed multiway cut problem is at most 2.

Key words. approximation algorithms, combinatorial optimization, multicommodity flow, multiway cut, directed graph

AMS subject classifications. 05C85, 68R10, 68Q20, 68Q25, 68Q35, 90C05, 94C15, 68W25

PII. S009753979732147X

1. Introduction.

We consider the minimum directed multiway cut problem in this paper. Let $G = (V, E)$ be a directed graph and let c_e, for all $e \in E$, be a nonnegative capacity associated with the edge set E. Let $T = \{s_1, \ldots, s_k\} \subseteq V$ be a set of terminals. A *directed multiway cut* in G is a set of edges whose deletion disconnects all directed paths between pairs of terminals. This notion is a natural generalization of $\{s, t\}$-cuts. The capacity of a multiway cut is defined to be the sum of the capacities of the edges belonging to it. A *minimum directed multiway cut* is a directed multiway cut of minimum capacity. The minimum directed multiway cut problem was considered by Garg, Vazirani, and Yannakakis [4], who showed that it is NP-hard and MAX SNP-hard even for two terminals [4] and also gave an algorithm that achieves an approximation factor of $2 \log k$.

We present a polynomial-time approximation algorithm for the minimum directed multiway cut problem that achieves an approximation factor of 2, thus improving on the approximation factor obtained by [4]. Our factor is very close to the approximation factors known for the undirected version of the problem. Dahlhaus et al. [2] gave a $(2 - 2/k)$-approximation algorithm for the undirected *edge* multiway cut problem. This result was recently improved by [1] to $3/2 - 1/k$ and further by [6] to $1.3438 - \epsilon_k$, where $\epsilon_k > 0$. For the undirected *vertex* multiway cut problem, Garg, Vazirani, and Yannakakis [4] gave a $(2 - 2/k)$-approximation algorithm. Note that in the directed case there is no need to distinguish between the edge and vertex versions of the problem.

The minimum directed multiway cut problem can be cast as an integer linear

*Received by the editors May 14, 1997; accepted for publication (in revised form) April 26, 2000; published electronically August 22, 2001. A preliminary version of this paper appeared in *Proceedings of the 38th IEEE Conference on Foundations of Computer Science*, Miami Beach, FL, 1997, pp. 548–553.

http://www.siam.org/journals/sicomp/31-2/32147.html

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

‡ITG, Inc., 44 Farnsworth Street, 9th Floor, Boston, MA 02210 (lzosin@itginc.com). Most of this work was done while this author was a doctoral student at the Computer Science Department, Technion, Haifa 32000, Israel.

program. The linear relaxation of this program is an assignment of lengths to the edges such that the distance between every pair of terminals is at least 1. The dual program of the linear relaxation is a multicommodity flow function, where commodity i, $1 \leq i \leq k$, is defined to be the flow that originates in any terminal $j \neq i$, $1 \leq j \leq k$, and whose destination is terminal s_i. The goal is to maximize the net flow entering the terminals. We refer to this problem as the *multiway flow* problem. By the duality theorem, the optima of the multiway flow problem and the linear relaxation of the multiway cut problem are equal.

The main technique used by our algorithm is a novel relaxation of the multiway flow problem. A *relaxed multiway flow* function allows the total flow on an edge $e \in E$ (taken over all commodities) to go up to $2 \cdot c_e$. However, the flow on edge e belonging to the same commodity is not allowed to exceed the capacity c_e. We call the dual problem of the relaxed multiway flow problem the *relaxed multiway cut* problem. A relaxed multiway cut attaches multiple lengths to each edge that correspond to the (primal) constraints on it. The constraints are with respect to the distances between pairs of terminals.

We present an approximation algorithm that achieves a factor of 2 for the directed multiway cut problem and is based on rounding an optimal relaxed multiway cut. We show that the value of the multiway cut produced by the approximation algorithm does not exceed the value of an optimal relaxed multiway flow function. A relaxed multiway flow function can ship at most twice the amount of flow that a (standard) multiway flow function can ship. By the duality theorem, the optima of the relaxed multiway flow problem and the relaxed multiway cut problem are equal. Hence, we get that the approximation factor of our algorithm is 2. We also get that the integrality gap of the (standard) linear program for the multiway cut problem is at most 2. However, we do not see a direct way for proving this. A more intuitive explanation for this might be that a relaxed multiway flow function provides more "information" about the integral solution than a standard multiway flow function.

A relaxation similar in "spirit" of a multicommodity flow function was previously used by Even, Naor, and Zosin [3] for approximating the undirected subset feedback vertex set. However, their analysis completely differs from ours and is much more complicated. We believe that our technique for relaxing a multicommodity flow function will find more applications in the future. We note that our approximation algorithm uses complementary slackness in a way similar to the $(2 - 2/k)$-approximation algorithm of Garg, Vazirani, and Yannakakis [4] for the undirected vertex multiway cut problem.

2. Relaxed multiway flow. A multiway flow function is defined as follows. Define commodity i, $1 \leq i \leq k$, to be the flow that originates in any terminal $j \neq i$, $1 \leq j \leq k$, and whose destination is terminal s_i. Thus, each commodity is a multisource, single-sink flow function. An optimal multiway flow function maximizes the net flow entering the terminals subject to standard multicommodity flow constraints. We henceforth refer to this flow function as the *standard* multiway flow function.

We now define a *relaxed* version of the standard multiway flow function. Denote by f^i the nonnegative flow function of commodity i and denote by $f^i(e)$ the flow on edge e belonging to commodity i. Let $N^+(v)$ ($N^-(v)$) denote the set of incoming (outgoing) edges into (from) vertex v.

Objective function.

$$\text{Maximize} \qquad \sum_{i=1}^{k} \left(\sum_{e \in N^+(s_i)} f^i(e) - \sum_{e \in N^-(s_i)} f^i(e) \right).$$

Preservation. For commodity i, $1 \leq i \leq k$, for each vertex $v \in V - T$, the amount of flow of commodity i entering v is equal to the amount of flow of commodity i leaving v.

Intracommodity constraints. For each commodity i, $1 \leq i \leq k$, and for each edge $e \in E$,

$$f^i(e) \leq c_e.$$

Intercommodity constraints. For each edge $e \in E$,

$$\sum_{i=1}^{k} f^i(e) \leq 2 \cdot c_e.$$

The following lemma is immediate.

LEMMA 1. *A relaxed multiway flow function can ship at most twice the amount of flow that an optimal (standard) multiway flow function ships.*

We now define the *relaxed multiway cut* problem which is the dual problem of the relaxed multiway flow problem. Recall that the linear relaxation of the (standard) multiway cut problem assigns a length to each edge such that the distance between every pair of terminals is at least 1. In the relaxed multiway cut problem we assign to each edge $e \in E$ nonnegative length variables as follows: variable $\ell(e)$ corresponds to the intercommodity constraint on e, and variables $d^i(e)$, $1 \leq i \leq k$, correspond to the intracommodity constraints on e. Define the i-length of a path p to be $\sum_{e \in p}(d^i(e) + \ell(e))$. The linear program of the relaxed multiway cut problem is defined as follows:

$$\text{Minimize} \qquad \sum_{i=1}^{k} \sum_{e \in E} c_e \cdot d^i(e) \; + 2 \cdot \sum_{e \in E} c_e \cdot \ell(e)$$

subject to

$$\forall\, i, j \;(i \neq j),\ \forall \text{ path } p \text{ from } s_j \text{ to } s_i, \quad \sum_{e \in p} (d^i(e) + \ell(e)) \geq 1$$

$$\forall\, i, e, \quad d^i(e), \ell(e) \geq 0.$$

It is implicit that in an optimal solution, all variables are upper bounded by 1. The above linear program can be solved in polynomial time by the ellipsoid method [5]. This follows by observing that for a given solution, a shortest path computation between each pair of terminals can either determine that all constraints are satisfied or discover a violated constraint.

3. The approximation algorithm. We present in this section the approximation algorithm for the directed multiway cut problem. The algorithm derives a near-optimal multiway cut from an optimal relaxed multiway cut. The algorithm is summarized as follows.

Algorithm. DIRECTED MULTIWAY CUT

Input: A directed graph $G = (V, E)$ with special vertices $s_1, \ldots, s_k$;
Output: A multiway cut C of G;

1. Compute an optimal relaxed multiway cut.
2. For each terminal s_i, $1 \leq i \leq k$, compute a cut C_i.
3. $C \leftarrow C_1 \cup \cdots \cup C_k$.

We now elaborate on the steps of the algorithm. In the first step, an optimal relaxed multiway cut is computed. In the second step, for each terminal s_i, $1 \leq i \leq k$, define a cut C_i as follows. Let S_i be the set of vertices such that for all $v \in S_i$, there exists a path in G from s_i to v whose j-length is zero for all $1 \leq j \leq k$ and $j \neq i$. Then, $C_i = (S_i, V - S_i)$, i.e., the set of (directed) edges going from S_i to $V - S_i$ in G. The multiway cut C is defined to be the union of the cuts C_i, $1 \leq i \leq k$.

THEOREM 2. *The cut C is a multiway cut with respect to the terminals $s_1, \ldots, s_k$.*

Proof. Suppose, in contradiction, that terminal $s_j \in S_i$ for some $i \neq j$. This can happen only if there exists a path in G from s_i to s_j whose j-length is zero. This means that the solution computed in the first step of the algorithm does not satisfy the constraints of the relaxed multiway cut problem, yielding a contradiction. ∎

3.1. Analysis. We prove that the capacity of the multiway cut C computed by the algorithm is not greater than the value of an optimal relaxed multiway flow function. Lemma 1 states that the value of an optimal relaxed multiway flow function is at most twice the value of an optimal (standard) multiway flow function. By duality, the value of an optimal multiway cut is greater than or equal to the value of an optimal (standard) multiway flow function, yielding the desired approximation factor of 2.

THEOREM 3. *Let f be an optimal relaxed multiway flow function. Then, the capacity of the multiway cut C is not greater than the value of f.*

Proof. We decompose f into a set of flow paths connecting pairs of terminals. The idea of the proof is to charge the edges of C to flow paths and show that each flow path is charged at most once. By charging a flow path we mean that it "pays" the value of the flow it carries. We allow a set of flow paths to be charged by an edge e if and only if the sum of the flow they carry is at least as big as c_e. We denote by P_e the set of flow paths which edge $e \in C$ charges. We show that for any $e_1, e_2 \in C$, the intersection of P_{e_1} and P_{e_2} is empty.

The complementary slackness conditions for the relaxed multiway cut problem state that

- the length of each flow path is precisely 1;
- if, for edge e, $d^i(e) > 0$, then the corresponding intracommodity constraint with respect to commodity i is tight;
- if, for edge e, $\ell(e) > 0$, then the corresponding intercommodity constraint is tight.

We first define for each edge $e \in C$ the set of flow paths P_e to which it is charged. If e is an edge in C, then there exists a terminal s_i such that $e \in C_i$. Edge e must satisfy at least one of the following:

(1) There exists a commodity $j \neq i$ such that $d_e^j > 0$.

(2) The intercommodity constraint is tight, i.e., $\ell(e) > 0$.

Suppose that edge e satisfies (1). In this case, by complementary slackness, the intracommodity constraint for edge e with respect to commodity j is tight. We define

P_e to be the set of all flow paths that run between s_h, $1 \le h \le k$ and $h \ne j$, and s_j, and use edge e. By definition, the total amount of flow carried by the flow paths in P_e is equal to c_e. Suppose that edge e satisfies (2). In this case, by complementary slackness, the intercommodity constraint for edge e is tight. By definition,

$$\sum_j f_e^j = f^i(e) \; + \; \sum_{j \ne i} f^j(e).$$

The intracommodity constraint for edge e and commodity j implies that the first summand can be at most c_e. Therefore,

$$\sum_{j \ne i} f_e^j \ge c_e.$$

We define P_e to be the set of flow paths that contribute to the left-hand side. If edge e satisfies both (1) and (2), then we define P_e in accordance with (1).

It remains to show that for any $e_1, e_2 \in C$, the intersection of P_{e_1} and P_{e_2} is empty. Assume the contrary, that is, a flow path p that runs from s_x to s_y is charged by both $e_1 = (v_1, u_1)$ and $e_2 = (v_2, u_2)$, which appear on p in this order. Let s_{i_1} and s_{i_2} be terminals such that $e_1 \in C_{i_1}$ and $e_2 \in C_{i_2}$. (Note that i_1 and i_2 are not necessarily distinct.) The sets P_{e_1} and P_{e_2} are chosen such that $i_1 \ne y$ and $i_2 \ne y$.

By the definition of C_{i_2}, there exists a path from s_{i_2} to v_2 whose y-length is zero. By construction, either $\ell(e_1) > 0$ or $d_{e_1}^y > 0$. By complementary slackness, the y-length of p is precisely 1. Hence, there exists a path from s_{i_2} to s_y whose y-length is strictly smaller than 1, in contradiction to the requirements of a relaxed multiway cut. This completes the proof. □

4. Discussion. Our approximation algorithm derives a 2-approximate multiway cut by rounding an optimal relaxed multiway cut solution. We note that it is also possible to compute a 2-approximate multiway cut from an optimal relaxed multiway flow function. However, that would require computing an optimal relaxed multiway flow function f which has the following property. If, for edge e, some constraint is tight in f, then this constraint has to be tight in *every* optimal relaxed multiway flow function. Computing an optimal relaxed multiway flow function which has the latter property is computationally more expensive than computing an optimal relaxed multiway cut solution. Also, the analysis of a "flow-based" algorithm is slightly more complicated than the analysis of the "cut-based" algorithm. For more details regarding the flow-based algorithm, the reader is referred to [7].

A simple example shows that an approximation factor of 2 is the best factor that can be achieved by our algorithms. Garg, Vazirani, and Yannakakis [4] show that the integrality gap for the undirected multiway cut problem is 2. There exists a simple reduction from the undirected multiway cut problem to the directed multiway cut problem, which implies that the same gap holds for the directed multiway cut problem as well.

Acknowledgment. We thank Madhu Sudan for fruitful discussions.

REFERENCES

[1] G. Calinescu, H. Karloff, and Y. Rabani, *An improved approximation algorithm for multiway cut*, J. Comput. System Sci., 60 (2000), pp. 564–574.

[2] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multiterminal cuts*, SIAM J. Comput., 23 (1994), pp. 864–894.

[3] G. Even, J. (S.) Naor, and L. Zosin, *An* 8*-approximation algorithm for the subset feedback vertex set problem*, SIAM J. Comput., 30 (2000), pp. 1231–1252.

[4] N. Garg, V. V. Vazirani, and M. Yannakakis, *Multiway cuts in directed and node weighted graphs*, in Proceedings of the 21st International Colloquium on Automata, Languages, and Programming, 1994, Jerusalem, Israel, pp. 487–498.

[5] M. Grötschel, L. Lovász, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.

[6] D. R. Karger, P. N. Klein, C. Stein, M. Thorup, and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, in Proceedings of the 31st Annual Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 668–678.

[7] J. Naor and L. Zosin, *A* 2*-approximation algorithm for the directed multiway cut problem*, in Proceedings of the 38th IEEE Conference on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 548–553.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 483–495

THE GLOBAL POWER OF ADDITIONAL QUERIES TO *P*-RANDOM ORACLES*

WOLFGANG MERKLE†

Abstract. We consider separations of reducibilities by random sets. First, we show a result on polynomial time-bounded reducibilities that query their oracle nonadaptively: for every p-random set R, there is a set that is reducible to R with $k+1$ queries but is not reducible to any other p-random set with at most k queries. This result solves an open problem stated in a recent survey paper by Lutz and Mayordomo [*EATCS Bulletin*, 68 (1999), pp. 64–80]. Second, we show that the separation result above can be transferred from the setting of polynomial time-bounds to a setting of rec-random sets and recursive reducibilities. This extends the main result of Book, Lutz, and Martin [*Inform. and Comput.*, 120 (1995), pp. 49–54] who, by using different methods, showed a similar separation with respect to Martin-Löf-random sets. Moreover, in both settings we obtain similar separation results for truth-table versus bounded truth-table reducibility.

Key words. separation of reducibilities, random sets, resource-bounded measure, effective measure, resource-bounded reducibilities, effective reducibilities, bounded truth-table reducibility, truth-table reducibility

AMS subject classifications. 03D15, 03D30, 68Q15, 68Q30

PII. S0097539700366711

1. Introduction and related work.

We consider separations of reducibilities in the context of resource-bounded measure theory. In the following, we use the symbol $\leq$ with appropriate sub- or superscripts to denote binary relations on Cantor space, the class of all sets of natural numbers. These binary relations are meant as reducibilities and, in particular, we will consider polynomial time-bounded reducibilities of the following types: Turing (p-T), truth-table (p-tt), bounded truth-table (p-btt), and bounded truth-table restricted to at most k queries (p-btt(k)); see section 2 for more precise definitions. We say two reducibilities $\leq_r$ and $\leq_s$ are *separated by an oracle* A if the lower spans of A with respect to these reducibilities, i.e., the classes $\{X : X \leq_r A\}$ and $\{X : X \leq_s A\}$, differ. It is easy to see that two reducibilities are different (as binary relations on Cantor space) iff they are separated by some oracle. Beyond this simple observation, the question of which reducibilities are separated by what kind of oracles has been the object of intensive studies. Here, for a given pair of reducibilities, typical questions are the following. Are there separating oracles of low complexity? How comprising is the class of separating oracles? Which properties are sufficient for being a separating oracle?

Ladner, Lynch, and Selman [12] considered separations of the usual polynomial time-bounded reducibilities in the range between many-one- (p-m-) and p-T-reducibility. They showed that for every distinct pair of such reducibilities, there is a separating oracle that can be computed in exponential time. In their seminal paper [6], Bennett and Gill then obtained results about *separations by random oracles*, i.e., they showed that certain pairs of reducibilities are separated by almost all oracles in the sense that the class of separating oracles has measure 1 with respect to uniform measure on Cantor space. Subsequently, for any $k > 0$ it was shown for p-T-, p-

*Received by the editors February 4, 2000; accepted for publication (in revised form) January 30, 2001; published electronically September 26, 2001.
http://www.siam.org/journals/sicomp/31-2/36671.html
†Universität Heidelberg, Mathematisches Institut, Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany (merkle@math.uni-heidelberg.de).

tt-, p-btt-, p-btt$(k+1)$-, and p-btt(k)-reducibility that the two latter reducibilities are separated by almost all tally oracles [10] and that in fact every pair of distinct reducibilities chosen from this list can be separated by random oracles [16, 21].

A separation by random oracles can be expressed equivalently by saying that the class of oracles that do not separate the reducibilities under consideration has uniform measure 0. Lutz and Mayordomo [16] could show for certain pairs of polynomial time-bounded reducibilities of truth-table type that the class of nonseparating oracles does not just have uniform measure 0 but can in fact be covered by a martingale that is computable in polynomial time. Typically, their results are derived from the assumption that for both reducibilities the number of queries is bounded by a function in the input length and that the two bounding functions are related in a specific way, say, one is growing faster than the square of the other. In the special case where the bounding functions are constant, they showed that for every natural number k, there is a martingale computable in polynomial time that covers all oracles that do not separate p-btt$(k+1)$- and p-btt(k)-reducibility; hence, in particular, these reducibilities are separated by every p-random oracle. The latter can be rephrased by saying that these reducibilities are locally separated by the class of p-random oracles. Here, formally, a nonempty class $\mathbf{C}$ *locally separates* two given reducibilities if and only if for every set A in $\mathbf{C}$, the lower spans of A with respect to these reducibilities are different.

We say a class $\mathbf{C}$ *globally separates* two given reducibilities in case for every set A in $\mathbf{C}$ there is a set B that is reducible to A with respect to one of the given reducibilities but B is not reducible to any set in $\mathbf{C}$ with respect to the other reducibility. Moreover, in case such a set B exists not for all but just for some sets A in $\mathbf{C}$, we say that $\mathbf{C}$ yields a weak global separation of the reducibilities under consideration.

The definitions of the concepts of separations given above are meant for being applied with pairs of reducibilities $\leq_r$ and $\leq_s$ where $X \leq_r Y$ implies $X \leq_s Y$. In this situation, the definitions are no more symmetric in the sense that for example sets A and B witnessing a global separation by a class $\mathbf{C}$ must satisfy $B \leq_s A$ and $B \not\leq_r Z$ for all Z in $\mathbf{C}$. In distinguishing local and global separation we follow Book, Lutz, and Martin [8], who discuss such separations for the classes of Martin-Löf-random, tally, and sparse sets.

In the sequel we will consider global separations by various classes of random sets. Such investigations can be viewed as part of a more comprising research project where one asks which types of reductions are able to transform random objects into what types of far from random objects. For results in this direction and for further discussion and references see Juedes, Lathrop, and Lutz [11] as well as Lutz and Schweizer [18].

Remark 1.1. By definition, every local or global separation by a class $\mathbf{C}$ extends trivially to every nonempty subclass of $\mathbf{C}$. This is false in general, however, for weak global separations. For example given an oracle A that separates p-btt(2)- and p-btt(1)-reducibility, the class $\{A, \emptyset\}$, but not its subclass $\{\emptyset\}$, yields a weak global separation of these two reducibilities.

Remark 1.2. By definition, global separations always imply the corresponding local separation. A similar remark for weak global separations in place of global separations, however, is false. In order to obtain a counterexample, consider the class $\mathbf{C}$ that consist of all p-random sets plus a single set A that is computable in polynomial time. Then $\mathbf{C}$ does not locally separate p-btt(1)- and p-btt(2)-reducibility because A does not separate these reducibilities. On the other hand, $\mathbf{C}$ yields a weak

global separation of these reducibilities. The latter fact follows from Theorem 4.1 and the observation that none of the sets that witness a global separation by the class of p-random sets will be p-btt(1)-reducible to A.

In Theorem 4.1, we show that the class of p-random oracles yields a global separation of p-btt$(k+1)$- and p-btt(k)-reducibility. This, together with Remark 4.3, solves Problem 7 in a recent survey article by Lutz and Mayordomo [17], where it has been asked to prove or disprove that, in our terms, the class of p-random oracles yields a weak global separation of these reducibilities. In section 5, we obtain by basically the same proof as for Theorem 4.1 that for every natural number k and any rec-random set R, there is a set that is p-btt$(k+1)$-reducible to R but is not btt(k)-reducible to any rec-random set, where btt(k)-reductions are restricted to at most k nonadaptive queries and are computed by total Turing machines that might run in arbitrary time and space. Thus in particular, the class of rec-random sets globally separates btt$(k+1)$-reducibility from btt(k)-reducibility. By an easy argument similar to the one given in Remark 1.1, this yields as a special case the main result of Book, Lutz, and Martin [8], who showed, by using different methods, a corresponding separation result with respect to the class of Martin-Löf-random sets, which is a proper subclass of the class of rec-random sets. Moreover, we will argue that in both settings, i.e., for polynomial time-bounded, as well as for recursive reductions and martingales, the corresponding random sets globally separate the corresponding notions of truth-table and bounded truth-table reducibility.

2. Notation. The notation used in the following is mostly standard, for unexplained notation refer to the surveys and textbooks cited in the bibliography [4, 7, 15]. All strings are over the alphabet $\Sigma = \{0,1\}$. We identify strings with natural numbers via the isomorphism that takes the length-lexicographical ordering on $\{\lambda, 0, 1, 00, \dots\}$ to the usual ordering on ω, the set of natural numbers. If not explicitly stated differently, the terms *set* and *class* refer to sets of natural numbers and to sets of sets of natural numbers, respectively.

A *partial characteristic function* is a (total) function from some subset of the natural numbers to $\{0,1\}$. A partial characteristic function is *finite* iff its domain is finite. The restriction of a partial characteristic function β to a set I is denoted by $\beta|I$ and thus, in particular, for any set X, the partial characteristic function $X|I$ has domain I and agrees there with X. We identify strings of length n in the natural way with a partial characteristic function with domain $\{0,\dots,n-1\}$ and hence strings can be viewed as prefixes of sets. For a partial characteristic function α with domain $\{z_0 < \cdots < z_{n-1}\}$, the *string associated with* α is the (unique) string β with domain $\{0,\dots,n-1\}$ defined by $\beta(j) = \alpha(z_j)$. For a set X and a partial characteristic function σ we write $\langle X, \sigma\rangle$ for the set that agrees with σ for all arguments in the domain of σ and agrees with X, otherwise.

We will consider the following polynomial time-bounded reducibilities: p-T, p-tt, where the queries have to be asked nonadaptively, p-btt, where for each reduction the number of queries is bounded by a constant, and, even more restrictive, *p-btt(k)-reducibility*, where for all reductions this constant is bounded by the natural number k. The relation symbol $\leq^{\mathrm{p}}_{\mathrm{btt}}$ refers to p-btt-reducibility, and relation symbols for other reducibilities are defined in a similar fashion. Expressions such as *p-T-reduction* and $\leq^{\mathrm{p}}_{\mathrm{T}}$*-reduction* will be used interchangeably. We represent p-btt-reductions by a pair of functions g and h computable in polynomial time where $g(x)$ gives the set of strings queried on input x and $h(x)$ is a truth-table of a Boolean function over k variables that specifies how the answers to the queries in the set $g(x)$ are evaluated. Here we

assume, first, via introducing dummy variables, that the cardinality of $g(x)$ is always exactly k and, second, by convention, that for $i = 1, \dots, k$, the ith argument of the Boolean function $h(x)$ is assigned the ith query in $g(x)$ where the queries are ordered by length-lexicographical ordering. All reducibilities mentioned above can be defined by specifying an appropriate sequence of total oracle Turing machines that compute the corresponding reductions. For any total oracle Turing machine M there is a corresponding functional Γ where $\Gamma(B) = A$ iff M computes the set A on oracle B. Equivalently, the functional Γ can be viewed as a binary function from pairs of sets and strings to $\{0, 1\}$ such that $\Gamma(B, x) = 1$ iff M accepts x on oracle B.

Given a reducibility r, the *lower r-span of a set A* is the class $\{X : X \leq_r A\}$ of sets that are r-reducible to A, and the *lower r-span of a class $\boldsymbol{C}$* is the class of all sets that are r-reducible to some set in $\boldsymbol{C}$.

3. Resource-bounded measure. We give a brief introduction to resource-bounded measure, which focuses on the concepts that will be used in subsequent sections. For more comprehensive accounts of resource-bounded measure theory see the recent survey papers by Ambos-Spies and Mayordomo [4] and by Lutz [15].

The theory of resource-bounded measure is usually developed in terms of *martingales*, which can be viewed as payoff functions of gambles of the following type. A player successively places bets on the individual bits of the characteristic sequence of an unknown set A or, for short, the player bets on A. The betting proceeds in rounds $i = 1, 2, \dots$, where during round i, the player receives the length $i - 1$ prefix of A and then, first, decides whether to bet on the ith bit being 0 or 1 and, second, determines the stake by specifying the fraction of the current capital that shall be bet. Formally, a player can be identified with a *betting strategy* $b : \{0, 1\}^* \to [-1, 1]$ where the bet is placed on the next bit being 0 or 1 depending on whether $b(w)$ is negative or nonnegative, respectively, and where the absolute value of the real $b(w)$ is the fraction of the current capital that shall be at stake.

The player starts with strictly positive, finite capital. At the end of each round, in case the current guess has been correct, the capital is increased by this round's stake and, otherwise, is decreased by the same amount. So given a betting strategy b, we can inductively compute the corresponding *payoff function* d by applying the equations

$$d(w0) = d(w) - b(w) \cdot d(w), \qquad d(w1) = d(w) + b(w) \cdot d(w).$$

Intuitively speaking, the payoff $d(w)$ is the capital the player accumulates till the end of round $|w|$ by betting on a set that has the string w as a prefix. Conversely, any function d from strings to nonnegative reals that for all strings w satisfies the fairness condition

$$d(w) = \frac{d(w0) + d(w1)}{2} \tag{3.1}$$

induces canonically a betting function b, where

$$b(w) = \frac{d(w1) - d(w0)}{2} \cdot \frac{1}{d(w)}$$

in case $d(w)$ differs from 0 and $b(w) = 0$ otherwise. We call a function d from strings to nonnegative reals a martingale iff $d(\lambda) > 0$ and d satisfies the fairness condition (3.1) for all strings w.

By the preceding discussion it follows for gambles as described above that for any martingale there is an equivalent betting strategy and vice versa. We will frequently

identify martingales and betting strategies via this correspondence and, if appropriate, notation introduced for martingales will be extended to the induced betting strategies.

We say a martingale d *succeeds* on a set A if d is unbounded on the prefixes of A, i.e., if

$$\limsup_{n\in\omega} d(A|\{0,\dots,n\}) = \infty,$$

and d *succeeds* on or *covers* a class iff d succeeds on every set in the class.

It is easy to see that every countable class $\boldsymbol{C} = \{C_1, C_2, \dots\}$ is covered by the following betting strategy. On input w, let i be the minimal index such that w is a prefix of C_i, then bet half of the current capital on the next bit agreeing with the corresponding bit of C_i (and abstain from betting if such an index does not exist). As a consequence, most of the classes considered in complexity and recursion theory can be covered by martingales. In order to distinguish such classes in terms of coverability, one has to restrict the class of admissible martingales. Here, in general, for a given class $\boldsymbol{C}$ we want to specify a class of admissible martingales that allows the covering of interesting subclasses of $\boldsymbol{C}$, but not of $\boldsymbol{C}$ itself. In the context of recursion theory, this led to the consideration of recursive martingales [23, 24, 25, 27], whereas in connection with complexity classes one has to impose additional resource-bounds [1, 4, 14, 15, 20]. An effective martingale d is always confined to rational values and there is a Turing machine that on input w outputs some appropriate finite representation of $d(w)$.

Recall the definition of the *uniform measure* (or *Lebesgue measure*) on Cantor space, which describes the distribution obtained by choosing the individual bits of a set by independent tosses of a fair coin. It has been shown by Ville that a class has uniform measure 0 iff the class can be covered by some martingale [4, 26]. The latter result justifies the following notation: a class has *measure* 0 with respect to a given class of martingales iff it is covered by some martingale in the class. The aim stated above can then be rephrased: for a given class $\boldsymbol{C}$, we want to specify a class of admissible martingales such that interesting subclasses of $\boldsymbol{C}$ have measure 0, but not $\boldsymbol{C}$ itself.

In connection with measure on complexity classes, most attention has been received by measure concepts for the exponential time-bounded classes

$$\mathbf{E} = \mathbf{DTIME}(2^{\mathrm{lin}}) \quad \text{and} \quad \mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}}).$$

For example, in the case of the class $\mathbf{E}$, Lutz proposed to use martingales that on input w are computable in time polynomial in the length of w. Observe that the latter time-bound yields the same class of martingales as the time-bound $2^{\mathrm{O}(|x|)}$ where x is the minimal string not in the domain of w; i.e., if w is viewed as prefix of a set A, then x is the minimal string y such that $A(y)$ is not encoded in w. Lutz could show that for every constant c, the subclass $\mathbf{DTIME}(2^{c\cdot n})$ of $\mathbf{E}$ can be covered by such a martingale, but not $\mathbf{E}$ itself. The class of polynomial time-bounds used to define measure on $\mathbf{E}$ is so robust that, similar to the case of unrestricted martingales, martingales and betting strategies that can be computed in polynomial time are essentially equivalent. (However, in general, a fixed polynomial bound on the running time might not be preserved in the transition from a betting strategy to the corresponding martingale [3].) Furthermore, there is a similar correspondence between martingales and betting strategies in the case of martingales used to define measure on $\mathbf{EXP}$ [3] and in the case of recursive martingales [24].

We say a set is *p-random* if the set cannot be covered by a martingale that is computable in polynomial time, and we write p-RAND for the class of all p-random

sets. The notion *rec-random set* and the class rec-RAND of all rec-random sets are defined likewise with recursive martingales in place of martingales that are computable in polynomial time. Moreover, we will consider Martin-Löf-random sets [19]. These have been characterized in terms of martingales by Schnorr [24]. A set is *Martin-Löf-random* if and only if it cannot be covered by a subcomputable martingale. A martingale d is *subcomputable* iff there is a recursive function g in two arguments such that for all strings w, the sequence $g(w,0), g(w,1), \dots$ is nondecreasing and converges to $d(w)$.

The classes of p-random, rec-random, and Martin-Löf random sets all have uniform measure 1 because the class of sets on which a single martingale succeeds always has uniform measure 0 and, by σ-additivity, the same holds for every countable union of such classes. By definition, any rec-random set is p-random but the reverse implication is false as one can construct a recursive p-random set by diagonalizing against an appropriate weighted sum of all p-betting strategies. By the characterization of Martin-Löf-random sets stated above, it is immediate that the class of Martin-Löf-random sets is a subclass of rec-RAND. Schnorr [24] has implicitly shown that this containment is proper. For a proof, it suffices to recall that the prefixes of a Martin-Löf-random set can not be compressed by more than a constant while a corresponding statement for rec-random sets is false [13, Theorem 3.6.1 and Exercise 2.5.13].

We conclude this section by two remarks in which we describe standard techniques for the construction of martingales.

Remark 3.1. Let a finite set D be given, as well as a list $\langle D_1, \dots, D_m\rangle$ of pairwise disjoint subsets of D that all have the same cardinality $k > 0$. Then for a partial characteristic function σ with domain D and a string w of length k we might ask for the frequency

$$\alpha(\sigma, w, \langle D_1, \dots, D_m\rangle) \quad := \quad \frac{|\{j : w \text{ is the associated string of } \sigma|D_j\}|}{m}$$

with which w occurs in σ as associated string at the positions specified by the D_i. In case the sets D_i are clear from the context, we suppress mentioning them and we write $\alpha(\sigma, w)$ for short.

If we choose the bits of σ by independent tosses of a fair coin, then for every w of length k, the expected value of $\alpha(\sigma, w)$ is $1/2^k$. For large m, only for a small fraction of all partial characteristic functions with domain D will the frequency of w deviate significantly from the expected value, as can, for example, be shown by using Chernoff bounds [22, Lemma 11.9]. By using such bounds it is indeed straightforward to show that given k and a rational $\varepsilon > 0$, we can compute a natural number $m(k, \varepsilon)$ such that for all $m \geq m(k, \varepsilon)$ and for all D and $D_1, \dots, D_m$ as above we have

$$\frac{|\{\sigma : D \to \{0,1\} \ : \ \frac{1}{2}\cdot\frac{1}{2^k} < \alpha(\sigma, w, \langle D_1, \dots, D_m\rangle) < \frac{3}{2}\cdot\frac{1}{2^k}\}|}{2^{|D|}} \quad \geq \quad 1 - \varepsilon. \tag{3.2}$$

Remark 3.2. Let I be a finite set and let Θ be a subset of all partial characteristic functions with domain I. We can easily construct a martingale that by betting on places in I, increases its capital by a factor of $2^{|I|}/|\Theta|$ for all sets B where $B|I$ is in Θ. Here the martingale takes the capital available when betting on the minimal element of I and distributes it evenly among the elements of Θ, then computing values upwards according to the fairness condition for martingales.

4. Separations by p-random oracles. Lutz and Mayordomo [16] have shown that for any p-random set R, the lower p-btt(k)-span of R is strictly contained in the

lower p-btt$(k+1)$-span of R, i.e., the class p-RAND yields a local separation of these reducibilities. In Theorem 4.1, we extend this local separation to a global separation, i.e., we show that for any p-random set R there is a set that is p-btt$(k+1)$-reducible to R but is not p-btt(k)-reducible to any p-random set.

THEOREM 4.1. *Let R be a p-random set and let k be a natural number. Then the lower p-btt$(k+1)$-span of R is not contained in the lower p-btt(k)-span of* p-RAND.

Proof. In order to define a set A and a p-btt$(k+1)$-reduction (g_0, h_0) from A to R, we let $h_0(x)$ be the truth-table of the $(k+1)$-place conjunction and we let

$$g_0(x) := \{x0^1 1^{k+1}, x0^2 1^k, \ldots, x0^{k+1}1^1\}, \qquad A := \{x : g_0(x) \subseteq R\}. \tag{4.1}$$

We are done if we can show that if A is p-btt(k)-reducible to a set, then this set cannot be p-random. So let B be an arbitrary set and assume that A is reducible to B via the p-btt(k)-reduction (g, h). We will construct a martingale d that is computable in polynomial time and succeeds on B. To this end, let $m(.,.)$ be the function defined in Remark 3.1 and define a sequence $n_0, n_1, \ldots$ with

$$n_0 = 0, \qquad n_{i+1} > 2^{n_i}, \qquad \log n_{i+1} > m\left(k+1, \frac{1}{2^{i+1}}\right) \tag{4.2}$$

such that given x of length n, we can compute in time $\mathrm{O}(n^2)$ the maximal i with $n_i \leq n$. Such a sequence can be obtained by standard methods as described in the chapter on uniform diagonalization and gap languages in Balcázar, Díaz, and Gabarró [7]. For example, we can first define a sufficiently fast growing time-constructible function $r : \omega \to \omega$ and then let n_i be the value of the i-fold iteration of r applied to 0.

It is helpful to view the betting strategy of the martingale d as being performed in stages $i = 0, 1, \ldots$ where the bets of stage i depend on the g-images of the strings of length n_i. While considering the queries made for strings of length n_i with $i > 0$, we will distinguish short queries with length strictly less than

$$l_i := \left\lfloor \frac{n_i}{2k} \right\rfloor \tag{4.3}$$

and long queries, i.e., queries of length at least l_i. We call two strings x and y equivalent iff, for some i, both have identical length n_i and in addition we have

$$h(x) = h(y) \quad \text{and} \quad g(x) \cap \{z : |z| < l_i\} = g(y) \cap \{z : |z| < l_i\}\ , \tag{4.4}$$

i.e., two strings of length n_i are equivalent iff they have the same truth-table and the same set of short queries. Then for some constant c, the number of equivalence classes of strings of length n_i can be bounded from above by

$$2^{2^k} \sum_{j=0}^{k} \binom{2^{l_i} - 1}{j} \leq 2^{2^k}(k+1) \cdot 2^{l_i \cdot k} \leq c \cdot 2^{\frac{n_i}{2k} \cdot k} = c \cdot 2^{\frac{n_i}{2}}.$$

So the 2^{n_i} strings of length n_i are partitioned into at most $c \cdot 2^{\frac{n_i}{2}}$ equivalence classes, hence there is i_0 such that for all $i > i_0$, there is an equivalence class of cardinality at least $m_i := \lfloor \log n_i \rfloor$. For all such i, among all equivalence classes of strings of length n_i we choose one with maximal cardinality (breaking ties by some easily computable but

otherwise arbitrary rule), we let J_i contain the first m_i strings in this equivalence class, and we let

$$\alpha_i \quad = \quad \frac{|A \cap J_i|}{|J_i|}. \tag{4.5}$$

We show now that due to R being p-random, almost all α_i are close to $1/2^{k+1}$.

CLAIM 1. *For almost all i, α_i is contained in the open interval K defined by*

$$K \ := \ \left(\frac{1}{2} \cdot \frac{1}{2^{k+1}}, \ \frac{3}{2} \cdot \frac{1}{2^{k+1}}\right). \tag{4.6}$$

Proof. Fix any index $i > i_0$. Let $z_1 < \cdots < z_{m_i}$ be the elements of J_i, let D_j be equal to $g_0(z_j)$ for $j = \{1, \ldots, m_i\}$, and let D be the union of D_1 through D_{m_i}. Recall from Remark 3.1 the definition of the function α. For any partial characteristic function σ with domain D and any string w of length $k+1$, $\alpha(\sigma, w)$ is equal to the fraction of all indices i among $1, \ldots, m_i$ such that the string associated with $\sigma|D_i$ is equal to w. Now the truth table h_0 is just the conjunction of the queries given by g_0 and thus by construction, z_i is in A iff all strings in D_i are in R. Hence by the definitions of α_i and α, we obtain $\alpha_i = \alpha(\sigma_i, v)$ where $v = 1^{k+1}$ and σ_i is the restriction of R to places in D.

On the other hand, by choice of the m_i and by (4.2), we know that $m_i = \lfloor \log n_i \rfloor$ is larger than $m(k+1, 1/2^i)$. By definition of the function m in Remark 3.1, it is then immediate that for all but a $1/2^i$-fraction of all partial characteristic functions σ with domain D the value $\alpha(\sigma, v)$ is in K. If α_i, and hence also $\alpha(\sigma_i, v)$, is not in K, then $\sigma_i = R|D$ belongs to this exceptional fraction. Remark 3.2 shows that in this situation, by betting on the places in D, a martingale can increase its capital by a factor of 2^i when betting against the unknown set R.

Now consider the following martingale, where we leave it to the reader to show that the martingale can be computed in polynomial time. The initial capital 1 is split into infinitely many parts $c_1, c_2, \ldots$ where $c_i = 1/2^i$ is exclusively used to place bets on the strings in the set D that corresponds to the index i, i.e., the strings that are in $g_0(x)$ for some x in J_i. By the preceding discussion, the martingale can increase the capital c_i to at least 1 for all $i > i_0$ such that α_i is not in K. But if this were the case for infinitely many values of i, the martingale would succeed on R, thus contradicting the assumption that R is p-random. □

By Claim 1 for almost all i, the density of the set A on J_i is confined to the small interval K with center $1/2^{k+1}$. While constructing the martingale d that is meant to succeed on B, we will exploit that thus in particular for almost all i, this density differs from 0 and is less than

$$\rho \ = \ \frac{3}{2} \cdot \frac{1}{2^{k+1}} \ . \tag{4.7}$$

Recall from the introduction that one can view reductions as functionals and let Γ be the functional that corresponds to the btt(k)-reduction given by (g, h), hence for example A is equal to $\Gamma(B)$. For all $i \geq i_0$, let

$$H_i \quad = \quad \bigcup_{x \in J_i} \{z : z \text{ in } g(x) \text{ and } |z| \geq l_i\},$$

i.e., H_i is the set of all long queries made by strings in J_i. Then we can argue that only for a fraction of all partial characteristic functions σ with domain H_i the

set $\Gamma(\langle B, \sigma\rangle)$ has density less than ρ on J_i. Formally, for every $i > i_0$ and for every partial characteristic function σ with domain H_i, we let

$$\beta_i(\sigma) \;=\; \frac{|\Gamma(\langle B, \sigma\rangle) \cap J_i|}{|J_i|}$$

and, further,

$$\Theta_i \;=\; \{\sigma : \sigma \text{ partial characteristic function with domain } H_i \text{ and } \beta_i(\sigma) < \rho\}.$$

By Claim 1 for almost all i, the density α_i of the set $A = \Gamma(B)$ on J_i is less than ρ, hence the restriction of B to H_i must be contained in Θ_i by definition of Θ_i.

We will argue next that there is some $\delta < 1$ such that for almost all i, the set Θ_i comprises at most a δ-fraction of all partial characteristic functions with domain H_i. We will then exploit the latter fact in the construction of the martingale d by betting against the $(1 - \delta)$-fraction of partial characteristic functions outside of Θ_i, which have already been ruled out as possible restriction of B to H_i.

For the moment, let τ_x be the Boolean function obtained from $h(x)$ by hard-wiring $B(z)$ into $h(x)$ for all short queries z in $g(x)$. By definition for all x, the queries in $g(x)$ are assigned to the variables of $h(x)$ in length-lexicographical order, hence for equivalent strings x and y, the Boolean functions τ_x and τ_y are identical. Thus for every $i > i_0$, all strings in J_i are mapped to the same Boolean function, which we denote by τ_i. We call a Boolean function constant iff it evaluates to the same truth value for all assignments to its arguments (and hence in particular all 0-place Boolean functions are constant).

Claim 2. *For almost all i, τ_i is not constant.*

Proof. If τ_i is constant, then the value $A(x)$ must be the same for all x in J_i. But then α_i is either 0 or 1, while Claim 1 implies that this is the case for at most finitely many i. □

Claim 3. *There is a constant $\delta < 1$ such that for almost all i, the set Θ_i comprises at most a δ-fraction of all partial characteristic functions with domain H_i.*

Proof. For a given $i > i_0$ such that τ_i is not constant, consider the random experiment where we use independent tosses of a fair coin in order to choose the individual bits of a random partial characteristic function $\widehat{\sigma}$ with domain H_i. Then all partial characteristic functions of this type occur with the same probability; hence the fraction we want to bound is just the probability of picking an element in Θ_i.

For every string x in J_i, define a 0-1-valued random variable b_x, and define a random variable γ_i with rational values in the closed interval $[0, 1]$ by

$$b_x(\widehat{\sigma}) := \Gamma(\langle B, \widehat{\sigma}\rangle, x), \qquad\qquad \gamma_i(\widehat{\sigma}) := \frac{1}{|J_i|} \sum_{x \in J_i} b_x(\widehat{\sigma}).$$

Consider an arbitrary string x in J_i. By assumption, τ_i is not constant, hence there is at least one assignment to $\widehat{\sigma}$ such that $b_x(\widehat{\sigma})$ is 1. Moreover such an assignment occurs with probability at least $1/2^k$ because $h(x)$, and thus also τ_i, has at most k variables. Thus the expected value of b_x is at least $1/2^k$ and by linearity of expectation we obtain

$$\mathbf{E}(\gamma_i) \;=\; \frac{1}{|J_i|} \sum_{x \in J_i} \mathbf{E}(b_x) \;\geq\; \frac{1}{|J_i|} \sum_{x \in J_i} \frac{1}{2^k} \;=\; \frac{1}{2^k}. \tag{4.8}$$

If we let p be the probability of the event "$\gamma_i < \rho$," we have

$$\frac{1}{2^k} \leq \mathbf{E}(\gamma_i) \leq p \cdot \rho + (1-p) \cdot 1 \leq \rho + (1-p) = \frac{3}{4} \cdot \frac{1}{2^k} + (1-p), \tag{4.9}$$

where the relations follow, from left to right, by (4.8), by definition of p and by $\gamma_i \leq 1$, because the probability p is bounded by 1, and by the choice of ρ in (4.7). By comparing the first and last term in (4.9) we then obtain that p is bounded from above by $\delta := 1 - 1/2^{k+2}$. □

For all i, let $I_i = \{x : l_i \leq |x| < l_{i+1}\}$. The values of n_i grow sufficiently fast such that for some i_1 and for all $i > i_1$, the set H_i is contained in I_i. Moreover, by Claim 3, for some $i_2 > i_0$ and all $i > i_2$, there is a set Θ_i of partial characteristic functions with domain H_i where Θ_i contains only a δ-fraction of all such partial characteristic functions and contains the restriction of B to H_i. Let i_3 be the maximum of i_1 and i_2.

Now we are in a position to describe a betting strategy that succeeds on B. For a given input w, let x be the $(|w|+1)$th string, i.e., the string on which we might bet. We first compute the index i such that x is in I_i, together with the corresponding set H_i. In case $i \leq i_3$ or if x is not in H_i, we abstain from betting. Otherwise, we place a bet on x according to the betting strategy as described in Remark 3.2, which, while placing bets on the strings in H_i, increases the capital by a factor of at least $1/\delta$ by betting against the partial characteristic functions that are not in Θ_i. Here all necessary computations can be performed in time $2^{\mathrm{O}(n_i)}$ and hence, by $|x| \geq l_i = \lfloor n_i/2k \rfloor$, in time $2^{\mathrm{O}(|x|)}$. It follows that this betting strategy induces a martingale computable in polynomial time that on interval I_i preserves its capital in case $i \leq i_3$ and increases its capital by a factor of at least $1/\delta$ for all $i > i_3$.

This finishes the proof of Theorem 4.1. Observe that the current proof would for example also go through if we had chosen the cardinality m_i of J_i to be equal to n_i. The actual choice of the m_i emphasizes that for given k, the complexity of the martingale d covering the set B in the upper p-btt(k)-span of A is dominated by the complexity of computing the sets J_i, whereas the complexity of handling the assignments on the sets H_i can be neglected. □

Remark 4.2. The assertion of Theorem 4.1 remains valid if we simply require the set R to be n-random instead of p-random, (i.e., if we require that there is no martingale computable in time $\mathrm{O}(n)$ that succeeds on R). For a proof, note that Ambos-Spies, Terwijn, and Zheng [5] have shown that for every n^2-random set R, there is a p-random set R_0 that is p-m-reducible to R while, in fact, the latter assertion is true for n-random R. Now the relaxed version of Theorem 4.1 follows because the existence of a separating set A as required in the theorem extends directly from R_0 to R.

Remark 4.3. Theorem 4.1 states that the lower p-btt($k+1$)-span of every p-random set R contains a set A that is not in the lower p-btt(k)-span of any p-random set. As already noted by Book, Lutz, and Martin [8], for a set R that is not just p-random but is even Martin-Löf-random, such a set A cannot be recursive. This follows from the fact that every recursive sets that is p-btt($k+1$)-reducible to a Martin-Löf-random set is in fact computable in polynomial time. The latter fact is attributed to folklore by Lutz and Schweizer [18] and can be obtained as a special case of a result of Book, Lutz, and Wagner [9]. They have shown from rather general assumptions on the reducibility under consideration that every recursive set that is reducible to a Martin-Löf-random set must be contained in the corresponding almost-class, i.e., in the class of sets that have an upper span of uniform measure 1. Their assumptions are satisfied for most bounded reducibilities considered in the literature

and, in particular, their result applies to p-btt(k)-reducibility for all $k \geq 0$. Moreover, for the latter reducibilities it was shown by Ambos-Spies [2] that the corresponding almost-classes are all equal to the class of sets computable in polynomial time. As a consequence, any recursive set A in the lower p-btt($k+1$)-span of a Martin-Löf-random set is computable in polynomial time and is hence in the lower p-btt(k)-span of every Martin-Löf-random set.

From the proof of Theorem 4.1 we obtain the following corollary.

COROLLARY 4.4. *For every p-random set R, the lower p-tt-span of R is not contained in the lower p-btt span of* p-RAND.

Proof. For a given p-random set R and for every k, let the set A_{k+1} be defined in the same way as the set A has been defined in (4.1) in the proof of Theorem 4.1. Then A_{k+1} is p-btt($k+1$)-reducible to R, but is not p-btt(k)-reducible to any p-random set. Moreover, the set

$$B \quad = \quad \{x : x = 1^k 0y \text{ and } y \text{ in } A_k\}$$

is p-tt-reducible to R by construction of the sets A_k. On the other hand, if B were p-btt-reducible to some p-random set R_0, then B would be in fact p-btt(k)-reducible to R_0 for some k. Hence, in particular, A_{k+1} were p-btt(k)-reducible to R_0, thus contradicting the choice of A_{k+1}. □

5. Separations by rec-random oracles. Lutz [14] showed that recursive martingales yield a reasonable measure concept for the class of recursive sets, where in particular the class of all recursive sets cannot be covered by a recursive martingale.[1] Recall from the introduction that a set is recursively random iff it cannot be covered by a recursive martingale and that rec-RAND denotes the class of all such sets. Next we state two results on recursively random sets that correspond rather closely to Theorem 4.1 and Corollary 4.4 on p-random sets.

THEOREM 5.1. *Let the set R be in* rec-RAND *and let k be a natural number. Then the lower* p-($k+1$)-tt*-span of R is not contained in the lower btt(k)-span of* rec-RAND.

COROLLARY 5.2. *For every set R in* rec-RAND*, the lower p-tt-span of R is not contained in the lower btt-span of* rec-RAND.

In connection with Theorem 5.1 and Corollary 5.2, recall that *btt-reducibility* is defined like p-btt-reducibility, except that a btt-reduction is required to be computed by a total Turing machine that might run in arbitrary time and space and that *btt(k)-reducibility* is the restriction of btt-reducibility where the number of queries is bounded by k.

We omit the proofs of Theorem 5.1 and Corollary 5.2, which are almost literally the same as in the case of p-random sets. Besides the fact that now we consider effective martingales and reductions instead of polynomial time-bounded ones, the main difference is that for arbitrary recursive reductions from A to B we cannot compute an a priori bound on the size of the queries. Hence in order to ensure that the sets H_i are pairwise disjoint, the definition of the n_i will now depend on the given reduction (g, h). Here we choose n_{i+1} so large that all strings queried on inputs of length n_i are short queries with respect to n_{i+1}, i.e., have length strictly less than $\lfloor n_{i+1}/2k \rfloor$.

Remark 5.3. Recall from the discussion preceding Remark 3.1 that the class of Martin-Löf-random sets is a proper subclass of rec-RAND. As a consequence,

[1] For further discussion of measure concepts for the class of recursive sets see for example Schnorr [24], Terwijn [25], and Wang [27].

by an easy argument similar to the one used in Remark 1.1, from the separation by the class rec-RAND stated in Theorem 5.1 we obtain the main result of Book, Lutz, and Martin [8], who showed that for all k, the lower p-btt$(k+1)$-span of a Martin-Löf-random set is never contained in the lower btt(k)-span of the class of all Martin-Löf-random sets.

Acknowledgments. We are grateful to Elvira Mayordomo for several useful corrections and suggestions, where the latter include a significant simplification of the argument in Remark 4.2 via referring to the cited result of Ambos-Spies, Terwijn, and Zheng [5]. Furthermore, we would like to thank Klaus Ambos-Spies, Jack Lutz, and Jan Reimann for helpful discussions. Finally, we are grateful for the detailed comments of the anonymous referees of ICALP 2000 and of the *SIAM Journal on Computing*.

REFERENCES

[1] E. Allender and M. Strauss, *Measure on small complexity classes, with applications for BPP*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 807–818.

[2] K. Ambos-Spies, *Randomness, relativizations, and polynomial reducibilities*, in Proceedings First Structure in Complexity Theory Conference, Lecture Notes in Comput. Sci. 223, Springer-Verlag, 1986, pp. 23–34.

[3] K. Ambos-Spies, E. Mayordomo, Y. Wang, and X. Zheng, *Resource-bounded balanced genericity, stochasticity and weak randomness*, in the 13th Annual Symposium on Theoretical Aspects of Computer Science, C. Puech and R. Reischuk, eds., Lecture Notes in Comput. Sci. 1046, Springer-Verlag, New York, 1996, pp. 63–74.

[4] K. Ambos-Spies and E. Mayordomo, *Resource-bounded measure and randomness*, in Complexity, Logic, and Recursion Theory, A. Sorbi, ed., Dekker, New York, 1997, pp. 1–47.

[5] K. Ambos-Spies, S. A. Terwijn, and X. Zheng, *Resource bounded randomness and weakly complete problems*, Theoret. Comput. Sci., 172 (1997), pp. 195–207.

[6] C. H. Bennett and J. Gill, *Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[7] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity,* Vol. I, Springer-Verlag, Berlin, 1995.

[8] R. V. Book, J. H. Lutz, and D. M. Martin, Jr., *The global power of additional queries to random oracles*, Informat. Comput., 120 (1995), pp. 49–54.

[9] R. V. Book, J. H. Lutz, and K. W. Wagner, *An observation on probability versus randomness with applications to complexity classes*, Math. Systems Theory, 27 (1994), pp. 201–209.

[10] R. V. Book and S. Tang, *Polynomial-time reducibilities and "almost all" oracle sets*, Theoret. Comput. Sci., 81 (1991), pp. 35–47.

[11] D. W. Juedes, J. I. Lathrop, and J. H. Lutz, *Computable depth and reducibility*, Theoret. Comput. Sci., 132 (1994), pp. 37–70.

[12] R. E. Ladner, N. A. Lynch, and A. L. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[13] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, New York, 1997.

[14] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[15] J. H. Lutz, *The quantitative structure of exponential time*, in Complexity Theory Retrospective II, L. A. Hemaspaandra and A. L. Selman, eds., Springer-Verlag, New York, 1997, pp. 225–260.

[16] J. H. Lutz and E. Mayordomo, *Cook versus Karp-Levin: Separating completeness notions if NP is not small*, Theoret. Comput. Sci., 164 (1996), pp. 141–163.

[17] J. H. Lutz and E. Mayordomo, *Twelve problems in resource-bounded measure*, Bull. Euro. Assoc. Theoret. Comput. Sci., 68 (1999), pp. 64–80.

[18] J. H. Lutz and D. L. Schweizer, *Feasible reductions to Kolmogorov–Loveland stochastic sequences*, Theoret. Comput. Sci., 225 (1999), pp. 185–194.

[19] P. Martin-Löf, *The definition of random sequences*, Informat. Control, 9 (1966), pp. 602–619.

[20] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Doctoral dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.
[21] W. Merkle and Y. Wang, *Random separations and "almost" classes for generalized reducibilities*, Math. Logic Quart., 47 (2001), pp. 249–269.
[22] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[23] C.-P. Schnorr, *A unified approach to the definition of random sequences*, Math. Systems Theory, 5 (1971), pp. 246–258.
[24] C.-P. Schnorr, *Zufälligkeit und Wahrscheinlichkeit*, Lecture Notes in Math. 218, Springer-Verlag, Berlin, 1971.
[25] S. A. Terwijn, *Computability and Measure*, Doctoral dissertation, Universiteit van Amsterdam, Amsterdam, Netherlands, 1998.
[26] J. Ville, *Étude Critique de la Notion de Collectif*, Gauthiers-Villars, Paris, 1939.
[27] Y. Wang, *Randomness and Complexity*, Doctoral dissertation, Universität Heidelberg, Mathematische Fakultät, INF 288, Heidelberg, Germany, 1996.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 496–526

GEOMETRIC COMPLEXITY THEORY I: AN APPROACH TO THE *P* VS. *NP* AND RELATED PROBLEMS*

KETAN D. MULMULEY† AND MILIND SOHONI‡

Abstract. We suggest an approach based on geometric invariant theory to the fundamental lower bound problems in complexity theory concerning formula and circuit size. Specifically, we introduce the notion of a *partially stable* point in a reductive-group representation, which generalizes the notion of stability in geometric invariant theory due to Mumford [*Geometric Invariant Theory,* Springer-Verlag, Berlin, 1965]. Then we reduce fundamental lower bound problems in complexity theory to problems concerning infinitesimal neighborhoods of the orbits of partially stable points. We also suggest an approach to tackle the latter class of problems via construction of *explicit obstructions.*

Key words. geometric invariant theory, computational complexity, algebraic geometry, representation theory, stability

AMS subject classifications. 20G05, 03D15, 70G55

PII. S009753970038715X

1. Introduction.

It is generally agreed by now [41] that the problems in complexity theory such as P vs. NP [10, 25, 20] and the related lower bound problems for formula or arithmetic-circuit size are of fundamental importance and may have connections with deep issues in other areas of mathematics. In this article we suggest an approach based on geometric invariant theory [37, 38] to several such lower bound problems concerning formula and circuit size. We shall focus on formulae and circuits over integers or algebraically closed fields of arbitrary characteristic before addressing the ones over finite fields. Our work is based on the classical geometric invariant theory developed by Hilbert [19], Weyl [46], and others, as well as on the modern theory developed by Mumford, Fogarty, and Kirwan [37], Nagata [31], Seshadri [40], Haboush [17], Luna [27], Kempf [23], and others. Especially central in this paper is the powerful notion of stability in geometric invariant theory [19, 37, 23, 27, 40].

For a survey of earlier algebraic and geometric techniques in lower bound problems, see [5]; for a survey of earlier combinatorial techniques, see [3]. A starting point for our investigation was a result by one of the authors [33] that provided a concrete support for the $P \neq NC$ conjecture by an unconditional proof of its weaker implication in a restricted but natural and realistic *PRAM model without bit operations*, which contains virtually all known fast parallel algorithms for algebraic and weighted-combinatorial-optimization problems. In [33] it is shown that well-known network flow problems such as min-cost flow and max-flow, which can be solved efficiently without bit operations sequentially—i.e., they have the so-called strongly polynomial time algorithms [16]—cannot be solved efficiently in this model. Specifically, consider the max-flow problem. Since it is P-complete, the $P \neq NC$ conjecture implies that it cannot be solved in the usual PRAM model in $O(\text{polylog}(N))$ time using $\text{poly}(N)$

*Received by the editors July 19, 2000; accepted for publication (in revised form) March 27, 2001; published electronically September 26, 2001.

http://www.siam.org/journals/sicomp/31-2/38715.html

†Department of Computer Science, Ryerson Laboratory, University of Chicago, 1100 E. 58th Street, Chicago, IL 60637 (mulmuley@cs.uchicago.edu). The work of this author was supported by the Guggenheim Fellowship and NSF grant CCR 9800042.

‡Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, Powai, Mumbai 400076, India (sohoni@cse.iitb.ernet.in).

processors, where N is the total bitlength of the input. Since the PRAM model without bit operations is just a restricted version of the usual PRAM model, obtained by not allowing the bit operations in the usual model, it follows from the $P \neq NC$ conjecture that the max-flow problem cannot be solved in the PRAM model without bit operations as well in $O(\text{polylog}(N))$ time using $\text{poly}(N)$ processors. In [33], this weaker implication of the $P \neq NC$ conjecture is proved unconditionally using some algebraic geometry. (In fact, a stronger lower bound is proved.)

This was a turning point that provided a compelling argument for believing that algebraic geometry will play a crucial role in the P vs. NC and other separation problems in complexity theory. The approaches attempted earlier (cf., e.g., [3]) were combinatorial, but they had a serious natural-proof limitation as pointed out in [39]. The present approach does not have this limitation, since it is based on *explicit constructions* (cf. section 5 and Part II [34]) and the results, such as the Hilbert–Mumford–Kempf criterion [37, 23] for stability (cf. section 3), which work only for points with specific types of stabilizers. Such points are very rare because almost all points in the representations under consideration have trivial stabilizers.[1] (Loosely speaking, according to the terminology in [39], the probabilistic proof for existence of expanders would be natural because it ends up proving that a significant fraction of graphs are expanders, whereas the proof based on explicit construction [26, 29] would be unnatural because it works for only a specific graph.) A closer analysis of the limitation of the techniques in [33] in the context of the usual P vs. NC problem suggested that geometric invariant theory may help. This led us to explore the role of geometric invariant theory in the fundamental lower bound problems of complexity theory.

Geometric invariant theory is the study of the action of an algebraic reductive group on an algebraic variety. One important problem here is to understand the structure of this action in the vicinity of the orbit of any fixed point. If the variety is affine and smooth, the characteristic is zero, and the point is stable in the sense of Mumford [36] and Kempf [23], i.e., its orbit is closed, then Luna's étale slice theorem (cf. [27] and the appendix of [37]), based on the earlier work on stability of Mumford [36] and others, says that the orbit has a neighborhood which looks like a certain homogeneous fiber bundle. This gives a complete understanding of the structure of the action in the vicinity of the orbit of a stable point. The action in the vicinity of the null cone, i.e., the set of points in V on which all nonconstant homogeneous G-invariant polynomial functions on V vanish, is, however, not well understood, though there are some results concerning the structure of the null cone, e.g., the Hilbert–Mumford criterion [37] and Luna's stratification [38]. The reason is that the action in the vicinity of the orbit of an arbitrary point of the null cone may be too complex. However, the situation may be better if the point under consideration has some weaker form of stability. We define one such notion, called partial stability, that is also applicable to the points in the null cone. Then we show that an in-depth understanding of the structure of a reductive-group action in the vicinity of the orbit of a partially stable point would lead to resolution of the lower bound problems mentioned above. We also suggest an approach, which will be developed in depth in Part II [34], for studying the group action in the vicinity of the orbit of a partially stable point in the null cone. Such a study is of independent mathematical interest in geometric invariant theory.

[1] For example, the proof of Theorem 4.1 works only for points that are stabilized by the stabilizer R of the determinant. However, in the space $P(V)$ under consideration, the determinant is the only point that is stabilized by R.

We mentioned above that the max-flow problem does not have an efficient algorithm in the PRAM model without bit operations. Its dual is the problem of computing a minimum-weight s-t-mincut in a flow network. This is the weighted optimization problem associated with s-t-mincuts. The corresponding weighted counting problem is the problem of computing total weight of all s-t-mincuts in a flow network. Up to fast parallel reductions, this problem is equivalent in the PRAM model without bit operations to the problem of computing the permanent of a matrix, which is $\#P$-complete [44]. Hence it is expected to be much harder than the problem of computing a minimum-weight s-t-mincut. And yet, paradoxically, the task of proving a lower bound for this harder problem in the PRAM model without bit operations seems harder. This is so even if the model is restricted even further by allowing the running time or the number of processors to depend only on the number of input parameters and not their bitlength and, furthermore, by not allowing any branching or indirect references. The lower bound problem in this substantially restricted model is then equivalent to the one concerning formula size over integers. Valiant [45] conjectured that the permanent has no formula of polynomial size; this was posed as an algebraic analogue of the P vs. NP problem with the hope that techniques used in resolving it may shed light on the P vs. NP problem. Indeed, it is the first problem that we address in this paper.

The rest of this paper is organized as follows. In section 2 we outline our unified approach, which reduces the fundamental lower bound problems under consideration to instances of a certain orbit-closure problem in geometric invariant theory. In section 3 we review the main results in geometric invariant theory that we need. In section 4, we illustrate our approach on the lower bound problem for the formula size of the permanent. In section 5, we formulate an approach to the orbit-closure problem via *explicit obstructions*; these will be studied in depth in Part II of this paper. In sections 6, 7, and 8 we consider the P vs. NP problem with emphasis on its arithmetic implication.

2. A unified approach. Let F be an algebraically closed field. Let V be the vector space of homogeneous forms of degree m in l variables over F. Let Y denote the l-vector whose entries are these variables. Consider the reductive group $G = SL_l(F)$. It has a natural action on V. The action of any $\sigma \in SL_l(F)$ on a form $h \in V$ is given by $(\sigma h)(Y) = h(\sigma^{-1}Y)$. Let $P(V)$ denote the projective space associated with V. Then G acts on $P(V)$ as well. We shall think of a form as belonging to either V or $P(V)$ depending upon the context. Following Mumford [36], Mumford, Fogarty, and Kirwan [37], and Kempf [23], we shall say that the form $f \in P(V)$ is *stable* if the G-orbit of f in the affine space V is (Zariski) closed.

We shall reduce all lower bound problems under consideration for formulae or circuits over integers or algebraically closed fields of arbitrary characteristic to instances of the following general problem.

The orbit closure problem. Given fixed forms $g, f \in P(V)$, does f belong to the (projective) Zariski closure $\Delta[g]$ of the G-orbit of g in $P(V)$?

The explicitly given forms g and f will depend on the lower bound problem under consideration. They will be parametrized by l, the number of variables, which asymptotically goes to infinity. The orbit closure $\Delta[g]$ is a compactification of the homogeneous space G/H, where H is the stabilizer of g. The algebraic geometry of compactifications of homogeneous spaces is well understood in some special cases such as toric varieties [13], spherical embeddings [28], and compactifications of symmetric spaces [9]. In general, however, it seems very complex: a rough measure of this

complexity, proposed by Luna and Vust [28], is the difference between the dimension of the compactification and that of the orbit of a Borel subgroup. This is zero for spherical embeddings but quite high—roughly half of $\dim(G)$—in our case. This is one reason why the orbit-closure problem might well be intractable if f and g were arbitrary. The crucial result in this paper, which gives structure to this otherwise wild problem and connects complexity theory and geometric invariant theory, is that f, which arises in the context of lower bound problems in complexity theory studied here, has certain stability property, in the sense of geometric invariant theory [37], and moreover, both f and g have large nontrivial stabilizers that in a sense characterize them. We then formulate an approach to the orbit-closure problem, based on so-called *explicit obstructions*, that exploits this stability and presence of large characterizing stabilizers (section 5). This approach is developed in depth in Part II [34].

Specifically, f that arises in our context turns out to be a partially stable point in the null cone of G (Theorems 4.3 and 7.2). Partial stability (section 3) is a version of stability for points in the null cone obtained by relaxing the notion of stability due to Mumford [36], which is for points outside the nullcone. Roughly this means that (1) f is "almost" stable with respect to a reductive subgroup $L \subseteq G$ that is a Levi subgroup of a maximal parabolic subgroup $P \subseteq G$, (2) the orbit Gf is a fiber bundle over the base space G/P (whose algebraic geometry is very well understood [24]) such that each fiber is isomorphic to the L-orbit of f, and (3) the dimension of the fiber is sufficiently large in comparison to that of the orbit Gf. The precise relationship between these two dimensions will depend on the lower bound problem under consideration. Generally, the dimension of the orbit Gf will be bounded by a polynomial in the dimension of the orbit Lf, as the parameter l goes to infinity. If one could show that the answer to such an instance of the orbit-closure problem in a particular lower bound problem is negative for all large enough (or, more generally, infinitely many) l, then it would imply the desired lower bound.

Since g that arises in our context has a nontrivial reductive stabilizer that in a sense characterizes its structure, it is possible to reduce the orbit-closure problem further to a certain stabilizer problem as follows. Let D_g denote the stabilizer of g in $SL_l(F)$. First, we observe that if f lies in the closure of the $SL_l(F)$-orbit of g in $P(V)$, then, thinking of f and g as elements of the affine space V, f will lie in the closure of the $GL_l(F)$-orbit of g in V. However, the stabilizer within $SL_l(F)$ of any point in the $GL_l(F)$-orbit of g is a conjugate within $SL_l(F)$ of D_g. Hence, every (Zariski) open neighborhood of f within V will contain a point whose stabilizer in $SL_l(F)$ is a conjugate of D_g, and the same will hold for any point in the $SL_l(F)$-orbit of f in V. When this happens we will say that the group $D_g \subseteq SL_l(F)$ is present in the infinitesimal neighborhood of the $SL_l(F)$-orbit of f. The orbit-closure problem can now be reduced to the following.

The stabilizer problem. Let $D \subseteq G = SL_l(F)$ be an explicitly given reductive subgroup and $f \in V$ an explicitly given form. Can D occur as a stabilizer in the infinitesimal neighborhood of the G-orbit of f in V?

If $f \in V$ were a stable form, then Luna's étale slice theorem provides a satisfactory answer to this problem. Luna's result in our setting can be roughly stated as follows. Let H be the stabilizer of f. If f is stable, then there exists an affine variety N with an H-action such that some G-invariant affine neighborhood of the orbit of f "looks like" the homogeneous fiber bundle $G \times_H N$. This is a fiber bundle over the base space G/H with fiber N, and it is associated with the primary bundle $G \to G/H$. It easily follows from this result that if D were to occur in the infinitesimal neighborhood

of the $SL_l(F)$-orbit of f, then some conjugate of D must be contained in H. This implies, in turn, that if there exists a G-representation W that contains a trivial H-representation but not a trivial D-representation, then D cannot lie in the infinitesimal neighborhood of $SL_l(F)$-orbit of f, and hence f cannot lie in the (projective) closure of the $SL_l(F)$-orbit of a form $g \in P(V)$ with $D_g = D$. We say that such a W is an *obstruction* for the pair (f, g); i.e., its existence acts as a guarantee that f cannot lie in the projective closure of the orbit of $g \in P(V)$.

In our problem, f would be only partially stable. Therefore what we need is an extension of the preceding approach based on Luna's result from the stable to the partially stable situation. We suggest one such approach in this paper (section 5) based on explicit construction of representation-based obstructions, as above, in the partially stable situation; this serves as a starting point for our work in Part II [34]. In the context of lower bound problems in complexity theory, one can argue that obstructions ought to exist in plenty, assuming that the conjectured lower bound really holds. However, the problem is such that one has to prove their existence by constructing them more or less explicitly. Explicit construction has already been useful in complexity theory in a different context. For example, Margulis [29] and Lubotzky, Phillips, and Sarnak [26] give an explicit construction of expanders that are useful in a pseudorandom generation. The essential difference is that proving existence of expanders is easy, while proving existence of obstructions is itself a main problem.

The lower bound problem that arises in the context of the actual P vs. NP question (nonuniform boolean version) is for circuits defined over finite fields.[2] We shall reduce it (section 7) to a family of orbit-closure and stabilizer problems where f ranges over forms that are so-called F_p-equivalent to a certain base form f_b. We show that f_b is partially stable and conjecture the same for all forms F_p-equivalent to it. In any case, the lower bound problem for circuits (or formulae) over integers is a weaker version of the one over finite fields, in the sense that if g is a polynomial function with integer coefficients (e.g., the permanent) and F_p a fixed finite field, then a lower bound on the size of a circuit (or formula) over F_p for computing g, modulo F_p, implies the same lower bound on the size of a circuit (or formula) over integers for computing g. As such, it seems preferable to focus at first only on circuits or formulae over integers. In turn, the lower bound problem for circuits or formulae over integers for computing g is a weaker version of the problem over complex numbers. Hence, we shall concentrate mainly on circuits or formulae over complex numbers, or more generally, over algebraically closed fields of arbitrary characteristic.

3. Preliminaries. Geometric invariant theory is the study of actions of algebraic groups on algebraic varieties. Here reductive algebraic groups, such as the special linear group $SL_n(F)$, F algebraically closed, play a special role because they are geometrically reductive: this means that if V is a representation of a reductive group G, then there are enough polynomial functions on V which are invariant under G to distinguish disjoint closed G-invariant subsets of V. This was proved by Mumford [36] for linearly reductive groups, which include the classical simple groups over the complex field (cf. Weyl [46]), and was conjectured in general. It was eventually established by Seshadri [40] for $SL_2(F)$ and by Haboush [17] in general. What we shall really need is geometric reductivity of the special linear group $SL_n(F)$, where F is an algebraically closed field of arbitrary characteristic; for general n this was proved

[2] This finite field is usually taken to be F_2, but F_p, for any fixed prime p, will also do.

by Formanek and Procesi [12] independently of Haboush's general result.

3.1. Stability and partial stability. Consider the action of a reductive group G on an algebraic variety $\mathcal{X}$. It turns out that one cannot always put the structure of an algebraic variety on the set of all orbits of G in $\mathcal{X}$ but only on a subset chosen judiciously. For this purpose Mumford [36] introduced the notion of stability, building on the classical work of Hilbert [19], and exploited it systematically in his construction of quotients and moduli spaces. The definition of stability in the literature, though, has been anything but stable. Here we shall give a simplified definition following Kempf [23], which is somewhat different from the one in [36, 37].

Let V be a representation of a reductive group G. Then G acts on the projective space $P(V)$ of lines in V through the origin. Let y be a point in $P(V)$ and $x \in V$ any nonzero point on y. We say that y is *G-stable*, or simply *stable* if G is clear from the context, iff the orbit $G.x$ is (Zariski) closed and *properly stable* iff, in addition, the stabilizer of y in G is finite. By abuse of notation, we shall sometimes say that x is stable. We say that y is *semistable* iff the closure of the orbit $G.x$ does not contain zero. These definitions clearly do not depend on the choice of x on y. We will say that $x \in V$ is *unstable* or nilpotent if the closure of its orbit $G.x$ contains zero. The set of all nilpotent points in V is called the null cone. It can also be characterized as the set of common zeroes of all homogeneous invariants of positive degree. The concept of the null cone, important to us, was introduced by Hilbert [19].

Next, we define a weaker form of stability called partial stability, which will play a crucial role in this paper, and will be studied in depth in Part II. First assume that the characteristic is zero. Roughly, a point is partially stable if it is stable with respect to a large, nicely embedded, regular[3] reductive subgroup of G. Formally, we say that y is partial stable if there exists a maximal parabolic subgroup $P \subseteq G$ containing the stabilizer H of x in G such that

1. H contains the unipotent radical R of P—in other words, the unipotent radical fixes x;
2. y is "almost" stable with respect to a Levi subgroup $L \subseteq P$: This means the stabilizer L' of x in L is reductive (as it would be if y were stable with respect to L [32]), and y is stable with respect to a regular reductive subgroup $K \subseteq L$, whose rank is less than that of L by a small defect δ (in our case δ will be one);
3. the dimension of the orbit $Lx = L/L'$ is large enough in comparison with the dimension of the orbit Gx. The exact ratio Δ between these two dimensions will depend on the lower bound problem under consideration. Generally, the dimension of Gx will be bounded by a polynomial in the dimension of Lx.

When $P = G$, $\delta = 0$, and $\Delta = 1$, this coincides with the usual notion of stability. Therefore δ and Δ measure deviation from (complete) stability. It follows from the definition that H has a Levi decomposition of the form $L'R$, where L' is a reductive subgroup of L. It also follows that the orbit Gx is a fiber bundle over G/P such that each fiber is an affine subvariety of V isomorphic to L/L', the L-orbit of x. Moreover, the dimension of the fiber is sufficiently large in comparison with the dimension of Gx. That Gx is a fiber bundle over G/P with affine fiber has an important consequence for us: computation of the cohomology of a quasi-coherent sheaf over Gx can be reduced to a similar problem over G/P (cf. Theorem 5.4), where it is well understood [24]. Such explicit computation of cohomology will turn out to be important in our

[3]This means that the root system of the subgroup is a subsystem of the root system of G.

approach to the orbit closure problem (section 5).

If the characteristic is positive, we assume in the preceding definition of partial stability that G is a connected split reductive F-group (i.e., defined over a finite field F) and that P and H are also F-groups [42]. This is required to ensure that the Levi decomposition holds [4, 42].

3.2. Example. Let Y be a vector space of dimension $m+1$ with coordinates $y_0, \ldots, y_m$; X its subspace with coordinates $x_0, \ldots, x_n$, $n < m$, where $x_i = y_i$, $i \leq n$; and $\hat{X} \subseteq X$ the subspace with coordinates $x_1, \ldots, x_n$. We shall also think of y_i's as variables and, abusing the notation, denote by Y the variable vector with entries $y_0, \ldots, y_m$, and so on. Let $V = \mathrm{Sym}^r(Y^*)$ be the space of forms of degree r in $y_0, \ldots, y_m$, and $W = \mathrm{Sym}^s(\hat{X}^*)$, $s < r$, the space of forms of degree s in $x_1, \ldots, x_n$. We can embed W in V in a natural way using x_0 as the homogenizing variable. Formally, let ϕ be an embedding of W in V given by $(\phi h)(Y) = x_0^{r-s} h(\hat{X})$. Suppose $h \in W$ is stable with respect to the natural action of $SL_n = SL(\hat{X})$ on W with (reductive) stabilizer $K \subseteq SL(\hat{X})$.

Claim 3.1. *The embedded form $\phi h \in V$ is partially stable with respect to the action of $SL_{m+1} = SL(Y)$ on V. In other words, any stable form in the $SL(\hat{X})$-representation W becomes partially stable when embedded in the $SL(Y)$-representation V.*

Proof. First notice that the stabilizer of $\phi(h)$ in $SL(X) \subseteq SL(Y)$ is just K, assuming $r-s$ is larger than n. Although ϕh is not stable with respect to $SL(X)$—in fact, it lies in its null cone—it is stable with respect to $SL(\hat{X})$. Let $P \subseteq SL(Y)$ be the maximal parabolic subgroup that preserves the space X, i.e., consisting of matrices, acting on columns, of the form

$$\begin{bmatrix} * & * \\ 0 & * \end{bmatrix}, \tag{1}$$

where the diagonal stars denote matrices of dimension $n+1$ and $m-n-1$, respectively. This contains the stabilizer H of ϕh, as the reader can easily show. The unipotent radical R of P (i.e., the top right $*$) acts trivially on ϕh and so belongs to H. Let L be the Levi subgroup of P consisting of block diagonal matrices. The lower block of matrices in L also acts trivially on ϕh. Therefore only the action of the upper block matters. Thus the stabilizer H has a Levi decomposition $L'R$, where $L' \subseteq L$ consists of matrices of the form

$$\begin{bmatrix} \alpha & 0 & 0 \\ 0 & C & 0 \\ 0 & 0 & D \end{bmatrix}, \tag{2}$$

where α is a nonzero constant, a multiple of C belongs to K, and $D \in SL(\bar{X})$, $\bar{X}$ being the complement of X in Y. Since ϕh is stable with respect to $SL(\hat{X})$, whose rank is one less than that of $SL(X)$, it is clear that ϕh is almost stable with respect to L with defect $\delta = 1$. □

Thus each $SL(\hat{X})$-stable point in $P(W)$ gives rise to a partially stable point in $P(V)$ with respect to $SL(Y)$-action with defect one. The partially stable points in all our applications will arise from stable points in this simple fashion. Notice that the $SL(Y)$-orbit of ϕy is a fiber bundle over G/P, which is just a Grassmanian in this case. In our applications, m will be typically polynomial in n, and so is the ratio Δ between the dimension of the $SL(Y)$-orbit and the dimension of the fiber. The

dimension of the fiber will be sufficiently large, in this sense, compared to that of the $SL(Y)$-orbit; the importance of this condition will become clear in sections 4 and 6.

Also notice that explicit decomposition of V as an $SL(\hat{X})$-module can be determined by successive applications of the Pieri formula for the branching rule. The point ϕh belongs to the unique irreducible $SL(\hat{X})$-module isomorphic to $\mathrm{Sym}^s(\hat{X}^*)$ in this decomposition, and it is (completely) stable in this module with respect to the $SL(\hat{X})$-action. This is another way of looking at partially stable points that would arise in our context.

3.3. Criterion for stability. Given a point $x \in V$, how can one effectively decide if it belongs to the null cone? This is rather important, because although the ring of invariants for a reductive group action on affine varieties is finitely generated, only in rare cases can one determine an explicit set of generators for this ring. For $SL_n(C)$, Hilbert [19] proved a remarkable criterion that quite often enables one to prove instability or semistability without explicitly writing down a single invariant. This criterion was generalized to arbitrary reductive groups over any characteristic by Mumford [36]. The resulting Hilbert–Mumford criterion is a powerful tool for proving semistability. Actually, Mumford proved the following more general result, which is what we shall need.

For any closed G-invariant subset S of V, define $x \in V$ to be S-unstable if the closure of the orbit $G.x$ meets S. In the preceding classical case, i.e., when x belongs to the null cone, S contains only the zero point of V. In general, if the orbit of x is not closed, the boundary of this orbit contains a unique closed orbit of minimum dimension. We can let S be this closed orbit, and then x is S-unstable. Mumford [36] proved that if x is S-unstable, then there is a one-parameter subgroup λ of G which drives x to S; i.e., the closure of the orbit $\lambda.x$ also meets S. Kempf [23] generalized this result significantly. His result will provide us the main technical tool for proving stability of various forms that arise in the context of lower bound problems.

Kempf's criterion for stability. Assume that G has no nontrivial central one-parameter subgroup. If the stabilizer of $y \in P(V)$ in G is not contained in any proper parabolic subgroup of G, then y is stable.

Kempf's other results relevant to us may be summarized as follows. Suppose $x \in V$ is S-unstable. Then we have the following:

1. One can identify a nonempty subset $\Delta_{S,x}$ of one-parameter subgroups λ which drive x to S "most rapidly."
2. With every one-parameter subgroup λ, there is associated a parabolic subgroup $P(\lambda)$ of G consisting of those elements $g \in G$ such that $\lim_{t\to 0} \lambda(t)g\lambda^{-1}(t)$ exists in G. Then there is a parabolic subgroup $P_{S,x}$ of G such that $P(\lambda) = P_{S,x}$ for all $\lambda \in \Delta_{S,x}$.
3. Every maximal torus of $P_{S,x}$ contains a unique element of $\Delta_{S,x}$.
4. If G' is the stabilizer of x in G, then $P_{S,x}$ contains G'.
5. If, in addition, the characteristic is zero, and H is a reductive subgroup of G fixing x, then there is an element of $\Delta_{S,x}$ contained in the centralizer $Z_G(H)$ of H in G. The result also holds in positive characteristic, provided G is an F-group, F a finite field, and the Levi decomposition holds for parabolic F-subgroups of G, as it will in our case.

3.4. Slice theorem. A powerful tool in the analysis of actions of reductive groups around orbits of stable points is Luna's étale slice theorem. Let x be a stable point in a linear representation V of a reductive group. Let H be the stabilizer of x, which is reductive [32]. Then the orbit Gx of x is isomorphic to G/H. Let T

denote the tangent space of the orbit at x; it has a natural H-action. Assume that the characteristic is zero. Since H is reductive, V has an H-module decomposition $T \oplus M$, where M can be thought of as a "normal" space at x. Luna's slice theorem says that the orbit $Gx = G/H$ has a neighborhood in V that looks like (or more precisely, has a finite sheeted covering by) the induced fiber bundle $G \times_H N$ for some algebraic H-subvariety $N \subseteq M$. Formally, the following result holds.

THEOREM 3.1 (see [27, 37, 38]). *There is an excellent map ϕ from $G \times_H N$ to a G-invariant Zariski-open neighborhood U of Gx.*

The map ϕ is called excellent (or strongly étale) if the induced map $\phi/G : G \times_H N/G \to U/G$ is étale and ϕ is obtained from ϕ/G by base extension. Here $G \times_H N$ is the quotient of $G \times N$ with respect to the action of H given by

$$h(g, n) = (gh^{-1}, hn).$$

It has a natural G-action induced by the G-action (left multiplication) on $G \times N$ via the translation of the first factor.

When the base field is the field of complex numbers, the slice theorem implies that the orbit Gx has an analytic neighborhood that is actually isomorphic as a G-variety to $G \times_H N$ for some analytic (but not necessarily algebraic) H-variety $N \subseteq M$; N is called the slice at x. The slice theorem also implies that if Q occurs as a stabilizer in the infinitesimal neighborhood of the orbit Gx, then some conjugate of Q is a subgroup of H. However, determining if some conjugate of a given $Q \subseteq G$ is a subgroup of another given $H \subseteq G$ is, in general, nontrivial. This problem can be handled using explicit obstructions as in section 5.

4. Formula size. In this section we illustrate our approach by addressing a concrete lower bound problem for the formula size over an algebraically closed field. Suppose we are given a homogeneous form $g(X)$ of degree d in the n-dimensional vector variable $X = (x_1, \ldots, x_n)$ over an algebraically closed field F of arbitrary characteristic. We want to know if $g(X)$ has a formula of poly(n) (polynomial in n) size, i.e., of size $\leq n^a$, for some fixed constant a. By the formula size we mean the number of arithmetic operators in the formula. The formula size of a function is a good indicator of its parallel complexity [11, 45].

For example, suppose $n = k^2$ for some k. Think of X as a $k \times k$ matrix; so we shall denote the entries of X by x_{ij} instead of x_i. Let $g(X)$ be the permanent function

$$g(X) = \text{perm}(X) = \sum_{\sigma} \prod_{i} X_{i\sigma(i)},$$

where σ runs over all permutations of size k. This formula for the permanent has size $k \cdot k!$. However, there is a nonobvious substantially smaller formula of size $O(2^{O(k)})$ [30]. Valiant [44] conjectured that the permanent does not have a formula of size polynomial in k. No nonlinear lower bound is known so far; cf. [15] for a linear lower bound.

We shall now reduce the general lower bound problem for the formula size to an instance of the orbit-closure problem in section 2. Let Y be an $m \times m$ variable matrix, where $m > n$. Identify X with any $n \times n$-submatrix $\phi(X)$ of Y. Use some entry y of Y not in $\phi(X)$ as a *homogenizing* variable. Then any homogeneous polynomial g of degree d on X can be trivially identified with a homogeneous polynomial g^ϕ of degree m in the entries of the variable Y, where $g^\phi(Y) = y^{m-d} g(\phi(X))$.

Let V be the vector space of homogeneous forms of degree m in Y. Let $P(V)$ be the associated projective space. Let W be the vector space of homogeneous forms of

degree d in X and $P(W)$ the associated projective space. Then the function ϕ, which maps g to g^ϕ, is an embedding of W ($P(W)$) in V (resp., $P(V)$).

Let $G = SL_{m^2}(F)$. It has a natural action on V. Specifically, any $\sigma \in SL_{m^2}(F)$ acts on an $h \in V$ as $(\sigma h)(Y) = h(\sigma^{-1}Y)$, where we think of Y as an m^2-vector, by linearly ordering its entries in any way, so that the $m^2 \times m^2$-matrix σ^{-1} acts on Y in the usual way. This induces a natural action on $P(V)$ as well. The space V (and $P(V)$) contains the determinant form $\det(Y)$. Let $G(\det(Y)) \subseteq P(V)$ denote the orbit of $\det(Y)$ under this action (thinking of $\det(Y)$ as an element of $P(V)$.) Let $\Delta[\det(Y)]$ denote its Zariski closure in $P(V)$.

PROPOSITION 4.1. *If $f(X)$ has a formula of size l, then $f^\phi(Y)$ lies in the projective orbit closure $\Delta[\det(Y)]$ of the determinant, where $m = \dim(Y) = 2l$.*

Proof. We use the well-known fact that the determinant is a complete function for the class of polynomials having small formulae [11, 45]. Formally [45], if $f(X)$ had a formula of size l, there would exist a matrix M of size at most $2l$, whose each entry is either a constant or some entry x_{ij} of X such that

$$f(X) = \det(M).$$

If we identify the entries x_{ij} with the entries of the submatrix $\phi(X)$ of Y, and multiply each constant entry in M with the variable y of Y not in $\phi(X)$, we get a possibly singular matrix whose each entry is a linear combination of the entries of Y. In other words, we get a possibly singular linear transformation $A : F^{m^2} \to F^{m^2}$, where $m \leq 2l$, such that

$$f^\phi(Y) = \det(AY), \tag{3}$$

where Y is treated as an m^2-vector as above. However, $GL_{m^2}(F)$ is dense in the space of all linear transformations from F^{m^2} to itself. Therefore $f^\phi(Y)$ is in the closure of the $GL_{m^2}(F)$-orbit of $\det(Y)$ in V. The proposition follows from this. □

Thus we are led to the following instance of the orbit closure problem in section 2.

The orbit-closure problem. Can $f(Y) = \mathrm{perm}^\phi(Y)$ lie in the projective orbit closure $\Delta[\det(Y)]$ of $g(Y) = \det(Y)$ in $P(V)$?

The orbit-closure problem can in turn be reduced to an instance of the stabilizer problem in section 2. Towards that end, let us compute the stabilizer R of $\det(Y)$ in $G = SL_{m^2}(F)$. It is known to be a reductive subgroup which consists of linear transformations in G of the form (thinking of Y as an $m \times m$ matrix)

$$Y \to AY^*B^{-1}, \tag{4}$$

where Y^* is either Y or Y^T, $A, B \in GL_m(F)$. The determinant is completely characterized by its stabilizer in the following sense.

PROPOSITION 4.2. *Every form $h(Y) \in V$ stabilized by R is a multiple of* $\det(Y)$.

Proof. The proof is easy.

Now the preceding orbit-closure problem for the determinant can be reduced as in section 2 to the following.

The stabilizer problem. Can R occur as a stabilizer in the infinitesimal neighborhood of the $SL_l(F)$-orbit of $\mathrm{perm}^\phi(Y)$ in V?

We conjecture that this cannot happen if $m = \mathrm{poly}(n)$. Formally, we conjecture the following.

CONJECTURE 4.3. *If $m = n^a$, where a is any fixed constant, then $\mathrm{perm}^\phi(Y)$ cannot lie in the projective orbit closure $\Delta[\det(Y)]$. More specifically, R cannot occur*

as a stabilizer in the infinitesimal neighborhood of the $SL_l(F)$-orbit of $\text{perm}^\phi(Y)$ in V.

By Proposition 4.1, this would imply that the permanent does not have a formula of size polynomial in n. In section 5 we shall suggest an approach to this conjecture based on the construction of so-called explicit obstructions. The conjecture is not equivalent to the the original lower bound question (cf. section 4.2). However, it is "almost" equivalent in the following sense.

PROPOSITION 4.4. *Suppose F is the field of complex numbers. If Conjecture* 4.3 *were false, then $\text{perm}(X)$ can be approximated infinitesimally closely by a formula of quasi-polynomial $n^{O(\log n)}$ size (in the sense that the coefficients of the polynomial computed by this formula would be infinitesimally close to those of the permanent).*

Proof. If the conjecture were false, $\text{perm}^\phi(Y)$ lies in the closure of the SL_{m^2}-orbit of $\det(Y)$ in $P(V)$ with $m = \text{poly}(n)$. This means that, as a point in V, it lies in the closure of the GL_{m^2}-orbit of $\det(Y)$. Therefore it can be approximated infinitesimally closely by a function of the form $\det(\sigma^{-1}Y)$ for some $\sigma \in GL_{m^2}$. Let us ignore the terms in this approximation that involve variables other than y or entries of $X = \phi(X)$, which cannot occur in $\text{perm}^\phi(Y)$; i.e., think of setting these variables of Y to zero. What remains is still an approximation to $\text{perm}^\phi(Y)$. Setting y to one, we get an infinitesimally close approximation to $\text{perm}(X)$ by a function of the form $\det(W)$, where W is an $m^2 \times m^2$-matrix whose each entry is a (possibly nonhomogeneous) linear combination of the entries of X. However, such a function $\det(W)$ has a formula of size $m^{O(\log m)} = n^{O(\log n)}$ [2]. □

However, such an approximation is not expected in view of Valiant's completeness result for the permanent [44]. This provides a good support for the conjecture.

As we remarked in section 2, the general orbit-closure or the stabilizer problem seems intractable if f and g were arbitrary forms. The situation is better here because they have the following nice properties: (1) g is stable and f is partially stable (section 4.1), and (2) both have large nontrivial stabilizers that capture their structure (Propositions 4.2 and 4.5).

We have already seen that $\det(Y)$ has a large characteristic stabilizer (Proposition 4.2). Now let us turn to $\text{perm}(X)$, where the dimension of X is k. Assume that the characteristic is not equal to two. (Otherwise the permanent is equal to the determinant.) X can also be thought of as an n-vector, where $n = k^2$. Consider the natural action of $SL_n(F)$ on the projective space $P(W)$ of forms in X of degree k. Then the stabilizer K of $\text{perm}(X)$ in $SL_n(F)$ is generated [30] by linear transformations of the form (thinking of X as a $k \times k$ matrix)

$$X \to \lambda X \mu^{-1}, \tag{5}$$

where λ and μ are either diagonal or permutation matrices, and $k \geq 3$. The connected component K^0 of K is a $(2k-1)$-dimensional torus.

The following is an analogue of Proposition 4.2 that says that the permanent is completely characterized by its stabilizer.

PROPOSITION 4.5. *Every form $h(X) \in W$ stabilized by K is a multiple of $\text{perm}(X)$.*

Proof. The proof is easy.

4.1. Stability. Now we turn to the issue of stability.

THEOREM 4.1. *The point $\det(Y) \in P(V)$ is stable with respect to the action of $G = SL_{m^2}(F)$ on $P(V)$, where V is the space of homogeneous forms of degree m in Y.*

Proof. The stabilizer R of $\det(Y)$ is not contained in any parabolic subgroup of G because its representation over the space of $m \times m$ matrices given by (4) is irreducible. Therefore stability follows from Kempf's criterion [23] (cf. section 3). □

Remark. Stability of $\det(Y)$ is not needed in our approach to Conjecture 4.3 in section 5; only the stability of $\mathrm{perm}(X)$ is needed. However, stability of $\det(Y)$ is needed in other problems, where the determinant plays the role of f in the orbit-closure problem, e.g., in the NC^1 vs. NC^2 problem [11].

THEOREM 4.2. *The point* $\mathrm{perm}(X) \in P(W)$, *where* $\dim(X) = k$, *is stable with respect to the action of* $SL_n(F)$ *on* $P(W)$, *where* $n = k^2$.

Proof. The stabilizer of $\mathrm{perm}(X)$ in $SL_n(F)$ is, again, not contained in a proper parabolic subgroup because the space of $n \times n$ matrices is an irreducible representation. Therefore the assertion again follows from Kempf's criterion (section 3).

We shall also give another proof based on the property (5) of Kempf's one-parameter subgroups (cf. section 3.3) because it illustrates in a simple setting an idea that will be used later (section 8) in proving the stability of another form. This second proof ignores the discrete part of the stabilizer.

Let $\mathcal{X}$ be the vector space whose coordinates are the entries x_{ij} of X. It is a representation of K^0 as given by (5). Let E_{ij} be the one-dimensional vector space of all matrices with the only nonzero entry in the (i,j)th spot. It is easy to check that $\mathcal{X}$ decomposes as a K^0-module into $\oplus_{i,j} E_{i,j}$ and that this representation is multiplicity-free; i.e., all characters are distinct. Now assume that $\mathrm{perm}(X)$ is not stable. Let S be an $SL_n(F)$-orbit of minimum dimension in the closure of the orbit of $\mathrm{perm}(X)$ in V. It must be closed; cf. Borel [4]. In fact, since $SL_n(F)$ is reductive, and hence geometrically reductive (cf. Haboush [17]), there is a unique closed orbit S in the closure of the orbit of $\mathrm{perm}(X)$ in V.

By Kempf's result, there would be a one-parameter subgroup $\lambda : F^* \to SL_n(F)$ which drives $\mathrm{perm}(X)$ into S and which commutes with K^0. Schur's lemma forces the action of $\lambda(t)$ on $\mathcal{X}$ to be of the form

$$e_{i,j} \to t^{\alpha_{i,j}} e_{i,j},$$

where $e_{ij} \in E_{ij}$ and α_{ij} is an integer. We also have the constraint that $\sum_{i,j} \alpha_{i,j} = 0$ because $\lambda(t) \in SL_n(F)$. The effect of $\lambda(t)$ on any monomial $m_\sigma = \prod_{i=1}^m x_{i,\sigma(i)}$ in the expansion of $\mathrm{perm}(X)$ is as follows:

$$m_\sigma \to t^{\sum \alpha_{i,\sigma(i)}} m_\sigma.$$

Thus for $\lim_{t \to 0} \lambda(t)(m_\sigma)$ to exist, $\sum_{i=1}^k \alpha_{i,\sigma(i)}$ must be nonnegative for every permutation on $[k]$ (since there is no cancellation across monomials). This is possible only when each sum is actually zero, as can be seen using Hall's theorem that every regular bipartite graph is a disjoint union of perfect matchings. Thus if $\lim_{t \to 0} \lambda(t)(\mathrm{perm}(X))$ exists, then it is $\mathrm{perm}(X)$ itself. In other words, $\mathrm{perm}(X)$ cannot be S-unstable for any G-invariant closed S. □

In contrast, the point $\mathrm{perm}^\phi(Y) \in P(V)$ is not stable with respect to the action of $G = SL_{m^2}(F)$. In fact, it belongs to the null cone; this can be shown using the Hilbert–Mumford criterion [37]. However, the following result holds.

THEOREM 4.3. *The point* $\mathrm{perm}^\phi(Y) \in P(V)$ *is partially stable with respect to the action of* $G = SL_{m^2}(F)$ *with defect* $\delta = 1$.

Proof. If $h \in P(W)$ is stable, then $h^\phi \in P(V)$ is partially stable with defect $\delta = 1$, arguing exactly as in Example 3.2. Specifically, X and Y are now matrices instead of

vectors, and the homogenizing variable is y here instead of x_0. The parabolic group P consists of the linear transformations that transform the variables in Y to their linear combinations. The Levi subgroup $L \subseteq P$ consists of the linear transformations that transform the variables in Y and $X \setminus Y$ to the linear combinations of the variables in Y and $X \setminus Y$, respectively. The subgroup $L' \subseteq L$ consists of the linear transformations that transform y to its constant multiple, and the variables in Y and $X \setminus (Y \cup \{y\})$ to the linear combinations of the variables in Y and $X \setminus (Y \cup \{y\})$, respectively.

The partial stability of $\mathrm{perm}^{\phi}(Y)$ then follows from Theorem 4.2. □

The SL_{m^2}-orbit Z of $\mathrm{perm}^{\phi}(Y)$ in V is a fiber bundle over G/P, where P is similar to the one in (1), and so G/P is a Grassmanian variety. The fiber is isomorphic to the SL_{n^2}-orbit of $\mathrm{perm}(X)$ in W. The stabilizer H of $\mathrm{perm}^{\phi}(Y)$ is related to the stabilizer K of $\mathrm{perm}(X)$ just as in (2). The dimension of Z is equal to $\dim(SL_{m^2}) - \dim(H)$, and that of the fiber W is $\dim(SL_{n^2}) - \dim(K)$. In our problem, m is not too large compared to n; in fact, it is polynomial in n. Hence the dimension of the global orbit Z is also polynomial in the dimension of the fiber. The parameters δ and Δ in the definition of partial stability (section 3) that measure deviation from complete stability are not too large: we have $\delta = 1$ and $\Delta = \mathrm{poly}(n)$. Roughly, this says that although $\mathrm{perm}^{\phi}(Y)$ belongs to the null cone, it is not "too unstable."

Remark. The role of the determinant in this section can also be played by the trace: specifically, the analogues of Proposition 4.1 and Theorem 4.1 hold for $\mathrm{trace}(Y^m)$ as well.

4.2. Exterior limit points. Here we shall indicate why Conjecture 4.3 is expected to be somewhat stronger than the original lower bound question for the formula size of the permanent. It is because $\Delta[\det(Y)]$ may contain points that do not have polynomial size formulae. Let $g = \det(Y)$. We call a point in $\Delta[g]$ an *interior limit point* if it is of the form $\det(\sigma Y)$ for some *possibly singular* linear transformation σ. In other words, the set of interior points is the image of the map $\alpha : M_{m^2}(F) \to \Delta[g]$, where $M_{m^2}(F)$ is the set of $m^2 \times m^2$-matrices over F, and α maps a possibly singular matrix $\sigma \in M_{m^2}(F)$ to $\det(\sigma Y)$. This image is a constructible subset [18] of $\Delta[g]$; i.e., it is a finite disjoint union of locally closed subsets. However, it is not expected to be equal to $\Delta[g]$. The points of $\Delta[g]$ not in this image will be called *exterior limit points*. Interior limit points have formulae of quasi-polynomial size (see the proof of Proposition 4.4), whereas exterior limit points need not.

We now describe some classes of conjectured exterior limit points in $\Delta[g]$. Formally proving that a given point is an exterior limit point amounts to showing that it does not have a formula of $\Omega(\mathrm{poly}(n))$ size because the determinant is complete for the class of polynomials with small formulae; cf. the proof of Proposition 4.1. This is a lower bound problem in itself, beyond our reach at present.

If f is an interior limit point of $\Delta[g]$, then $f = \det(\sigma Y)$, where σ is possibly singular. Since diagonalizable matrices are dense in $M_{m^2}(F)$, it follows that some conjugate f' of f is of the form $\beta(g')$, β is diagonal, and g' is some conjugate of g. Let $\beta(t)$ be the one-parameter subgroup obtained by replacing the zero diagonal entries of β by t. Then $\beta = \lim_{t \to 0} \beta(t)$. We shall now give an example of a point f in $\Delta[g]$, which is of the form $\lim_{t \to 0} \det(\alpha(t)Y)$ for some one-parameter group $\alpha(t)$, but it is not expected to be of the form $\beta(g)$, where $\beta = \lim_{t \to 0} \beta(t)$ for some one-parameter group $\beta(t)$.

Let G be a nonbipartite graph on m vertices whose edges have nonnegative integer weights. Consider its skew-symmetric Tutte matrix M whose (i,j)th entry M_{ij}, $i < j$, is the variable y_{ij} if G has an edge joining vertices i and j and zero otherwise. We

have

$$\det(M) = \sum_{\pi} \operatorname{sign}(\pi)\operatorname{wt}(\pi),$$

where π ranges over all permutations of $[1, \ldots, m]$, and $\operatorname{wt}(\pi)$ is the product of the entries $M_{i\pi(i)}$. Define the cycle graph of π with nonzero weight to be the subgraph of G formed by the edges $(i, \pi(i))$. Then it is known and easy to see that

$$\det(M) = \sum_{\pi'} \operatorname{sign}(\pi')\operatorname{wt}(\pi'), \tag{6}$$

where π' ranges over only those permutations with nonzero weight whose cycle graphs contain only cycles of even length. The contributions of other permutations cancel out.

Construct a modified Tutte matrix $\tilde{M}$ by replacing every variable y_{ij}, if it indeed occurs in M, by $t^{w_{ij}} y_{ij}$, where t is an indeterminate parameter and w_{ij} is the weight of the edge joining the nodes i and j. In other words, $\tilde{M} = \beta(t)(M)$, where $\beta(t)$ is the one-parameter subgroup that maps such y_{ij} to $t^{w_{ij}} y_{ij}$. Let Y be the vector of the variables y_{ij}. Then (6) implies that

$$D(Y) = \det(\tilde{M}) = t^{2W} h(Y) + \text{higher order terms in } t,$$

where W is the weight of a minimum-weight perfect matching in G and

$$h(Y) = \sum_{\pi} \operatorname{sign}(\pi)\operatorname{wt}(\pi),$$

where π ranges over all permutations with nonzero weight whose cycle graphs can be decomposed as the union of two minimum-weight perfect matchings, not necessarily distinct. Clearly, $D(Y) = \det(\tilde{M})$ belongs to the projective orbit closure $\Delta[\det(Y)]$ for every t. Projectively, $D(Y)$ is equal to $h(Y)$ plus higher order terms in t. Since

$$h(Y) = \lim_{t \to 0} t^{-2W} D(Y),$$

in the projective space $P(V)$, we have

$$h(Y) = \lim_{t \to 0} D(Y) = \lim_{t \to 0} \det(\beta(t)M).$$

In other words, $h(Y)$ belongs to the projective closure $\Delta[\det(Y)]$; notice, however, that $\det(\beta M)$, $\beta = \lim_{t \to 0} \beta(t)$, is not well defined in the projective space $P(V)$. We conjecture that for every m there are weighted graphs G such that $h(Y)$ does not have a small polynomial size formula. This would imply that $h(Y)$ is an exterior point of $\Delta[\det(Y)]$, since the determinant is complete for the class of polynomials with small formulae.

For most weight assignments on the edges of G, there is a unique minimum-weight perfect matching (this follows from the so-called isolation lemma [35]), which means $h(Y)$ contains just one monomial, and hence surely has a small formula. What the conjecture says is that there exist pathological weighted graphs. Nonbipartiteness of G is essential here. Because if G is bipartite, then for any weight-assignment on G consider the subgraph G' formed by the union of all minimum-weight perfect matchings in G. It easily follows from Hall's theorem that the set of perfect matchings

in G' is equal to the set of minimum-weight perfect matchings in G. Therefore $h(Y)$ for G is equal to the determinant of the Tutte matrix for G' and thus has a small formula [2].

Though $h(Y)$ is an exterior limit point, it can be approximated infinitesimally closely by a formula of quasi-polynomial size (Proposition 4.4), whereas this is not expected for the permanent because of its completeness.

5. Explicit obstructions. Let V be a linear representation of a reductive group G and $P(V)$ be the associated projective space. A nonzero point $v \in V$ also corresponds to a point in $P(V)$. By abuse of notation, we denote the corresponding point in $P(V)$ by v as well. Suppose we are given two explicit points $f, g \in V$, and the goal is to show that that f is not in the closure of the G-orbit of g in the projective space $P(V)$. Conjecture 4.3 is a special case of this problem, where V is the space of homogeneous forms of degree m in the entries of an $m \times m$ variable matrix Y, $f = \text{perm}^{\phi}(Y)$, $g = \det(Y)$, and $G = SL(Y)$. Here f lies in the closure of the G-orbit of g in $P(V)$ iff it lies, considered as a point in V, in the closure of the $GL(Y)$-orbit of g in V.

We call a mathematical structure an *obstruction* for the pair (f, g) if its existence acts as a guarantee that f does not lie in the closure $\Delta[g]$ of the G-orbit of g in the projective space $P(V)$.

5.1. Polynomials as obstructions. Let Z denote the closure of the G-orbit of g in $P(V)$. Let $I(Z)$ denote the ideal of polynomial functions vanishing on Z. Any polynomial in $I(Z)$ that does not vanish at f is an obstruction for the pair (f, g) because it vanishes on the orbit of g but not on the orbit of f. We say that it *separates* the two orbits. If f does not lie in Z, then "almost every" polynomial in $I(Z)$ is an obstruction. In the setting of Conjecture 4.3, where $g = \det(Y)$ and $f = \text{perm}^{\phi}(Y)$, it is thus expected that almost every polynomial in $I(Z)$ will be an obstruction for the pair (f, g) when m is polynomial in n. To prove Conjecture 4.3 it suffices to construct an obstruction for every such m and n. Even though we know, assuming Conjecture 4.3, that such obstructions exist in plenty, the problem is such that one has to prove their existence by constructing them more or less *explicitly*.

One approach to constructing an explicit polynomial obstruction is to construct an explicit set of generators of $I(Z)$ and then isolate a generator that does not vanish at f. This problem is addressed in Part II [34], where we give a sufficiently explicit set Σ_g of generators for the ideal of the G-orbit of g in the affine space V, assuming that g is stable and has a "nice" stabilizer, and conjecture that this set actually generates $I(Z)$ for g's that arise in the context of the lower bound problems in this paper. The generators in Σ_g are quite complex and verifying that one of them does not vanish at f is a difficult task. However, they have nice representation-theoretic properties. In Part II we develop in detail an approach to the orbit-closure problem via explicit construction of obstructions based on these properties.

Here we wish only to indicate how representation theory comes into play by briefly considering explicit construction of obstructions in the context of the related stabilizer problem. The scheme for explicit construction suggested here depends on the representation-theoretic properties of the stabilizers of f and g, and serves as a starting point for the work in Part II.

Specifically, the scheme will exploit the fact that, in our context, f is partially stable (cf. Theorem 4.3) and both f and g have nontrivial reductive stabilizers that capture their structure (Propositions 4.5 and 4.2); g may or may not be stable. The

obstructions that one hopes to construct this way are not polynomials but rather representations having certain properties (cf. Theorems 5.1 and 5.2).

5.2. Representations as obstructions. In this section, we shall present our scheme when f is (completely) stable with respect to the G-action. In section 5.5 we shall suggest an extension of this scheme to the case when f is only partially stable as in Conjecture 4.3. There we shall also point out the main problems that are encountered in such an extension.

When f is stable, obstructions based on representations can be constructed using the following immediate consequence of Luna's slice theorem.

THEOREM 5.1 (characteristic zero). *Let $H \subseteq G$ be the stabilizer of $f \in V$ and $Q \subseteq G$ the stabilizer of $g \in V$. Suppose f is stable with respect to the action of G. Then a (nonzero) representation W of G is an obstruction for the pair (f, g) if W contains a trivial H-submodule but not a trivial Q-submodule. In other words, if such a W exists, then f cannot lie in the closure of the G-orbit of g in the projective space $P(V)$. More generally, W is an obstruction for (f, g) if the multiplicity of the trivial H-representation within W exceeds that of the trivial Q-representation.*

Proof. Since f is stable, by Luna's slice theorem (section 3), there is a slice N at f, a G-invariant neighborhood U of the orbit Gf, and a map $\tau : G \times_H N \to U$ that is excellent. This implies that (cf. [38]) the stabilizer of any point in U is a conjugate of a subgroup of H.

Suppose, to the contrary, that f, considered as a point in $P(V)$, lies in the closure of the G-orbit of g in $P(V)$. Then the neighborhood U must contain a point p whose some multiple p' lies in the G-orbit of g in the affine space V. The stabilizer of p is equal to the stabilizer of p', which is a conjugate of the stabilizer Q of g. Since some conjugate of the stabilizer of p is contained in H, it follows that some conjugate of Q is contained in H. Hence the multiplicity of the trivial Q-representation in any G-module must exceed that of the trivial H-representation.

Therefore if there exists a G-module W such that the multiplicity of the trivial H-representation within W exceeds that of the trivial Q-representation, then f cannot lie in the closure of the orbit of g in $P(V)$. □

Second proof. We now prove the first criterion of Theorem 5.1 (though not the second) without using Luna's slice theorem but using Peter–Weyl theorem instead; the idea in this proof will be useful later in section 5.5. We shall denote the algebraically closed base field of characteristic zero by k. Because Gf is a closed affine subvariety of V and the characteristic is zero,

$$k[V] = k[Gf] \oplus I, \tag{7}$$

as a G-module, where I is the G-invariant ideal of Gf. Thus $k[Gf] = k[G/H]$ can always be embedded equivariantly within $k[V]$. Now we shall determine an explicit G-module decomposition of the ring $k[G/H]$. First, consider $k[G]$, the ring of regular functions on G, as a $G \times G$ module by letting one copy of G act on it by left translations and the other by right translations. Then $k[G]$ has the following decomposition as a $G \times G$ module [6]:

$$k[G] = \sum_P P \otimes \bar{P}, \tag{8}$$

where P runs over all finite-dimensional G-modules, and $\bar{P}$ is its dual. This follows by applying the Peter–Weyl theorem to the Hilbert space $L_2(K)$ of L_2-functions on

a maximal compact subgroup K of G and using the fact that the K-finite vectors in $L_2(K)$ correspond to the regular polynomial functions on G; cf. [6, 22]. Since f is stable, H is reductive [32]. Then $k[G/H]$ is simply the ring of $k[G]^H$ of (right) H-invariants within $k[G]$ [37]. Therefore the decomposition in (8) implies that

$$k[Gf] = k[G/H] = \sum_P P \otimes \bar{P}^H, \tag{9}$$

where P runs over all finite-dimensional G-modules, $\bar{P}^H$ is the sum of all trivial H-submodules of $\bar{P}$, and G acts on the first factor of $P \otimes \bar{P}^H$.

Let W be an irreducible representation containing a trivial H-module but not a trivial Q-module. It can be equivariantly embedded within $k[G/H] = k[Gf]$, by (9), and hence within $k[V]$ by (7). Thus each element in W corresponds to a polynomial function on V. Now suppose to the contrary that the G-orbit of f in $P(V)$ lies in the projective closure $\Delta[g]$ of the G-orbit of g in $P(V)$. Not all functions in W can vanish on the affine cone C in V corresponding to the G-orbit of g in $P(V)$ because then they will also vanish on the affine cone in V corresponding to $\Delta[g]$, and hence on the G-orbit of f in V, since $f \in \Delta[g]$. But this is impossible since W is a nonzero submodule of $k[Gf]$. Therefore W contains at least one function ϕ which does not vanish at some point of the cone C, say, p. The stabilizer $Q_p \subseteq G$ of p is conjugate to the stabilizer Q of g. Let $W_\phi \subseteq W$ be the Q_p-submodule generated by ϕ. Consider the evaluation map from W_ϕ to k, which maps any function in W_ϕ to its value at p. Since ϕ is nonzero at p, this evaluation map is nonzero. It is also equivariant with respect to the Q_p-action. Since k is the trivial Q_p-representation, and the characteristic is zero, W_ϕ, as a Q_p-module, contains a trivial Q_p-submodule, and so does W. Since W is a G-module and Q_p is a conjugate of Q in G, it follows that W must contain a trivial Q-module; a contradiction. □

To use the obstruction criterion in this theorem effectively, one needs an explicit formula for the dimension of the trivial H (or Q) representation within a G-module W. This is an instance of the subgroup restriction problem, which will also arise later (section 5.5).

5.2.1. Example 1. Let X be the variable matrix of size d, $V = \mathrm{Sym}^d(X)$, the space of forms in X of degree d, and $G = SL(X) = SL_{d^2}(\mathbb{C})$. Let $f = \mathrm{perm}(X)$ and $g = \det(X)$ with stabilizers H and Q, respectively. We want to show that f is not in the closure of the G-orbit of g in $P(V)$.

In the present case, this can be shown by only dimension considerations. Indeed, if f were in the closure of the G-orbit of g in $P(V)$, then by Luna's slice theorem some conjugate of Q will be contained within H. However, since the dimension of Q exceeds that of H this is not possible.

However the dimension-based criterion seems too crude to be generalized to the partially stable case. Because when $(f, g) = (\mathrm{perm}^\phi(Y), \det(Y))$ as in Conjecture 4.3, the dimension of the stabilizer of f is far greater than that of g when m is only mildly larger than n. On the other hand, representations as obstructions, as in Theorem 5.1, seem suited even for the partially stable case (section 5.5). With that in mind, we construct an obstruction for the pair (f, g) using Theorem 5.1, which is applicable since $f = \mathrm{perm}(X)$ is stable with respect to the action of $G = SL(X)$ (Theorem 4.2).

Let $W = Sym^r(X)$ be the G-module of homogeneous polynomials of degree r in the entries of X, where r is a multiple ad of d for some integer $a > 1$. The multiplicity of the trivial H-representation in W exceeds one, because $\mathrm{perm}(X)^a \in W$ is fixed by

H, and so is the permanent of the matrix obtained by replacing each entry of X by its ath power. As a Q-module [7]

$$W = \sum_{\lambda} S_\lambda \otimes \bar{S}_\lambda,$$

where λ ranges over all Young diagrams of size r with at most d rows or columns, S_λ denotes the corresponding Weyl module, and $\bar{S}_\lambda$ denotes its dual. Here the elements in Q of the form (A, B), $A, B \in GL_d$, $\det(AB^{-1}) = 1$, act on $S_\lambda \otimes \bar{S}_\lambda$ in the obvious manner, and the transpositions in Q interchange the two factors. When $r = ad$, $S_\lambda \otimes \bar{S}_\lambda$ is trivial as a Q-module precisely when λ is rectangular with d rows and a columns. Hence the multiplicity of the trivial Q-module in W is precisely one. This trivial module is generated by $\det^a(X)$. By Theorem 5.1, any $\mathrm{Sym}^r(X)$, $r = ad$, $a > 1$, is an obstruction for the pair (f, g).

The preceding obstructions were constructed using the second multiplicity-based criterion of Theorem 5.1. It would be interesting to construct obstructions using the first criterion based on absence or presence of trivial representations. Specifically, it would be interesting to construct a representation W, for each d, such that W contains a trivial H-representation but not a trivial Q-representation. For this one needs to know how a given G-representation W splits when considered as a Q-module or as an H-module. Q contains a copy of $SL_d \otimes SL_d$, and H contains a copy of $S_d \times S_d$, where S_d is the symmetric group. Hence this problem is closely linked to the so-called plethysm problem of representation theory. This asks for explicit decomposition of $S_\lambda(V \otimes W)$ as a $GL(V) \times GL(W)$-module and that of $S_\lambda(S_\mu V)$ as a $GL(V)$-module, where S_λ is the Schur-functor [14].

5.3. Importance of stability. The stability of f is crucial for Theorem 5.1 to hold. Otherwise, the first proof does not apply because Luna's slice theorem may fail when f is not stable even if the stabilizer H of f is reductive (cf. Example 2 below). The second proof also fails when f is not stable because it may not be possible to extend the functions in the module $W \subseteq k[Gf] = k[G/H]$ there equivariantly to the whole of V, since Gf is not closed. It may happen that $k[V]$ contains a G-submodule W that contains a trivial H-representation but not a trivial Q-representation. That still does not guarantee that W is an obstruction because all functions in W may vanish identically on the the affine cones in V corresponding to the G-orbits of g and f in $P(V)$. In other words, an abstract representation W need not be an obstruction even if it contains a trivial H-module but no trivial Q-module. However, such a W becomes an obstruction if it has a concrete realization as a G-submodule of $k[V]$ that does not vanish when restricted to the G-orbit of f in V. When this happens (as it does when f is stable), a slight modification of the argument in the second proof of Theorem 5.1 shows that it contains a function that separates the orbits of f and g, i.e., vanishes on the orbit of g but not on the orbit of f, though such a separating function is not explicitly constructed. Thus representation-based obstructions provide an indirect way of constructing—or more precisely, guaranteeing the existence of—separating functional obstructions as in section 5.1.

5.3.1. Example 2. We give a simple example that demonstrates why stability of f is crucial for Theorem 5.1 to hold. Let V be the space of forms of total degree 3 in variables x and y, and $G = SL_2$. Let $f = x^2y$, and $g = x^3 + y^3$ [38]. Then it is easy to show that f lies in the projective closure $\Delta[g]$ of the G-orbit of g in $P(V)$. (The G-orbit of g in $P(V)$ is three-dimensional, and that of f is two-dimensional, since the

G-stabilizer of the line in V containing f and the origin is one-dimensional.) The stabilizer $H \subseteq G$ of f for the G-action on V is trivial, hence reductive, but f is not stable for the G-action. The stabilizer Q of g is equal to $\{\mathrm{diag}(\epsilon, \epsilon^{-1}) | \epsilon^3 = 1\}$. Since H is trivial, any G-representation contains a trivial H-representation. Now there are G-representations which do not contain a trivial Q-representation, e.g., the standard representation. Yet none of these is an obstruction for the pair (f, g), since we already know that f lies in $\Delta[g]$.

5.4. Positive characteristic.

THEOREM 5.2. *With the same terminology as in Theorem* 5.1*, suppose, in addition, that the orbit map is separable and that the ambient space V has an H-module decomposition in which the tangent space of the orbit Gf at f splits off as a direct summand. Then the obstruction criterion given there holds.*

Here, by multiplicity of the trivial H- or Q-representation in W, we now mean the total dimension of the sum of such trivial submodules of W. This need not be equal to the number trivial quotients in a Jordan-series filtration of W, since, in positive characteristic, W need not decompose into irreducible Q- or H-submodules.

Proof. Luna's slice theorem holds in arbitrary characteristic if the orbit map is separable and V has an H-module decomposition as above [1]. Therefore the proof of Theorem 5.1 holds. □

This points out the issues that arise in positive characteristic because of the lack of linear reductivity.

5.5. Partially stable case. In the preceding section we gave a method (Theorem 5.1 and 5.2) for constructing representation-theoretic obstructions for the pair (f, g) when f is stable. We now suggest an extension of this approach to produce obstructions when f is only partially stable. The setting in Conjecture 4.3 falls in this category. Let H denote the stabilizer of f. Let Q denote the stabilizer of g; we assume that it is reductive.

Let α denote the embedding of the orbit $Z = Gf$ in the ambient space V. Let $\mathcal{O}_V$ denote the sheaf of *analytic* functions on V. Let $\mathcal{O}_V^Z = \alpha^{-1}\mathcal{O}_V$ denote the inverse image sheaf on Z (cf. Hartshorne [18]). The stalk of $\mathcal{O}_V^Z$ at any point $z \in Z$ coincides with the stalk of $\mathcal{O}_V$ at z. The global sections, i.e., the elements of $H^0(Z, \mathcal{O}_V^Z)$, of this sheaf correspond to germs of analytic functions that are defined in some analytic open neighborhood of Z in V. The space $H^0(Z, \mathcal{O}_V^Z)$ is a G-module in a natural way.

We denote the algebraically closed base field by k in this section.

PROPOSITION 5.1. *Assume that the characteristic is zero. Let W be any finite-dimensional G-module in $H^0(Z, \mathcal{O}_V^Z)$ that does not vanish identically under the (natural) restriction to Z. Then W is an obstruction for the pair (f, g) if it does not contain a trivial Q-representation.*

Proof. Fix such a G-module W that does not vanish when restricted to Z. Since the G-action on Z is transitive, there exists a global section $\phi \in W$ that does not vanish at f when restricted to Z. Let U_ϕ be an analytic neighborhood of Z in V on which ϕ is well defined. Suppose to the contrary that the closure of the G-orbit of g in $P(V)$ contains f. Then U_ϕ contains a point p arbitrarily close to f whose multiple $\hat{p}$ lies in the G-orbit of g in the affine space V. Choose p close enough to f so that ϕ does not vanish at p. The stabilizer of p is the same as that of $\hat{p}$, which in turn is equal to the conjugate Q^α of Q for some $\alpha \in G$. Consider any $\beta \in Q^\alpha$. The function $\beta(\phi)$ is defined in the neighborhood $\beta(U_\phi)$ of Z which contains p, since it is fixed by β. Thus all functions in the Q^α-submodule $W_\phi \subseteq W$ generated by ϕ are well defined at p. Consider the evaluation map from W_ϕ to k, which maps any function

in W_ϕ to its value at p. Since ϕ is nonzero at p, this evaluation map is nonzero. It is also equivariant with respect to the Q^α-action. Since k is a trivial Q-representation, and the characteristic is zero, W_ϕ contains a trivial Q^α-module in its decomposition into irreducible Q^α-modules, and so does W. Since W is a G-module and $\alpha \in G$, its decomposition into irreducible Q-modules also contains a trivial Q-module. □

To construct an obstruction using this proposition, one needs to calculate explicitly an interesting family of finite-dimensional G-submodules within $H^0(Z, \mathcal{O}_V^Z)$. When f is stable, one gets such a family as follows.

THEOREM 5.3. *Suppose $k = \mathbb{C}$, and f is stable. Then every irreducible G-representation W that contains a trivial representation as an H-module occurs in $H^0(Z, \mathcal{O}_V^Z)$ (with multiplicity at least equal to the multiplicity of the trivial H-representation in W).*

Proof. By Luna's slice theorem (section 3), the orbit $Gf \cong G/H$ has an analytic G-invariant neighborhood of the form $G \times_H N$, where N is an analytic slice at f. This implies that any function $\phi \in k[Gf]$ can be extended canonically to a function $\bar{\phi}$ on this neighborhood so that it is constant on the slice at any point of the orbit. This extension is G-equivariant. Therefore we get a canonical G-equivariant embedding of $k[Gf] = k[G/H]$ in $H^0(Z, \mathcal{O}_V^Z)$. The Peter–Weyl theorem (cf. (9)) gives the following explicit G-module decomposition of the ring $k[G/H]$:

$$k[Gf] = k[G/H] = \sum_P P \otimes \bar{P}^H, \tag{10}$$

where P runs over all finite-dimensional G-modules, $\bar{P}^H$ is the sum of all trivial H-submodules of $\bar{P}$, and G acts on the first factor of $P \otimes \bar{P}^H$. This means every G-representation that contains a trivial representation as an H-module occurs in $k[Gf]$, and hence in $H^0(Z, \mathcal{O}_V^Z)$. □

In conjunction with Proposition 5.1, Theorem 5.3 again implies the first criterion of Theorem 5.1. The proof of Theorem 5.3 fails when f is not stable because then it may not be possible to embed $W \subseteq k[Gf] = k[G/H]$ equivariantly within $H^0(Z, \mathcal{O}_V^Z)$ since Luna's slice theorem can fail (cf. Example 2), and so Proposition 5.1 may not be applied to such a W even if it contains a trivial H-module (see also section 5.3).

To construct an obstruction using Proposition 5.1 when f is only partially stable, one needs to calculate explicitly an interesting class of finite-dimensional G-modules that occur within $H^0(Z, \mathcal{O}_V^Z)$, as we did in the stable case (Theorem 5.3). Just as in the completely stable case, we expect that partial stability of f should play a crucial role in this calculation (cf. Theorem 5.4).

When f is stable, the orbit Z is affine, and the stabilizer of f is reductive [32]. In this case, it can be shown that the higher cohomology groups $H^i(Z, \mathcal{O}_V^Z)$, $i > 0$, all vanish; we omit the proof. This statement is analogous to the Cartan–Serre–Grothendick vanishing theorem for the vanishing of higher cohomology of a (quasi-) coherent algebraic sheaf on an affine variety (or more generally on a formal scheme) [18] and the Cartan–Oka vanishing theorem for coherent analytic sheaves on Stein spaces [43]. However, it does not formally follow from the Cartan–Oka vanishing theorem because, in general, $\mathcal{O}_V^Z$ is neither coherent in the sense of Cartan and Oka nor an inverse limit of coherent sheaves.

When f is only partially stable, the higher cohomology groups need not vanish, and they can be expected to play a role in the construction of representation-theoretic obstructions. When f is stable, Theorem 5.3 and Proposition 5.1 together say that $H^0(Z, \mathcal{O}_V^Z)$ contains information in the form of finite-dimensional G-submodules that

puts fairly strong constraints on the stabilizers that can occur in the infinitesimal neighborhood of the orbit of f in V. Analogously, we expect $H^i(Z, \mathcal{O}_V^Z)$, $i \geq 0$, to contain similar information, when f is partially stable, that will constrain the (reductive) stabilizers that can occur in the infinitesimal neighborhood of the orbit of f in V. With this in mind, our approach for constructing representation-theoretic obstructions, when f is partially stable, can be formulated as follows.

1. Prove a generalization of the obstruction-based criterion in Proposition 5.1 to higher cohomology groups.
2. Explicitly calculate an interesting class of finite-dimensional G-submodules that occur within $H^0(Z, \mathcal{O}_V^Z)$, and more generally, in $H^i(Z, \mathcal{O}_V^Z)$, generalizing the method in the proof of Theorem 5.3 for the stable case.
3. Explicitly isolate a G-submodule in this class that satisfies the obstruction criterion for the pair (f, g). Thus, in the setting of Conjecture 4.3, one hopes to get an explicit obstruction this way when m is polynomial in n, thereby proving that the permanent has no polynomial size formula.

Now we elaborate on the main problems that arise in these three steps.

Generalization of the obstruction criterion. The proof of Proposition 5.1 uses evaluation of functions in $H^0(Z, \mathcal{O}_V^Z)$. As it is, this does not work for higher cohomology, since we do not have an appropriate notion of evaluation for elements of higher cohomology representations.

Explicit computation of $H^i(Z, \mathcal{O}_V^Z)$. Here partial stability is expected to be crucial, just as stability was crucial in the computation of $H^0(Z, \mathcal{O}_V^Z)$ based on Luna's slice theorem and Peter–Weyl theorem (Theorem 5.3) when f was stable. Indeed one important feature of our definition of partial stability is the following theorem.

THEOREM 5.4. *Let $\mathcal{F}$ be a coherent (algebraic or analytic) sheaf over the orbit Z. Let $\mathcal{F}^*$ be its direct image over G/P with respect to the projection of the fiber bundle $Z = Gf \to G/P$. Then $H^i(Z, \mathcal{F}) = H^i(G/P, \mathcal{F}^*)$.*

Proof. By the definition of partial stability, the fiber of the bundle $Z \to G/P$ is affine, since it is isomorphic to L/L', where L is a Levi subgroup of P and L' is a stabilizer of f within L that is reductive. Since higher cohomology of a coherent algebraic (or analytic) sheaf over an affine variety vanishes, by the Cartan–Serre theorem [18] (resp., the Cartan–Oka theorem [43]), the result follows by the Leray spectral sequence. □

This indicates why the group action in the vicinity of the orbit of a partially stable point in the null cone ought to have more structure than in the vicinity of an arbitrary point in the null cone. The sheaf $\mathcal{O}_V^Z$ is not coherent. However, the idea in the preceding proof can be pushed towards computation of some interesting families of finite-dimensional G-submodules in $H^i(Z, \mathcal{O}_V^Z)$. This will be reported elsewhere.

Construction of an explicit G-module satisfying an obstruction criterion. This step gives rise to the *subgroup restriction problem* of calculating an explicit decomposition of a G-module W when the action is restricted to a reductive subgroup of G. We have already seen this happen when f is completely stable (cf. Theorem 5.1 and the remark following its proof); the same is expected in the partially stable case. As pointed out in Example 5.2.1, the specific version of the subgroup restriction problem that arises in the context of the determinant vs. permanent problem is the plethysm problem of representation theory, which has no satisfactory solution so far. An explicit combinatorial formula for multiplicities of irreducible Weyl modules within $S_\lambda(V \otimes W)$ and $S_\lambda(S_\mu V)$, based on Weyl's character formula, is known [21].

However, this involves alternating signs, and so it is difficult to ascertain from the formula whether the specific multiplicity is zero or nonzero. What is desirable is a formula for decomposition that does not involve signs, analogous to the Littlewood–Richardson rule for the decomposition of the tensor product of two Weyl modules. Such a decomposition for the plethysm problem is so far not known.

It would be interesting to use the scheme in this section (using, say, the obstruction criterion in Proposition 5.1) on a computer to come up with explicit representation-based obstructions for the pair (f, g) in Conjecture 4.3 for small values of m and n, say, $m = 3$ and $n = 2$. Unfortunately, even for very small values, this computational problem seems quite hard. (The plethysm problem is one of the computational bottlenecks. For $m = 3$ one requires explicit decomposition of $S_\lambda(V \otimes V)$, where $\dim(V) = 9$.)

6. Circuit size. In section 5 we formulated an approach to a well-behaved instance of the orbit-closure (and the stabilizer) problem for a pair of forms (f, g) with nontrivial characteristic stabilizers, wherein f is partially stable. In section 4 we reduced the lower bound problem for the formula size of the permanent to such an instance. Next we shall do the same for certain lower bound questions concerning circuit size, which will include (section 7) the usual P vs. NP question (the stronger nonuniform[4] version) and its arithmetic analogue over an algebraically closed field.

Let $f(X)$ be again any fixed form in the variable vector X of dimension n over an algebraically closed field F. Now we wish to know if it has an arithmetic circuit over F of size l. An arithmetic circuit is a directed acyclic graph. Input variables and constants represent nodes with indegree zero. The remaining nodes are labeled as addition or multiplication nodes. All nodes function as per their labels and the result is output at the unique node with outdegree zero. The size of the circuit is the size of the graph.

The role of the determinant function in section 4 would now be played by a certain function $H(Y)$, which would be complete for the class of polynomials having small circuit size. To define it, consider a generic arithmetic circuit of depth k and width m. It consists of $k + 1$ levels of nodes, numbered 0 to k, each level containing m nodes, except the root level zero, which contains a single output node. Each node in the level $i < k$ is connected to every node in level $i + 1$. Each node u in the input level k is labeled with the variable y^u; the function computed by this node is defined to be y^u. The function $h(u)$ computed by a node u in level $i < k$ is defined to be $\sum_{v,w} y^u_{v,w} h(v)h(w)$, where v and w range over nodes in level $i + 1$ and each $y^u_{v,w}$ is an indeterminate. Let Y be the vector of the variables y^us at the input level k and the variables $y^u_{v,w}$s. Let $H(Y)$ be the function computed at the root level zero. It is a homogeneous form in Y with total degree exponential in k.

Any arithmetic circuit of size r can be obtained from the generic circuit of depth $k \le r$ and width $m \le r$ by specializing the indeterminates $y^u_{v,w}$ to some constants and the indeterminates y^us at the input level to the input parameters or constants. Hence $H(Y)$ is complete for the class of polynomials with small circuits. This can be used to prove the following analogue (Proposition 6.1) of Proposition 4.1. Let V be the space of homogeneous forms in Y with degree equal to that of $H(Y)$. Let $l = O(r^2)$ be the size of Y that corresponds to the generic circuit with depth and width r. We can define, as in section 4, the function ϕ for embedding in V the forms of lower degree

[4]In the nonuniform version, the computational circuits for various input sizes n need not be correlated, whereas in the uniform version, correlation is enforced by requiring a $\log(n)$-space machine that can recognize the circuits for all n.

in the smaller variable vector X of dimension $n < m$: specifically, identify X with some of the input variables in Y and use some remaining variable as a homogenizing parameter. Let $G = SL_l(F)$. Then the following proposition results.

PROPOSITION 6.1. *If $f(X)$ has an arithmetic circuit of size r, then $f^\phi(Y)$ lies in the (projective) Zariski closure $\Delta[H(Y)]$ of the G-orbit of $H(Y)$ in $P(V)$, where $l = O(r^2)$.*

Proof. The proof is similar to that of Proposition 4.1. The role of determinant in that proof is now played by $H(Y)$, which is complete for the class of functions with small circuits. □

The form $H(Y)$ turns out to be good like the determinant function because it has a large nontrivial stabilizer that in a sense characterizes it. Indeed, any automorphism of the generic circuit that fixes the nodes in the input level k gives rise to an element of $SL_l(F)$ that stabilizes $H(Y)$. Quite likely, the stabilizer of $H(Y)$ is precisely the discrete group Q of such automorphisms. It contains k copies of the symmetric group S_m, where k is the depth of the generic circuit and m is its width. The total degree of $H(Y)$ in the variables $y^u_{v,w}$ that occur at level i is $d_i = 2^i$, and its total degree in the input variables y_us is $d_{k+1} = 2^{k+1}$. Therefore $H(Y)$ has multidegree $(d_1, \ldots, d_{k+1})$ in these groups of variables. Let $M(Y)$ be the set of monomials over Y with total degree equal to that of $H(Y)$, i.e., with degree $d_1 + \cdots + d_{k+1}$. It has a natural Q-action induced by that on the generic circuit. Order the monomials in $M(Y)$ lexicographically. We call a monomial maximal if it is the largest monomial in its Q-orbit. Let $M_m(Y) \subseteq M(Y)$ be the subset of maximal monomials. Given any monomial α, let $\hat{\alpha}$ denote it symmetrization $\sum_{\sigma \in Q} \sigma(\alpha)$. Let V be the space of homogeneous forms in Y of total degree equal to that of $H(Y)$. The following is easy to prove.

PROPOSITION 6.2. *The set of forms $\hat{\alpha}$, $\alpha \in M_m(Y)$, is a basis of the space V^Q of Q-invariants in V.*

Unlike the determinant (cf. Proposition 4.2), $H(Y)$ is not a unique form stabilized by Q. However, all forms stabilized by Q are strongly linked to the generic circuit as in Proposition 6.2. In that sense, Q captures the structure of $H(Y)$.

Via Proposition 6.1 we can reduce our lower bound problem for the circuit size of $f(X)$ to an instance of the orbit-closure problem (section 2) with $H(Y)$ and $f^\phi(Y)$ playing the role of $g(Y)$ and $f(Y)$ there. Since $H(Y)$ has a nice stabilizer that captures its structure (Proposition 6.2), this problem in turn can be reduced to an instance of the stabilizer problem (section 2), where we let D be equal to the stabilizer of $H(Y)$. For these problems to be well behaved both $H(Y)$ and $f(X)$ should be nice functions like the determinant and permanent. We have already seen that $H(Y)$ has a nontrivial characteristic stabilizer. The form $f(X)$ should also be nice. This will indeed be the case for the form that arises in the context of the P vs. NP question, as we shall see next.

7. P vs. NP. Before turning to the $P \neq NP$ conjecture, we shall treat its arithmetic implication first. Let X be a variable vector (or matrix) of dimension n. Let $f(X)$ be a polynomial, which is defined in some uniform way for each n, such that, when reduced modulo 2, it gives an NP-complete function in the usual sense. If $P \neq NP$ (the nonuniform version), then $f(X)$, modulo 2, cannot be computed by a boolean circuit of poly(n) size. This implies that $f(X)$ cannot be computed by an integral (arithmetic) circuit of poly(n) size either; otherwise this circuit can be reduced modulo 2 to obtain a boolean circuit of poly(n) size. By arithmetic (integral) implication of the $P \neq NP$ conjecture (nonuniform version), we mean the problem

of showing that such $f(X)$ does not have an integral circuit of poly(n) size. Since it is an implication of the usual nonuniform $P \neq NP$ conjecture, and at the same time does not involve problems of positive characteristic, it is natural to address this arithmetic implication first. In turn, it suffices to consider the complex version of the $P \neq NP$ conjecture, where the problem is to show that $f(X)$ does not have a circuit of poly(n) size defined over $\mathbb{C}$. The complex version trivially implies the arithmetic one, since every integral circuit is also a complex circuit. Hence, the version of the $P \neq NP$ conjecture over $\mathbb{C}$, or more generally, over an arbitrary algebraically closed field F, should be addressed first.

More generally, we can let $f(X)$ be any integral polynomial, which when reduced modulo 2 becomes a hard function NP—this means it belongs to NP but is not expected to be in P (but it need not be NP-complete). The problem then is to show that $f(X)$ does not have a complex, and hence, an integral circuit of poly(n) size.

We begin by instantiating the form $f(X)$ in the preceding section to a certain form $E(X)$ that corresponds to a hard function in NP. Take m-dimensional vector variables $\{X_i^j | 1 \leq j \leq k,\ 1 \leq i \leq m\}$, for some fixed constant $k \geq 3$, say, $k = 3$, for simplicity. Let X be the matrix whose columns consist of these km vectors. For any function $\sigma : \{1, \ldots, m\} \to \{1, \ldots, k\}$, let $\det_\sigma(X)$ denote the determinant of the matrix whose ith column is $X_i^{\sigma(i)}$. Define $E(X) = \prod_\sigma \det_\sigma(X)$, where σ ranges over all such functions. This is well defined over any field. If the field of definition is F_2, then (the negation of) this function is almost NP-complete if $k \geq 3$; see the note at the end of this paper. When the field of definition is F_p, p a fixed prime, we let F be its algebraic closure. The problem then is to show that $E(X)$ does not have an arithmetic circuit of poly(n) size over F. Let $n = km^2$ be the total number entries in X. The stabilizer $K = \mathrm{stab}(E(X))$ of $E(X)$ in $SL_n(F)$ is nice.

PROPOSITION 7.1. *The connected component K_0 of the identity in K is of the form ST, where $S \simeq SL_m(F)$, and $T \simeq (F^*)^{km-1}$ is the $(km-1)$-dimensional algebraic torus. Each $A \in S \simeq SL_m(F)$ corresponds to a linear transformation of the form*

$$X = [\ldots, X_i^j, \ldots] \to [\ldots, AX_i^j, \ldots]. \tag{11}$$

Each element of the torus T corresponds to a linear transformation of the form

$$X = [\ldots, X_i^j, \ldots] \to [\ldots, \lambda_i^j X_i^j, \ldots]. \tag{12}$$

The discrete group K/K_0 contains the wreath product of the alternating group $\mathcal{A}_m$ and $\mathcal{S}_k$. (It acts on X by permuting its columns in the obvious way.)

Proof. That K_0 contains linear transformation of the form (11) or (12) is easy. Since $E(X) = \prod_\sigma \det_\sigma(X)$, where each $\det_\sigma(X)$ is a distinct and irreducible polynomial, each element of K must permute factors, modulo multiplications of factors by constants as in (12). Since the stabilizer of each determinantal factor is known (cf. (4)), the elements in K that fix all factors $\det_\sigma(X)$ are seen to be precisely those of the form in (11).

The elements of K/K_0 that permute the factors $\det_\sigma(X)$ are those that permute the columns of X appropriately. They form a group isomorphic to the wreath product of $\mathcal{A}_m$ and $\mathcal{S}_k$. □

Moreover, the stabilizer K characterizes $E(X)$ in the following sense. Let d denote the total degree of $E(X)$. Let $[E(X)]$ be the set of forms of deg d in the entries of X that are stabilized by K.

PROPOSITION 7.2. *Any form $h(X) \in [E(X)]$ can be expressed as a homogeneous polynomial in the maximal minors of X (which correspond to Plücker coordinates). Among such forms $E(X)$ is the simplest one, in the sense that its expansion in terms of Plücker coordinates contains only one monomial.*

Proof. Since $h(X)$ is stabilized by SL_n, which acts on X by left multiplication, the first assertion follows from classical invariant theory. In arbitrary characteristic, it follows from the result of De Concini and Procesi [8]. The second assertion then follows immediately. □

Let W be the space of forms in X with degree equal to that of $E(X)$.

THEOREM 7.1. *The point $E(X) \in P(W)$ is stable with respect to the action of $G = SL_n(F)$ on $P(W)$. We assume that the characteristic does not divide k, $k-1$, or m.*

This will be proved in section 8. Stability of $E(X)$ is a key fact that connects the P vs. NP question to geometric invariant theory.

Let V be the space of forms in Y as in section 6, where the size of Y is $l > n$. Take a suitable embedding ϕ from W $(P(W))$ to V $(P(V))$ as usual. Then Theorem 7.1 immediately implies the following analogue of Theorem 4.3.

THEOREM 7.2. *$E^\phi(Y) \in P(V)$ is partially stable with respect to the action of $SL_l = SL(Y)$ with defect $\delta = 1$. (We assume that the characteristic does not divide k, $k-1$, or m.)*

Now we make the following analogue of the first assertion in Conjecture 4.3.

CONJECTURE 7.3. *If $l = n^a$, where a is any fixed constant, then $E^\phi(Y)$ cannot lie in the (projective) Zariski closure $\Delta[H(Y)]$ of the $SL_l(F)$-orbit of $H(Y)$ in $P(V)$.*

More generally, let $[H(Y)]$ be the set of forms in the entries of Y, with the same degree as $H(Y)$, that are stabilized by the stabilizer of $H(Y)$. Let $\tilde{\Delta}[H(Y)]$ be the projective closure of $SL_l(F) \cdot [H(Y)]$. A stronger conjecture is the following one.

CONJECTURE 7.4. *If $l = n^a$, where a is any fixed constant, then $E^\phi(Y)$ cannot lie in $\tilde{\Delta}[H(Y)]$.*

By Proposition 6.1, Conjecture 7.3 would imply that $E(X)$ does not have an arithmetic circuit over F of size polynomial in n. The conjecture is not expected to be equivalent to the original lower bound question over F. However, it is "almost" equivalent to it, when F is the field of complex numbers, in the following sense.

PROPOSITION 7.5. *If Conjecture 7.3 were false, then $E(X)$ can be approximated infinitesimally closely by an arithmetic circuit of poly(n) size.*

Proof. The proof is similar to that of Proposition 4.4. □

However, such an approximation is not expected in view of the computational hardness of $E(X)$ over F_2.

Since $H(Y)$ has a nontrivial reductive stabilizer that captures its structure (Propositions 6.2), the orbit-closure problem in Conjecture 7.3 can be reduced to the stabilizer problem. Specifically, we make the following stronger conjecture, which implies Conjecture 7.3 or 7.4.

CONJECTURE 7.6. *The stabilizer of $H(Y)$ cannot occur in the infinitesimal neighborhood of the $SL_l(F)$-orbit of $E^\phi(Y)$ in the affine space V.*

Since $E(X)$ is stable, and both $H(Y)$ and $E(X)$ have nontrivial stabilizers that capture their structure, the approach based on explicit obstructions (section 5) seems appropriate for Conjecture 7.6. In the context of constructing explicit obstructions, the following problems then arise (cf. section 5.5):

1. Explicitly compute finite-dimensional G-submodules of $H^i(Z, \mathcal{O}_V^Z)$, with $E^\phi(Y)$ playing the role of f in section 5.5, and $Z = Gf$. As we mentioned there, the

fact that the orbit Gf is a fiber bundle over G/P, in conjunction with the the Leray spectral sequence, is important in this context.

2. The instances of the subgroup restriction problem that arise in the present context are the following. How does a Weyl module representation $S_\lambda(Y)$ of $SL(Y)$ decompose as a Q-module, where Q is the stabilizer of $H(Y)$? How does a Weyl module representation $S_\mu(X)$ of $SL(X)$ decompose as a K-module, where K is the stabilizer of $E(X)$?

Now, we turn to the actual P vs. NP question (the stronger nonuniform version). Here the field under consideration is no longer algebraically closed. The base field that is actually used for computation in real computers is finite, in fact, F_2, though, without of loss of generality, we can take it to be F_p for any fixed prime. We will reduce the P vs. NP question to a certain parametrized family of the general orbit-closure or stabilizer problems over F, where F is the algebraic closure of F_p.

The starting point is the following analogue of Proposition 6.1. We say that a form is an F_p-form if its coefficients are in F_p. The two forms $h_1(Y), h_2(Y) \in V$ are said to be F_p-equivalent if they are equal for all F_p-vectors Y, i.e., when we let each entry of Y range over F_p.

PROPOSITION 7.7. *If $E(X)$, considered as a function of the F_p-vector X, has an arithmetic circuit over F_p of size of r, then some F_p-form h that is F_p-equivalent to $E^\phi(Y)$ lies in the projective orbit closure $\Delta[H(Y)] \subseteq P(V)$, where the size l of Y is $O(r^2)$.*

Proof. The proof is analogous to that of Proposition 4.1. □

This leads us to the following stronger version of Conjecture 7.6, which would imply that $P \neq NP$.

CONJECTURE 7.8. *If $l = n^a$, where a is any fixed constant, then no F_p-form $h(Y)$ F_p-equivalent to $E^\phi(Y)$ can lie in the (projective) Zariski closure $\Delta[H(Y)]$ of the $SL_l(F)$-orbit of $H(Y)$ in $P(V)$. More strongly, the stabilizer of $H(Y)$ cannot occur in the infinitesimal neighborhood of the $SL_l(F)$-orbit of any such $h(Y)$ in the affine space V.*

For an approach based on explicit obstructions (section 5.5) to be applicable to this conjecture, we would want $h(Y)$ to be a good form like $E^\phi(Y)$. The latter is partially stable (Theorem 7.2), and we conjecture that, in fact, a stronger statement holds.

CONJECTURE 7.9. *Any F_p-form $h(Y)$ F_p-equivalent to $E^\phi(Y)$ is partially stable with respect to the action of $SL_l(F)$ on $P(V)$ with defect $\delta = 1$.*

One can approach other fundamental questions of complexity theory, such as NC^1 vs. NC^2, $P^{\#P}$ vs. NP in a similar fashion by reducing them to various instances of the orbit closure and the stabilizer problem.

8. Stability of $E(X)$. We prove Theorem 7.1 in this section. Let us assume that the characteristic is zero; however, essentially the same proof also works when the characteristic does not divide m, k, or $k-1$. For simplicity, we also assume that $k = 3$.

Let V be a vector space of dimension m. Let I denote the set $\{(i,j) | 1 \leq i \leq 3, 1 \leq j \leq m\}$. Let $X = \{X[i,j] | (i,j) \in I\}$ be a collection of (indeterminate) vectors of V. Let 3^m denote the collection of all functions $\sigma : \{1, \ldots, m\} \to \{1,2,3\}$. For a $\sigma \in 3^m$, let $\det_\sigma(X)$ denote the determinant of the matrix $[X[\sigma(1),1], \ldots, X[\sigma(m),m]]$. With this notation,

$$E(X) = \prod_{\sigma \in 3^m} \det_\sigma(X).$$

Clearly, the action of $SL(V)$ on V stabilizes $E(X)$. In other words, if $A \in SL(V)$ and AX denotes the collection $\{AX[i,j]|(i,j) \in I\}$, then $E(AX) = E(X)$. Let D denote the wreath-product of $\mathcal{S}_m$ with $\mathcal{S}_3$. An element of D is given by a tuple $\beta = (\mu; \alpha_1, \ldots, \alpha_m)$, where $\mu \in \mathcal{S}_m$ and each $\alpha_i \in \mathcal{S}_3$. D acts on I naturally, viz., $\beta = (\mu, (\alpha_r))(i,j) = (\alpha_j(i), \mu(j))$. This action on I extends to X naturally, viz., $\beta(X[i,j]) = X[\beta(i,j)]$, and this action stabilizes $E(X)$ up to sign.

Let $\mathcal{Y}$ be the vector space consisting of formal linear combinations of the symbols $\{y_{ij}|(i,j) \in I\}$. D acts on $\mathcal{Y}$ naturally, viz., $\beta(y_{ij}) = y_{\beta(i,j)}$. We shall examine this representation of D.

Let $e' = \sum_{(i,j)\in I} y_{ij}$ and $f'_j = y_{1j} + y_{2j} + y_{3j}$. Let $\mathcal{E}$ be the span of e' and $\mathcal{F}$ be the span of all vectors of the form $\{f'_r - f'_s|1 \leq r, s \leq m\}$. It is clear that $\mathcal{E}, \mathcal{F}$ are invariant under the action of D. We claim that $\mathcal{E}$ and $\mathcal{F}$ are, in fact, irreducible representations of D. Irreducibility of $\mathcal{E}$ is clear. For $\mathcal{F}$ note that $\mathcal{F} = \{\sum_{r=1}^m c_r f'_r | \sum_{r=1}^m c_r = 0\}$. Note that the action of D on $\mathcal{F}$ reduces to an action of $\mathcal{S}_m$ on $\mathcal{F}$ which is isomorphic to the irreducible representation of $\mathcal{S}_m$ corresponding to the partition $(m-1, 1)$. Let $\mathcal{H}$ be the following subspace of $\mathcal{Y}$:

$$\mathcal{H} = \left\{ \sum c_{ij} y_{ij} | c_{1j} + c_{2j} + c_{3j} = 0 \text{ for all } j \right\}.$$

It is easy to show that $\mathcal{H}$ is also D-irreducible. Thus we see that the D-module $\mathcal{Y}$ splits into irreducible components $\mathcal{Y} = \mathcal{E} \oplus \mathcal{F} \oplus \mathcal{H}$ as D-modules and thus $\mathcal{Y}$ is a multiplicity-free representation of D.

Let D' be the wreath-product of A_m (the alternating group) with $\mathcal{S}_3$. We note that (i) $D' \subset D$ actually stabilizes $E(X)$ and (ii) for $m > 4$ the irreducible D-modules $\mathcal{E}, \mathcal{F}$, and $\mathcal{H}$ are irreducible as D'-modules as well.

Let $h'_{1j} = 2y_{1j} - y_{2j} - y_{3j}$. Define h'_{2j} and h'_{3j} similarly and note that $h'_{ij} \in \mathcal{H}$. We may check that

$$3m.y_{ij} = e' + \sum_{r:r\neq i} (f'_i - f'_r) + m.h'_{ij}. \tag{13}$$

Next, we analyze the stability of the form $E(X)$. If $E(X)$ were unstable, then by Kempf's criteria, there is a one-parameter subgroup $\lambda : k^* \to SL(X)$ "witnessing" its instability and which commutes with the stabilizer of the form $E(X)$. In other words, there would be an $SL(X)$-invariant closed subset of S of forms on X, and a one-parameter subgroup λ commuting with D' above, driving $E(X)$ into S. Since $SL(V)$ acts on X as a diagonal action on $3m$ copies of V, by Schur's lemma, we see that the image of λ must lie in the commutator of this representation of $SL(V)$. This reduces λ to be of the form $\lambda = \gamma \otimes I(V)$, where $I(V)$ is the identity map on V and $\gamma : k^* \to SL(\mathcal{Y})$. Within $SL(\mathcal{Y})$ it must further commute with the D'-representation $\mathcal{Y}$. Since $\mathcal{Y} = \mathcal{E} \oplus \mathcal{F} \oplus \mathcal{H}$ splits as a D'-module, we conclude that D' is actually contained in a suitable parabolic subgroup of $SL(X)$, whence Kempf's first criteria, based on noncontainment of the stabilizer in a proper parabolic group, fails to apply. However, since λ commutes with D', we see that λ can be expressed as

$$\gamma(t)(3m.y_{ij}) = t^a e' + t^b \sum_{r:r\neq i} (f'_i - f'_r) + t^c.m.h'_{ij}. \tag{14}$$

Since $\dim(\mathcal{E}) = 1, \dim(\mathcal{F}) = m-1$ and $\dim(\mathcal{H}) = 2m$, and $\lambda(t) \in SL(X)$ for all t, we have the important relation

$$a + (m-1)b + 2mc = 0. \tag{15}$$

We define the "unprimed" vectors $e(X), f_j(X), h_{ij}$ (X) as the analogues of e', f'_j, and h'_{ij} by replacing the symbol y_{rs} by the vector variable $X[r,s]$. Thus $e(X) = \sum_{(i,j)\in I} X[i,j]$ and $f_j(X) = X[1,j]+X[2,j]+X[3,j]$. The vector $h_{ij}(X)$ is similarly defined. For notational convenience, we drop the argument (X) from $e(X), f_i(X)$, and $h_{ij}(X)$. In this new notation, we have the analogue of (13):

$$3m.X[i,j] = e + \sum_{r:r\neq i} (f_i - f_r) + m.h_{ij}. \tag{16}$$

Let $\chi_i = \chi_i(X)$ denote the vector $\sum_{r:r\neq i}(f_i - f_r)$. In the form $E(X)$, we may substitute the above expression for each $X[i,j]$ and using the multilinearity of the determinant expand $E(X)$ into its "constituents." These constituents may be aggregated by their weights with respect to the one-parameter subgroup λ as follows. Each constituent is a product of determinants with columns from the collection $\{e, (\chi_i), (h_{ij})\}$. If a particular determinant takes for its arguments k_1 copies of e, k_2 elements of (χ_i), and k_3 elements from the set (h_{ij}), (with $k_1 + k_2 + k_3 = m$), then the degree of that determinant may be taken as $k_1a + k_2b + k_3c$. The degree of a product of determinant factors is clearly the sum of the individual degrees. Then we have

$$\lambda(t)(E(X)) = \sum_w t^w E_w(X), \tag{17}$$

where for an integer w the term $E_w(X)$ is the form of degree w. We say that the integer w is a weight in the above expression, if $E_w(X)$ is nonzero. If $E(X)$ were unstable, then there are integers a, b, c satisfying (15) above such that

$$E_w(X) = 0 \text{ for all } w < 0. \tag{18}$$

This is precisely the condition for $\lim_{t\to 0} \lambda(t)(E(X))$ to exist. We shall show that this condition is not possible for any choice of a, b and c as above.

In the expression $E(X) = \sum_w E_w(X)$, there are many terms which vanish. First note that the vector e cannot occur twice inside any nontrivial determinant. Similarly, we have the relation $\sum_{i=1}^m \chi_i = 0$; thus not all the arguments of a nontrivial determinant can be from the set (χ_i). We also note that for any collections $\{v, (w_i)_{i=1}^m, (y_{ij})_{(i,j)\in I}\}$ of vectors in V such that $\sum_i w_i = 0$ and $y_{1j} + y_{2j} + y_{3j} = 0$ for all j, there are vectors $\overline{x} = \{x[r,s] \in V | (r,s) \in I\}$ such that $e(\overline{x}) = v, \chi_i(\overline{x}) = w_i$, and $h_{ij}(\overline{x}) = y_{ij}$.

Therefore suppose that $E(X)$ were unstable and a, b, c were as above.

Case 1. $a, b \geq c$. In this case, (15) tells us that $c \leq 0$. Thus the minimum possible weight is $w_{min} = 3^m.m.c$ coming from terms such as $\prod_{\sigma\in 3^m} \det(h_{\sigma(1),1}, \ldots, h_{\sigma(m),m})$, with a few more terms if $b = c$ or $a = c$.

It is easy to show that $E_{w_{min}} \neq 0$. Select $\overline{x} = (x[r,s])$ such that $e(\overline{x}) = \chi_i(\overline{x}) = 0$ (for all i) and $h_{1j}(\overline{x}) = h_{2j}(\overline{x}) = e_j$ and $h_{3j}(\overline{x}) = -2e_j$, where e_j is the jth "unit" basis vector. For this choice of $\overline{x}$, we see that

$$E_{w_{min}}(\overline{x}) = \prod_{\sigma\in 3^m} \det(h_{\sigma(1),1}, \ldots, h_{\sigma(m),m}) \neq 0.$$

Thus, unless $c = 0$, which in turn implies $a, b, c = 0$, there is a negative weight term, contradicting (18).

Case 2. $\boldsymbol{c \geq a, b.}$ We choose $\overline{x}$ such that $h_{ij}(\overline{x}) = 0$ for all $(i,j) \in I$. Then the minimum-weight term occurs in the expansion of

$$[\det(e(\overline{x}) + \chi_1(\overline{x}), \ldots, e(\overline{x}) + \chi_m(\overline{x}))]^{3^m}$$

(with the relation $\chi_1(\overline{x}) + \cdots + \chi_m(\overline{x}) = 0$).

Since e may occur only once and the χ_i's only $m-1$ times, the only weight possible is $w_{min} = (m-1)b + a$. We may check that $E_{w_{min}}$ is actually nonzero by choosing $\overline{x}$ such that in addition to the requirements above, we have $\chi_i(\overline{x}) = e_i$ for $i = 1, \ldots, m-1$ and $\chi_m(\overline{x}) = -(e_1 + \cdots + e_{m-1})$ and $e(\overline{x}) = e_m$. For this choice of $\overline{x}$, we see that $E_{w_{min}}(\overline{x}) \neq 0$. Since $a + (m-1)b + 2mc = 0$ (see (15)), we have that $a + (m-1)b < 0$. Thus, again, unless $a, b, c = 0$, there is a negative weight term, again contradicting (18).

Case 3. $\boldsymbol{a > c > b.}$ In this case consider the assignment $\overline{x}$ such that (i) $e = 0$, (ii) $h_{1j} = h_{2j} = e_j$, $h_{3j} = -2e_j$ for all j, and (iii) $\chi_i = e_i$ for $i = 1, \ldots, m-1$ with $\chi_m = -(e_1 + \cdots + e_{m-1})$. We see that for any $\sigma \in 3^m$, in the expression

$$\det(\chi_1 + m.h_{\sigma(1),1}, \ldots, \chi_m + m.h_{\sigma(m),m})$$

the minimum-weight term is

$$\sum_{i=1}^{m} m.\det(\chi_1, \ldots, \chi_{i-1}, s.e_i, \chi_{i+1}, \ldots, \chi_m).$$

Here the constant $s = 1$ if $\sigma(i) = 1$ or 2 and -2 otherwise. This expands exactly to either of the numbers m or $-2m$ and is thus nonzero. The weight of this term is minimum possible, which is $(m-1)b + c$. If indeed λ witnesses the instability of $E(X)$ (and therefore satisfies (18)), then $(m-1)b + c \geq 0$. If $c \geq 0$, then this implies that $(m-1)b + 2mc \geq 0$. Since $a > 0$, we have $a + (m-1)b + 2m.c > 0$ which is a contradiction to (15). On the other hand, $c \leq 0$ and $c > b$ together imply $(m-1)b + c < 0$. Thus there is a negative weight nonzero term, contradicting (18).

Case 4. $\boldsymbol{b > c > a.}$ In this case consider the assignment $\overline{x}$ such that (i) $e = e_m$, (ii) $h_{1j} = h_{2j} = e_j$, $h_{3j} = -2e_j$ for all j, and (iii) $\chi_i = 0$ for all i. With $b > c > a$, the minimum weight possible is $(m-1)c + a$. We show that this weight is indeed achieved. We analyze the expression

$$\det(e + m.h_{\sigma(1),1}, \ldots, e + m.h_{\sigma(m),m}).$$

Since $e = e_m$, the only term which survives is

$$\det(m.h_{\sigma(1),1}, \ldots, m.h_{\sigma(m-1),m-1}, e_m).$$

This is clearly nonzero. Thus there is a nonzero term of weight $(m-1)c + a$. Since (a, b, c) is a witness to the instability of $E(X)$, we must have $(m-1)c + a \geq 0$. The only way this can hold is with $c \geq 0$. This implies that $2mc + a = (m-1)c + a + (m+1)c \geq 0$. However, $b > 0$, and consequently $a + (m-1)b + 2mc > 0$, which is untenable, because of (15).

Thus we must have $(a, b, c) = (0, 0, 0)$ which proves the stability of $E(X)$. ∎

Note added in proof. The function $E(X)$ in section 7 is expected to be hard in the following sense. Consider a system of n integral polynomial equations of some constant degree $k \geq 3$ in n variables:

$$g_i(y_1, \ldots, y_n) = 0, \quad 1 \leq i \leq n, \tag{19}$$

where we assume that each g_i is a product of k linear forms. The problem of finding if this system has a nonzero solution over F_p, where $p \leq 2^{\text{poly}(n)}$, is in NP. If g_is are allowed to be nonhomogeneous, it is also NP-complete, even for $p = 2$ and $k = 3$ (reduction from 3-CNF).

Consider the special case of this problem when all g_is are homogeneous. It is expected to be hard, i.e., not expected to be in P for large enough p's; but it is not known if it is NP-complete. Take m-dimensional vector variables $\{X_i^j | 1 \leq j \leq k, 1 \leq i \leq m\}$, where, for any $i, X_i^j, 1 \leq j \leq k$, are the coefficient vectors of the k forms whose product is g_i. Let X be the matrix whose columns consist of these km vectors. Then $E(X)$, modulo F_p, is precisely the resultant of this system (19) with $m = n$.

Acknowledgments. We wish to thank Prof. Madhav Nori, Prof. C. S. Seshadri, and Prof. Burt Totaro for several illuminating discussions and help throughout the course of this research.

REFERENCES

[1] P. Bardsley and R. Richardson, *Étale slices for algebraic transformation groups in characteristic p*, Proc. London Math. Soc. (3), 51 (1985), pp. 295–317.
[2] S. Berkowitz, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.
[3] R. Boppana and M. Sipser, *The complexity of finite functions,* in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 757–804.
[4] A. Borel, *Linear Algebraic Groups*, 2nd ed., Springer-Verlag, New York, 1991.
[5] P. Bürgisser, M. Clausen, and M. Shokhrollahi, *Algebraic Complexity Theory*, Springer-Verlag, Berlin, 1997.
[6] R. Carter, G. Segal, and I. Macdonald, *Lectures on Lie Groups and Lie Algebras*, Cambridge University Press, Cambridge, UK, 1995.
[7] C. De Concini, D. Eisendud, and C. Procesi, *Young diagrams and determinantal varieties*, Invent. Math., 56 (1980), pp. 129–165.
[8] C. De Concini and C. Procesi, *A characteristic-free approach to invariant theory*, Adv. Math., 21 (1976), pp. 330–354.
[9] C. De Concini and C. Procesi, *Complete symmetric varieties,* in Invariant Theory, Lecture Notes in Math. 996, Springer-Verlag, Berlin, 1983, pp. 1–44.
[10] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.
[11] S. Cook, *The classification of problems which have fast parallel algorithms*, in Foundations of Computation Theory, Springer-Verlag, Berlin, 1983, pp. 78–93.
[12] E. Formanek and C. Procesi, *Mumford's conjecture for the general linear group*, Adv. Math., 19 (1976), pp. 292–305.
[13] W. Fulton, *Introduction to Toric Varieties*, Princeton University Press, Princeton, NJ, 1993.
[14] W. Fulton and J. Harris, *Representation Theory*, Springer-Verlag, New York, 1991.
[15] J. von zur Gathen, *Permanent and determinant,* Linear Algebra Appl., 96 (1987), pp. 87–100.
[16] M. Grotschel, L. Lovász, and A. Schriver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
[17] W. Haboush, *Reductive groups are geometrically reductive*, Ann. of Math. (2), 102 (1975), pp. 67–83.
[18] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, New York, 1977.
[19] D. Hilbert, *Über die vollen Invariantensysteme*, Math. Ann., 42 (1893), pp. 313–373.
[20] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[21] A. Klymyk, *Multiplicities of weights of representations and multiplicities of representations of semisimple Lie algebras*, Sov. Math. Dokl., 8 (1967), pp. 1531–1534.
[22] H. Kraft, P. Slodowy, and P. Springer, *Algebraische Transformationsgruppen aund Invarianttentheorie*, DMV Sem. 15, Birkhäuser, Basel, 1993.
[23] G. Kempf, *Instability in invariant theory*, Ann. of Math. (2), 108 (1978), pp. 299–316.
[24] V. Lakshmibai, C. Musili, and C. S. Seshadri, *Geometry of G/P*, Bull. Amer. Math. Soc. (N.S.), 1 (1979), pp. 432–435.
[25] A. Levin, *Universal sequential search problems*, Problemy Peredachi Informatsii, 9 (1973) (in Russian).
[26] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[27] D. Luna, *Slices Etales*, Bull. Soc. Math. France Mem., 33 (1973), pp. 81–105.
[28] D. Luna and Th. Vust, *Plongements d'espaces homogenes*, Comment. Math. Helv., 58 (1983), pp. 186–245.
[29] G. Margulis, *Explicit constructions of concentrators*, Problemy Inf. Trans., 9 (1973), pp. 325–332.
[30] H. Minc, *Permanents*, Addison-Wesley, Reading, MA, 1978.
[31] M. Nagata, *Polynomial Rings and Affine Spaces*, CBMS Reg. Conf. Ser. Math. 37, AMS, Providence, RI, 1978.
[32] Y. Matsushima, *Espaces homogènes de Stein des groupes de Lie complexes*, Nagoya Math. J., 16 (1960), pp. 205–218.
[33] K. Mulmuley, *Lower bounds in a parallel model without bit operations*, SIAM J. Comput., 28 (1999), pp. 1460–1509.
[34] K. Mulmuley and M. Sohoni, *Geometric Complexity Theory* II*: Explicit Obstructions*, manuscript.
[35] K. Mulmuley, U. Vazirani, and V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–113.
[36] D. Mumford, *Geometric Invariant Theory*, Ergeb. Math. Grenzgeb. (3) 34, Springer-Verlag, Berlin, 1965.
[37] D. Mumford, J. Fogarty, and F. Kirwan, *Geometric Invariant Theory*, Springer-Verlag, Berlin, 1994.
[38] V. Popov and E. Vinberg, *Invariant Theory*, in Algebraic Geometry IV, Encyclopaedia Math. Sci. 55, Springer-Verlag, Berlin, 1991.
[39] A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
[40] C. Seshadri, *Theory of moduli*, in Algebraic Geometry, AMS, Providence, RI, 1975, pp. 263–304.
[41] S. Smale, *Mathematical problems for the next century*, in Mathematics: Frontiers and Perspectives, AMS, Providence, RI, 2000, pp. 271–294.
[42] T. Springer, *Linear algebraic groups*, in Algebraic Geometry IV, Encyclopaedia Math. Sci. 55, Springer-Verlag, Berlin, 1991.
[43] H. Grauert and R. Remmert, *Theory of Stein Spaces*, Springer-Verlag, Berlin, 1979.
[44] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[45] L. Valiant, *Completeness classes in algebra*, in Proceedings of the Eleventh ACM Symposium on Theory of Computing, 1979.
[46] H. Weyl, *The Classical Groups. Their Invariants and Representations*, Princeton University Press, Princeton, NJ, 1939.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 527–549

ON-LINE LOAD BALANCING IN A HIERARCHICAL SERVER TOPOLOGY*

AMOTZ BAR-NOY†, ARI FREUND‡, AND JOSEPH (SEFFI) NAOR‡

Abstract. In a hierarchical server environment jobs are to be assigned in an on-line fashion to a collection of servers which form a hierarchy of capability: each job requests a specific server meeting its needs, but the system is free to assign it either to that server or to any other server higher in the hierarchy. Each job carries a certain load, which it imparts to the server it is assigned to. The goal is to find a competitive assignment in which the maximum total load on a server is minimized.

We consider the linear hierarchy in which the servers are totally ordered in terms of their capabilities. We investigate several variants of the problem. In the *unweighted* (as opposed to *weighted*) problem all jobs have unit weight. In the *fractional* (as opposed to *integral*) model a job may be assigned to several servers, each receiving some fraction of its weight. Finally, *temporary* (as opposed to *permanent*) jobs may depart after being active for some finite duration of time. We show an optimal e-competitive algorithm for the unweighted integral permanent model. The same algorithm is $(e+1)$-competitive in the weighted case. Its fractional version is e-competitive even if temporary jobs are allowed. For the integral model with temporary jobs we show an algorithm which is 4-competitive in the unweighted case and 5-competitive in the weighted case. We show a lower bound of e for the unweighted case (both integral and fractional). This bound is valid even with respect to randomized algorithms. We also show a lower bound of 3 for the unweighted integral model when temporary jobs are allowed.

We generalize the problem and consider hierarchies in which the servers form a tree. In the tree hierarchy, any job assignable to a node is also assignable to the node's ancestors. We show a deterministic algorithm which is 4-competitive in the unweighted case and 5-competitive in the weighted case, where only permanent jobs are allowed. Randomizing this algorithm improves its competitiveness to e and $e+1$, respectively. We also show an $\Omega(\sqrt{n})$ lower bound when temporary jobs are allowed.

Key words. on-line algorithms, load balancing, hierarchical servers, temporary jobs, resource procurement

AMS subject classifications. 68Q10, 68Q17, 68Q25, 68W99

PII. S0097539798346135

1. Introduction.

One of the most basic on-line load-balancing problems is the following. Jobs arrive one at a time and each must be scheduled on one of n servers. Each job has a certain load associated with it and a subset of the servers on which it may be scheduled. The goal is to assign jobs to servers so as to minimize the *cost* of the assignment, defined as the maximum load on a server.

The nature of the load-balancing problem considered here is on-line: decisions must be made without any knowledge of future jobs, and previous decisions may not be revoked. We compare the performance of an on-line algorithm to the performance of an optimal off-line scheduler—one that knows the entire sequence of jobs in advance. The efficacy parameter of an on-line scheduler is its *competitive ratio*, roughly defined

*Received by the editors October 26, 1998; accepted for publication (in revised form) March 29, 2001; published electronically September 26, 2001. An extended abstract of this paper appeared in *Proceedings of the 7th European Symposium on Algorithms*, Lecture Notes in Comput. Sci. 1643, Springer-Verlag, Berlin, pp. 77–88.

http://www.siam.org/journals/sicomp/31-2/34613.html

†AT&T Shannon Lab, 180 Park Ave., P.O. Box 971, Florham Park, NJ 07932 (amotz@research.att.com). This author was on leave from the Electrical Engineering Department, Tel Aviv University, Tel Aviv 69978, Israel.

‡Computer Science Department, Technion, Haifa 32000, Israel (arief@cs.technion.ac.il, naor@cs.technion.ac.il).

as the maximum ratio, taken over all possible sequences of jobs, between the cost incurred by the algorithm and the cost of an optimal assignment.

1.1. The hierarchical servers problem. In the *hierarchical servers* problem the servers form a hierarchy of capability; a job which may run on a given server may also run on any server higher in the hierarchy. We consider the *linear hierarchy* in which the servers are numbered 1 through n, and we imagine them to be physically ordered along a straight line running from left to right, with server 1 leftmost and server n rightmost. Leftward servers are more capable than rightward ones. We say that servers $1, \ldots, s$ are *to the left* of s, and that servers $s+1, \ldots, n$ are *to the right* of s.

The input is a sequence of jobs, each carrying a positive *weight* and *requesting* one of the servers. A job requesting server s can be assigned to any of the servers to the left of s. These servers are the job's *eligible* servers. The assignment of a job with weight w to server s increases the *load* on s by w (initially, all loads are 0). We use the terms "job" and "request" interchangeably. The *cost* of a given assignment is $COST = \max_s \{l_s\}$, where l_s is the load on server s. We use OPT for the cost of an optimal off-line assignment. An algorithm is *c-competitive* if there exists some $b > 0$, independent of the input, such that $COST \leq c \cdot OPT + b$ for all input sequences. For scalable problems (such as ours) the additive factor b may be ignored in lower bound constructions.

We consider variants, or *models*, of the problem according to three orthogonal dichotomies. In the *integral* model each job must be assigned in its entirety to a single server; in the *fractional* model a job's weight may be split among several eligible servers. In the *weighted* model jobs may have arbitrary positive weights; in the *unweighted* model all jobs have unit weight. Our results for the fractional model hold for both the unweighted and weighted cases, so we do not distinguish between the unweighted fractional model and the weighted fractional model. Finally, *permanent* jobs continue to load the servers to which they are assigned indefinitely; *temporary* jobs are *active* for a finite duration of time, after which they depart. The duration for which a temporary job is active is *not* known upon its arrival. We may allow temporary jobs or we may restrict the input to permanent jobs only. When temporary jobs are allowed, the cost of an assignment is defined as $COST = \max_t \max_s \{l_s(t)\}$, where $l_s(t)$ is the load on server s at time t. The version of the problem which we view as basic is the weighted integral model with permanent jobs only.

A natural generalization of the problem is for the servers to form a (rooted) *tree hierarchy*; a job requesting a certain server may be assigned to any of its ancestors in the tree. The various models pertain to this problem as well.

The hierarchical servers problem is an important practical paradigm. It captures many interesting applications from diverse areas such as assigning classes of service to calls in communication networks, routing queries to hierarchical databases, signing documents by ranking executives, and upgrading classes of cars by car rental companies.

From a theoretical point of view, the hierarchical servers problem is also interesting by virtue of its relation to the problem of *related machines* [3]. In this problem all servers are eligible for every job, but they may have different *speeds*; assigning a job of weight w to a server with speed v increases its load by w/v. Without loss of generality, assume $v_1 \geq v_2 \geq \cdots \geq v_n$, where v_i is the speed of server i. Consider a set of jobs to be assigned at a cost bounded by C, and let us focus on a particular job whose weight is w. To achieve $COST \leq C$ we must refrain from assigning this

job to any server i for which $w/v_i > C$. In other words, there exists a rightmost server to which we may assign the job. Thus, restricting the cost induces eligibility constraints similar to those in the hierarchical servers problem. Some of the ideas developed in the context of the hierarchical servers problems are applicable to the problem of related machines, leading to better bounds for that problem [10].

1.2. Background. Graham [16] explored the assignment problem where each job may be assigned to any of the servers. He showed that the greedy algorithm has competitive ratio $2 - \frac{1}{n}$. Later work [8, 9, 17, 2] investigated the exact competitive ratio achievable for this problem for general n and for various special cases. The best results to date for general n are a lower bound of 1.852 and an upper bound of 1.923 [2].

Over the years many other load-balancing problems were studied; see [4, 20] for surveys. The assignment problem in which arbitrary sets of eligible servers are allowed was considered by Azar, Naor, and Rom [7]. They showed upper and lower bounds of $\Theta(\log n)$ for several variants of this problem. Permanent jobs were assumed. Subsequent papers generalized the problem to allow temporary jobs; in [5] a lower bound of $\Omega(\sqrt{n})$ and an upper bound of $O(n^{2/3})$ were shown. The upper bound was later tightened to $O(\sqrt{n})$ [6].

The *related machines* problem was investigated by Aspnes et al. [3]. They showed an 8-competitive algorithm based on the doubling technique. This result was improved by Berman, Charikar, and Karpinski [12], who showed a more refined doubling algorithm that is $3 + \sqrt{8} = 5.828$-competitive. By randomizing this algorithm, they were able to improve the bound to 4.311. They also showed lower bounds of 2.438 (deterministic) and 1.837 (randomized). The randomized bound was recently improved to 2 [14]. Azar et al. [6] generalized the problem to allow temporary jobs. They showed a deterministic upper bound of 20 (which implies a randomized upper bound of $5e \approx 13.59$) and a lower bound of 3. The upper bounds were later improved to $6 + 2\sqrt{5} \approx 10.47$ (deterministic) and 9.572 (randomized) [10].

The *resource procurement* problem was defined and studied by Kleywegt et al. [18] independently of our work. In this problem jobs arrive over (discrete) time, each specifying a deadline by which it must complete, and all jobs must be executed on a single server. We can view this as a problem of assigning permanent jobs to parallel servers if we think of the time slots as servers. In fact, the problem is equivalent to the following variant of the hierarchical servers problem. The model considered is the fractional model with permanent jobs only. The input consists of precisely n jobs. The jth job to arrive specifies a server $s_j \leq n - j + 1$; the servers eligible for the job are $s_j, s_j + 1, \ldots, n - j + 1$. In addition, the on-line nature of the problem is less demanding. The scheduler need not commit to the full assignment of a job immediately on its arrival. Rather, when the jth job arrives, it must decide what fraction of each of the first j jobs to assign to server $n - j + 1$. Kleywegt et al. [18] developed a lower bound technique similar to ours and were able to establish a lower bound of 2.51 by analytic and numerical means. They also described a 3.45-competitive algorithm.

1.3. Our results. A significant portion of our work is devoted to developing a continuous framework in which we recast the problem. The continuous framework is not a mere relaxation of the problem's discrete features. Rather, it is a fully fledged model in which a new variant of the problem is defined. The advantage of the continuous model lies in the ability to employ the tools of infinitesimal calculus, making analysis much easier.

In section 2 we use the continuous model to design an optimal e-competitive algorithm. Surprisingly, this algorithm operates counterintuitively; the weight distribution of an assigned job is biased to the *left*, i.e., more weight ends up on the leftward servers. We show a general procedure for transforming an algorithm for the continuous model into an algorithm for the fractional model. We also show a general procedure for transforming an algorithm for the fractional model into an algorithm for the integral model. Thus we get an e-competitive algorithm for the fractional model and an algorithm which is, respectively, e and $(e+1)$-competitive for the unweighted integral and weighted integral models. The former algorithm admits temporary jobs; the latter does not. Our upper bound of e also applies to the resource procurement problem of Kleywegt et al. [18] by virtue of Theorem 2 in their paper. Thus we improve their best upper bound of 3.45.

In section 3 we develop a procedure for deriving lower bounds in the context of the continuous model. The construction of the continuous model is rather unconventional in its not being a generalization of the discrete model. In fact, on the surface of things, the two models seem incomparable, albeit analogous. At a deeper level, though, it turns out that the continuous model is actually a special case of the discrete model, making lower bounds obtained in the continuous model valid in the discrete setting as well. This makes the upper bound all the more intriguing, as it is developed in the continuous framework and transported back to the discrete model. The lower bounds obtained with our procedure are also valid in the discrete models (fractional as well as integral), even in the unweighted case with permanent jobs only and even with respect to randomized algorithms. Using our procedure we find that e is a tight lower bound. Since our lower bound technique is the same as the one used (independently) by Kleywegt et al. [18] in the context of the resource procurement problem, our lower bound of e applies to that problem as well, and it improves their best lower bound of 2.51. Thus our work solves this problem completely by demonstrating a tight bound of e.

In section 4 we consider temporary jobs in the integral model. We show a doubling algorithm that is 4-competitive in the unweighted case and 5-competitive in the weighted case. We also show a deterministic lower bound of 3.

Section 5 extends the problem to the tree hierarchy. We show an algorithm which is, respectively, 4-, 4-, and 5-competitive for the fractional, unweighted integral, and weighted integral models. Randomizing this algorithm improves its competitiveness to e, e, and $e+1$, respectively. We show lower bounds of $\Omega(\sqrt{n})$ for all models, both deterministic and randomized, when temporary jobs are allowed.

The effect of restricting the sets of eligible servers in several other ways is discussed in section 6. In the three cases we consider, we show a lower bound of $\Omega(\log n)$. For example, this lower bound holds in the case where the servers form a circuit (or a line) and eligible servers must be contiguous. Note that since these problems are all special cases of the problem considered in [7], an upper bound of $O(\log n)$ is immediate.

2. Upper bounds. In this section we show an algorithm whose respective versions for the fractional, unweighted integral, and weighted integral models are e, e, and $(e+1)$-competitive. The fractional version admits temporary jobs; the integral versions do not. We build up to the algorithm by introducing and studying the *semicontinuous* model and the class of *memoryless* algorithms. We begin with the optimum lemma, which characterizes OPT in terms of the input sequence.

2.1. The optimum lemma. For the fractional and unweighted integral models the lemma provides an exact formula for OPT. For the weighted integral case it gives

a 2-approximation.[1] The optimum lemma is a recurrent theme in our exposition.

For a given input sequence and a given server s, denote by W_s the total weight of jobs requesting servers to the left of s, and let $\mu_s = W_s/s$. Define $H = \max_s \{\mu_s\}$. Clearly, H is a lower bound on OPT, and in the unweighted integral model we can tighten this to $\lceil H \rceil$. In addition, the maximum weight of a job in the input sequence, denoted $w_{\max}$, is also a lower bound on OPT in the integral model.

Turning to upper bounds on OPT, let us say that a given server is *saturated* at a given moment if its load is at least H. For the integral model, consider an algorithm that assigns each job to its rightmost unsaturated eligible server. This algorithm treats the jobs in an on-line fashion but requires advance knowledge of H so it is off-line. Clearly, if an unsaturated eligible server can always be found, then $COST < H + w_{\max}$. We claim that this is indeed the case. To see this, suppose that when some job of weight w arrives, all of its eligible servers are saturated. Let s be maximal such that the servers to the left of s are all saturated. By the maximality of s, the jobs assigned to the left of s must have all requested servers to the left of s. Since their total weight is at least $s \cdot H$, we have $W_s \geq s \cdot H + w > s \cdot H$, a contradiction.

For the fractional model we modify the above algorithm as follows. When a job of weight w arrives, we assign it as follows: let s be the rightmost unsaturated eligible server and let $\delta = H - l_s$, where l_s is the current load on s. If $\delta \geq w$, we assign the job in its entirety to s. Otherwise, we split the job and assign δ units of weight to s and treat the remainder recursively as a new job to be assigned. This algorithm achieves $COST \leq H$.

The optimum lemma summarizes these results.

LEMMA 1 (optimum lemma).

- *In the fractional model,* $OPT = H$.
- *In the unweighted integral model,* $OPT = \lceil H \rceil$.
- *In the weighted integral model,* $\max\{H, w_{\max}\} \leq OPT < H + w_{\max}$.

2.2. Memoryless algorithms. A *memoryless* algorithm is an algorithm that assigns each job independently of previous jobs. Of course, memoryless algorithms are only of interest in the fractional model, which is the model we are going to consider here. We focus on a restricted type of memoryless algorithms, namely, *uniform* algorithms. Uniform memoryless algorithms are instances of the generic algorithm shown below; each instance is characterized by a function $u : \mathcal{N} \to (0, 1]$ satisfying $u(1) = 1$.

Algorithm GenericUniform

When a job of weight w requesting server s arrives, do:

1. $r \leftarrow w$; $i \leftarrow s$.
2. While $r > 0$:
   3. Assign $a = \min\{w \cdot u(i), r\}$ units of weight to server i.
   4. $r \leftarrow r - a$.
   5. $i \leftarrow i - 1$.

[1] It is unreasonable to expect an easily computable formula for OPT in the weighted integral model, for that would imply a polynomial-time solution for the NP-hard problem PARTITION

The algorithm starts with the server requested by the job and proceeds leftward as long as the job is not fully assigned. The fraction of the job's weight assigned to server i is $u(i)$, unless $w \cdot u(i)$ is more than the remainder of the job when i is reached. The condition $u(1) = 1$ ensures that the job will always be fully assigned by the algorithm.

Note that the assignment generated by a uniform memoryless algorithm is independent of both the number of servers and the order of jobs in the input. Moreover, any collection of jobs with total weight w, requesting some server s, may be replaced by a single request of weight w for s. We therefore assume that exactly one job requests each server (we allow jobs of zero weight) and that the number of servers is infinite. We denote the weight of the job requesting server s by w_s.

Consider a job of weight w requesting a server to the right of a given server s. If the requested server is close to s the job will leave $w \cdot u(s)$ units of weight on s regardless of the exact server requested. At some point, however, the job's contribution to the load on s will begin to diminish as the distance of the request from s grows. Finally, if the request is made far enough away, it will have no effect on s. We denote by p_s the point beyond which the effect on s begins to diminish and by p'_s the point at which it dies out completely:

$$p_s = \max\left\{ s' \left| \sum_{i=s}^{s'} u(i) \le 1 \right. \right\}, \qquad p'_s = \max\left\{ s' \left| \sum_{i=s+1}^{s'} u(i) < 1 \right. \right\}.$$

Note that p_s and p'_s may be undefined, in which case we take them to be infinity. We are interested only in functions u satisfying $p_s < \infty$ for all s.

The importance of p_s lies in the fact that the load on s due to jobs requesting servers in the range $s, \ldots, p_s$ is simply $u(s)$ times the total weight of these jobs. The following lemma and corollary are not difficult to prove.

LEMMA 2 (worst case lemma). *Let $\mathcal{A}$ be a uniform memoryless algorithm. The following problem,*

> *Given $K > 0$ and some server s, find an input sequence I that maximizes the load on s in $\mathcal{A}$'s assignment subject to $OPT = K$,*

is solved by $I = \langle w_1, w_2, \ldots \rangle$, where

$$w_i = \begin{cases} 0, & 1 \le i < p_s, \\ p_s K, & i = p_s, \\ K, & p_s < i \le p'_s, \\ 0, & i > p'_s \text{ (if } p'_s < \infty), \end{cases}$$

and l_s—the resultant load on s—satisfies $p_s \cdot Ku(s) \le l_s \le p'_s Ku(s)$.

COROLLARY 3. *Let $\mathcal{A}$ be a uniform memoryless algorithm, and let $C_{\mathcal{A}}$ be the competitive ratio of $\mathcal{A}$. Then $\sup_i \{p_i u(i)\} \le C_A \le \sup_i \{p'_i u(i)\}$.*

2.3. The semicontinuous model. In both the fractional and the integral versions of the problem, the servers and the jobs are discrete objects. We therefore refer to these models as the *discrete* models. In this section we introduce the *semicontinuous* model, in which the servers are made continuous. In section 3 we define the *continuous* model by making the jobs continuous as well.

The semicontinuous model is best understood through a physical metaphor. Consider the bottom of a vessel filled with some nonuniform fluid applying varying degrees of pressure at different points. The force acting at any single point is zero, but any

region of nonzero area suffers a net force equal to the integral of the pressure over the region. Similarly, in the semicontinuous model we do not talk about individual servers; rather, we have a continuum of servers, analogous to the bottom of the vessel. An arriving job is analogous to a quantity of fluid which must be added to the vessel. The notions of *load* and *weight* become divorced; *load* is analogous to pressure and *weight* is analogous to force.

Formally, the *server interval* is $[0, \infty)$, to which jobs must be assigned. Job j has weight w_j, and it requests the point $s_j > 0$ in the server interval. The assignment of job j is specified by an integrable function $g_j : [0, \infty) \to [0, \infty)$ satisfying

1. $\int_0^{s_j} g_j(x)\,dx = w_j$;
2. $x > s_j \Rightarrow g_j(x) = 0$;
3. g_j is continuous from the right at every point.

The *full assignment* is $g = \sum_j g_j$. For a given full assignment g, the *load* l_I on an interval $I = (x, x + \Delta)$, where $\Delta > 0$, is defined as $l_I = \frac{1}{\Delta} \int_x^{x+\Delta} g(z)\,dz$—the mean weight density over I. The load at a point x is defined as $l_x = \lim_{\Delta \to 0} \left\{ l_{(x,x+\Delta)} \right\} = g(x)$. (We introduce the notation l_x for consistency with previous notation.) The cost of the assignment is $COST = \sup_x \{l_x\}$.

LEMMA 4 (optimum lemma: semicontinuous model). *Let $W(x)$ be the total weight of requests made to the left of x (including x itself) and $H = \sup_{x>0} \{W(x)/x\}$. Then $OPT = H$.*

Proof. The lower bound is trivial. For the upper bound, let $x_1 \le x_2 \le \cdots$ be the points requested by the jobs and rearrange the jobs such that the jth job requests x_j. The idea is to pack the jobs (in order) in a rectangle extending from the left end of the server interval. Let $\alpha_j = \sum_{i=1}^{j-1} w_i/H$ for all j and consider the following assignment:

$$g_j(x) = \begin{cases} H, & x \in [\alpha_j, \alpha_j + w_j/H), \\ 0 & \text{otherwise.} \end{cases}$$

This assignment clearly attains $COST = H$. It follows from the definition of H that $\alpha_j + w_j/H \le x_j$ for all j, which is sufficient for the assignment's validity. ∎

We adapt the definition of uniform memoryless algorithms to the semicontinuous model. In this model a uniform algorithm is characterized by a function $u : (0, \infty) \to (0, \infty)$ as follows. For a given point $x > 0$, let $q(x)$ be the point satisfying the equation $\int_{q(x)}^{x} u(z)\,dz = 1$. Then the assignment of job j is

$$g_j(x) = \begin{cases} w_j u(x), & q(s_j) \le x < s_j, \\ 0 & \text{otherwise.} \end{cases}$$

For $q(x)$ and g_j to be defined properly we must require that $\int_0^{\epsilon} u(x)\,dx = \infty$ for all $\epsilon > 0$. (Otherwise, the algorithm may fail to fully assign jobs requesting points close to 0.) Note that the load at 0 is always zero.

For a given point $x > 0$, we define $p(x)$ as the point such that $\int_x^{p(x)} u(z)\,dz = 1$. If $p(x)$ does not exist, then the algorithm's competitive ratio is unbounded, as demonstrated by the request sequence consisting of $m \to \infty$ jobs, each of unit weight, where the jth job requests the point $s_j = j$. For this sequence, $l_x = m \cdot u(x)$, whereas $OPT = 1$. We shall therefore allow only algorithms satisfying $\int_M^{\infty} u(x)\,dx = \infty$ for all $M \ge 0$.

The semicontinuous model has the nice property that p_s and p'_s, which were disparate in the discrete model, fuse into a single entity, $p(s)$. The worst case lemma and its corollary become the following lemma.

LEMMA 5 (worst case lemma: semicontinuous model). *Let $\mathcal{A}$ be a uniform memoryless algorithm defined by $u(x)$. The following problem,*

> *Given $K > 0$ and some point $s > 0$ in the server interval, find an input sequence that maximizes the load at s in $\mathcal{A}$'s assignment, subject to $OPT = K$,*

is solved by a single job of weight $p(s)K$ requesting the point $p(s)$, and the resultant load at s is $p(s)Ku(s)$.

COROLLARY 6. *The competitive ratio of $\mathcal{A}$ is $\sup_x \{p(x)u(x)\}$.*

2.4. An e-competitive algorithm for the semicontinuous model. Consider Algorithm Harmonic, the uniform memoryless algorithm defined by $u(x) = 1/x$. Let us calculate $p(x)$:

$$1 = \int_x^{p(x)} \frac{dz}{z} = \ln \frac{p(x)}{x},$$

$$p(x) = ex.$$

Thus, the competitive ratio of Algorithm Harmonic is $\sup_x \left\{ex\frac{1}{x}\right\} = e$.

2.5. Application to the discrete models. Having devised a competitive algorithm for the semicontinuous model, we wish to import it to the discrete model. We start by showing how to transform any algorithm for the semicontinuous model into an algorithm for the (discrete) fractional model. Following that, we show how any algorithm for the fractional model may be transformed into an algorithm for the integral models.

Semicontinuous to fractional. Let I be an input sequence for the fractional model. If we treat each server as a point on $(0, \infty)$, that is, we view a request for server s as a request for the point s, then we can view I as a request sequence for the semicontinuous model as well. By the respective optimum lemmas (Lemmas 1 and 4), the value of OPT is the same for both models.

Let $\mathcal{A}$ be a c-competitive online algorithm for the semicontinuous model. Define algorithm $\mathcal{B}$ for the fractional model as follows. When job j arrives, $\mathcal{B}$ assigns $\int_{i-1}^{i} g_j(x)\,dx$ units of weight to server i, for all i, where g_j is the assignment function generated by $\mathcal{A}$ for the job. Clearly, the cost incurred by $\mathcal{B}$ is bounded by the cost incurred by $\mathcal{A}$. Thus, $\mathcal{B}$ is c-competitive.

An important observation is that if $\mathcal{A}$ is memoryless, then so is $\mathcal{B}$. Thus, even if temporary jobs are allowed, the assignment generated by $\mathcal{B}$ will be c-competitive at all times, compared to an optimal (off-line) assignment of the active jobs.

We give the algorithm thus derived from Algorithm Harmonic the name FractionalHarmonic.

PROPOSITION 7. *Algorithm FractionalHarmonic is e-competitive even when temporary jobs are allowed.*

Fractional to integral. Let $\mathcal{A}$ be an algorithm for the fractional model. Define algorithm $\mathcal{B}$ for the integral model (both weighted and unweighted) as follows. As jobs arrive, $\mathcal{B}$ keeps track of the assignments $\mathcal{A}$ would make. A server is said to be *overloaded* if its load in $\mathcal{B}$'s assignment exceeds its load in $\mathcal{A}$'s assignment. When a job arrives, $\mathcal{B}$ assigns it to the rightmost eligible server which is not overloaded (after $\mathcal{A}$ is allowed to assign the job).

PROPOSITION 8. *Whenever a job arrives at least one of its eligible servers is not overloaded.*

Proof. Denote by $l_i^A(j)$ and $l_i^B(j)$ the load on server i after job j is assigned by $\mathcal{A}$ and $\mathcal{B}$, respectively. When job j is considered for assignment by $\mathcal{B}$, server i is overloaded iff $l_i^B(j-1) > l_i^A(j)$. Define $A_i(j) = \sum_{k=1}^{i} l_k^A(j)$ and $B_i(j) = \sum_{k=1}^{i} l_k^B(j)$. We claim that for all j,

1. when job j arrives, server 1 (which is eligible) is not overloaded;
2. $A_i(j) \geq B_i(j)$ for all i.

The proof is by induction on j. The claim is clearly true for $j = 1$. Consider some job $j > 1$ whose weight is w. We have $l_1^A(j) = A_1(j) \geq A_1(j-1) \geq B_1(j-1) = l_1^B(j-1)$, where the second inequality is justified by the induction hypothesis. Thus, server 1 is not overloaded. It remains to show that for all i, $A_i(j) \geq B_i(j)$. Let a be the rightmost server to which algorithm $\mathcal{A}$ assigns part of job j, i.e., $a = \max\{s \mid A_s(j) > A_s(j-1)\}$. Let b be the server to which $\mathcal{B}$ assigns the job. By the induction hypothesis, $A_i(j-1) \geq B_i(j-1)$ for all i. Clearly, $B_i(j) \leq B_i(j-1) + w$ for all i, and $B_i(j) = B_i(j-1)$ for $i < b$. Also, $A_i(j) \geq A_i(j-1)$ for all i, and $A_i(j) = A_i(j-1) + w$ for $i \geq a$. Thus, $A_i(j) \geq B_i(j)$ for $i \geq a$ and for $i < b$.

Assuming $b < a$, we still have to prove the claim for $i \in \{b, \ldots, a-1\}$. Algorithm $\mathcal{B}$ assigns job j to server b and not to one of the servers $b+1, \ldots, a$, all of which are eligible and to the right of b. It must therefore be the case that $l_k^A(j) < l_k^B(j-1)$ for $b < k \leq a$. Thus, for $i \in \{b, \ldots, a-1\}$,

$$\begin{aligned} A_i(j) &= A_a(j) - \sum_{k=i+1}^{a} l_k^A(j) = A_a(j-1) + w - \sum_{k=i+1}^{a} l_k^A(j) \\ &> A_a(j-1) + w - \sum_{k=i+1}^{a} l_k^B(j-1) \geq B_a(j-1) + w - \sum_{k=i+1}^{a} l_k^B(j-1) \\ &= B_i(j-1) + w = B_i(j). \end{aligned}$$

The second inequality is justified by the induction hypothesis. ∎

Let $w_{\max}(j)$ be the maximum weight of a job among the first j jobs. Algorithm $\mathcal{B}$ maintains $l_i^B(j) \leq l_i^A(j) + w_{\max}(j)$ for all i and j. In the unweighted case we have $w_{\max} = 1$ and in the weighted case $w_{\max} \leq OPT$. By the optimum lemma (Lemma 1) the value of OPT in the integral model is at least as high as its value in the fractional model. Thus if $\mathcal{A}$ is c-competitive, then $\mathcal{B}$ is c-competitive in the unweighted case and $(c+1)$-competitive in the weighted case.

We give the algorithm thus derived from Algorithm FractionalHarmonic the name IntegralHarmonic.

PROPOSITION 9. *Algorithm IntegralHarmonic is e-competitive in the unweighted case and $(e+1)$-competitive in the weighted case.*

3. Lower bounds. In this section we devise a technique for proving lower bounds in the limit $n \to \infty$. The bounds obtained are valid in both the fractional and integral models, even in the unweighted case. In fact, they remain valid even in the presence of randomization with respect to oblivious adversaries. Using this technique, we obtain a tight constant lower bound of e. The success of our approach is facilitated by transporting the problem from the discrete setting into a *continuous* model, in which both jobs and servers are continuous.

3.1. A simple lower bound. We consider the fractional model, restricting our attention to *right-to-left* input sequences, defined to be sequences in which for all $i < j$, all requests for server j are made before any request for server i. We further restrict our attention to sequences in which each server is requested exactly once. (We allow jobs of zero weight.)

FIG. 1. (a) *Histogram of job weights;* $w = 12$. (b) *Histogram of* kh_i*;* k=3.

Let $\mathcal{A}$ be a k-competitive algorithm. For a given right-to-left input sequence, denote by w_s the weight of the job requesting server s and by l_s the load on server s at a given moment. Suppose the first $n-i+1$ jobs (culminating with the request for server i) have been assigned by $\mathcal{A}$. Recall the definition of H in the optimum lemma (Lemma 1); denote by h_i the value of H with respect to these jobs. Since $\mathcal{A}$ is k-competitive, the loads must obey $l_s \le kh_i$ for all s. For $j \ge i$, define $h_{i,j} = \frac{1}{j}\sum_{s=i}^{j} w_s$. Then $h_i = \max_{i \le j \le n}\{h_{i,j}\}$.

Now consider the specific input sequence defined by $w_1 = \cdots = w_n = w$ for some $w > 0$. For this sequence, $h_i = (n-i+1)w/n$ for all i. Thus, after the first job is assigned we have $l_n \le kw/n$. After the second job is handled we have $l_{n-1} \le 2kw/n$, but $l_n \le kw/n$ still holds because the new job could not be assigned to server n. In general, after the request for server s is processed, we have $l_i \le (n-i+1)kw/n$ for all $i \ge s$. Noting that the total weight of the jobs in the input equals the total load on the servers once the assignment is complete, we get

$$nw = \sum_{i=1}^{n} w_i = \sum_{i=1}^{n} l_i \le \sum_{i=1}^{n} (n-i+1)\frac{kw}{n} = \sum_{j=1}^{n} j\frac{kw}{n} = \frac{kwn(n+1)}{2n}.$$

Hence, $k \ge \lim_{n\to\infty} 2 \cdot \frac{n}{n+1} = 2$.

3.2. Discussion. Figure 1 depicts the request sequence and the resultant kh_i's in histogram-like fashion with heights of bars indicating the respective values. The bars are of equal width, so we can equivalently consider their area rather than height. To be precise, let us redraw the histograms with bars of width 1 and height equal to the numerical values they represent. Then, the total weight to be assigned is the total area of the job bars, and the total weight actually assigned is bounded from above by the total area of the kh_i bars. Now, instead of drawing a histogram of kh_i, let us draw a histogram of h_i. The lower bound is found by solving

$$\textit{total area of job bars} \le k \cdot \textit{total area of } h_i \textit{ bars},$$

$$k \ge \frac{\textit{total area of job bars}}{\textit{total area of } h_i \textit{ bars}}.$$

Note that if we multiply the weights of all jobs by some constant $c > 0$, the heights of both the job bars and the h_i bars will increase by a factor of c, leaving the area ratio intact. Similarly, we can express the scaling of job weights by scaling the width of the bars in both histograms. This, too, has no effect on the resultant ratio. Thus we can express the entire procedure in geometric terms as follows. Select an "input" histogram in which the width of each bar is $\frac{1}{n}$. Let $h_{i,j}$ be the area of bars i through j divided by j/n (the width of j bars), and let $h_i = \max_{i \le j \le n}\{h_{i,j}\}$. (We divide the area by j/n rather than j because h_i is the *height* of the bar whose *area*

equals the value of OPT for the first $n - i + 1$ jobs.) Divide the area of the input histogram by the area of the h_i histogram (drawn to the same scale) to obtain a lower bound. The scaling of the histograms allows us to keep considering finite areas as n goes to infinity. This forms the link between the discrete model and the continuous model, which we introduce next.

3.3. The continuous model. The continuous model is motivated by the observation that the analysis suggested in the previous section tends to be exceedingly difficult for all but the simplest of input histograms. We turn to the continuous model in order to avail ourselves of the machinery of infinitesimal calculus. The continuous model differs from the semicontinuous model introduced in section 2 in two ways. Instead of the infinite server interval, we use a finite interval $[0, S]$, and, more importantly, jobs in the continuous model are not discrete; rather, we have a continuous job flow arriving over time.

It is possible to define a general continuous model in which the arrival of jobs over time is described by a function of place (in the server interval) and time. Although this model is an interesting mathematical construction in its own right, we focus here on a more restricted model—one that allows only the equivalent of right-to-left sequences. Formally, the input is a *request function*, which is an integrable nonnegative real function $f(x)$ defined on the server interval $[0, S]$. The interpretation of f is by means of integration, i.e., $\int_{x_0}^{x_1} f(x)\,dx$ is the total amount of weight requesting points in the interval $[x_0, x_1]$. The underlying intuition is that the request flow is right-to-left in the sense that the infinitesimal request for point x is assumed to occur at "time" $S - x$. Assignment are continuous too; an assignment is described by an *assignment function*, which is an integrable nonnegative real function $g(x)$ on $[0, S)$ that (1) is continuous from the right at every point and (2) satisfies $\int_{x_0}^{S} g(x)\,dx \le \int_{x_0}^{S} f(x)\,dx$ for all $x_0 \in [0, S)$ with equality for $x_0 = 0$. An on-line algorithm in this model is an algorithm which, given $f(x)$, outputs $g(x)$ such that for all $x \in [0, S)$, $g(x)$ on the interval $[x, S)$ is independent of $f(x)$ outside that interval. The definition of *load* and *cost* are the same as in the semicontinuous model.

LEMMA 10 (optimum lemma: continuous model).

$$OPT = \sup_{z \in (0,S]} \left\{ \frac{1}{z} \int_0^z f(x)\,dx \right\}.$$

Proof. Let $H = \sup_{z \in (0,S]} \left\{ \frac{1}{z} \int_0^z f(x)\,dx \right\}$ and $W = \int_0^S f(x)\,dx$. Clearly, $OPT \ge H$. In addition, the assignment function

$$g(x) = \begin{cases} H, & x \le W/H, \\ 0, & x > W/H \end{cases}$$

achieves cost H. □

Let us adapt the lower bound procedure from section 3.2 to the continuous model. Consider a request function $f(x)$ and the corresponding assignment $g(x)$ generated by some k-competitive on-line algorithm. We wish to bound the value of $g(x)$ at some fixed point a. Define a new request function

$$f_a(x) = \begin{cases} f(x), & a \le x \le S, \\ 0 & \text{otherwise.} \end{cases}$$

Define, for $b > a$, $h_a(b) = \frac{1}{b} \int_a^b f(x)\,dx$. Define $h(a) = \sup_{b \in (a,S]} \{h_a(b)\}$. Then OPT with respect to f_a equals $h(a)$. (Note the analogy with $h_{i,j}$ and h_i in the discrete

model.) Define $W = \int_0^S f(x)\,dx$ and $W' = \int_0^S h(a)\,da$. The value of g in $[a, S)$ must be the same for f and f_a, as g is produced by an on-line algorithm; thus $g(a) \le kh(a)$. This is true for all a, hence,

$$W = \int_0^S f(x)\,dx = \int_0^S g(x)\,dx \le k\int_0^S h(a)\,da = kW',$$

from which the lower bound W/W' is readily obtained.

For certain request functions we can simplify the procedure. If $f(x)$ is a continuous monotonically decreasing function tending to 0 at some point $x_0 \ge S$ (where $x_0 = \infty$ is allowed), and we use f's restriction to $[0, S]$ as a request function, then we have the following shortcut to $h(a)$. Solve

$$0 = \frac{d}{db}h_a(b) = \frac{bf(b) - \int_a^b f(x)\,dx}{b^2} \iff f(b) = \frac{1}{b}\int_a^b f(x)\,dx$$

for b, and let $b(a)$ be the solution. The following is easy to justify:

$$h(a) = \frac{1}{b(a)}\int_a^{b(a)} f(x)\,dx = f(b(a)).$$

Note that if $x_0 > S$, this simplified procedure may return values for $b(a)$ outside the server interval $[0, S]$. In this case the true value of $h(a)$ is less than the value computed, leading to a less tight, but still valid, lower bound. We can therefore use the simplified method without being concerned by this issue. Also, it is sometimes more convenient to assume that the server interval, rather than being finite, is $[0, \infty)$. This too can be easily seen to make no difference, at least as far as using the simplified procedure is concerned.

Example. Let $f(x) = e^{-kx}$ with server interval $[0, \infty)$ for some $k > 0$. Then $W = \int_0^\infty e^{-kx}\,dx = \frac{1}{k}$. We find it easier to solve $e^{-kb} = \frac{1}{b}\int_a^b e^{-kx}\,dx$ for a rather than b:

$$a = b - \frac{1}{k}\ln(kb+1),$$

$$da = \left(1 - \frac{1}{kb+1}\right)db.$$

Thus, setting $z = kb + 1$,

$$\begin{aligned} W' &= \int_0^\infty h(a)\,da = \int_0^\infty f(b)\,da = \int_0^\infty e^{-kb}\left(1 - \frac{1}{kb+1}\right)db \\ &= \frac{1}{k}\int_1^\infty e^{1-z}\left(1 - \frac{1}{z}\right)dz = \frac{1}{k}\left(1 - e\int_1^\infty \frac{e^{-z}}{z}\,dz\right) = \frac{1}{k}(1 - eE_1(1)), \end{aligned}$$

where $E_n(x) = \int_1^\infty t^{-n}e^{-xt}\,dt$ is the familiar exponential integral function [1]. The lower bound obtained is therefore

$$\frac{W}{W'} = \frac{1}{1 - eE_1(1)} \approx 2.4773. \qquad \square$$

CLAIM 11. *A lower bound of e can be obtained with our method by considering the request function* $e^{-kx^{1/k}}$ *in the limit* $k \to \infty$ *with server interval* $[0, \infty)$.

Proof. For convenience we consider only integral values of k. We start by reviewing some elementary facts concerning gamma functions [1]. The gamma function is defined by $\Gamma(a) = \int_0^\infty t^{a-1}e^{-t}\,dt$, and it can be shown that for positive integer a, $\Gamma(a) = (a-1)!$. The incomplete gamma function is defined by $\Gamma(a,z) = \int_z^\infty t^{a-1}e^{-t}\,dt$. Integrating by parts we obtain the recurrence $\Gamma(a+1,z) = a\Gamma(a,z) + z^a e^{-z}$. We also need Stirling's approximation: $k! = \sqrt{2\pi k}\,(k/e)^k e^{o(1)}$, which implies $\lim_{k\to\infty}(k/e)^k/k! = 0$. Finally, consider the integral $\int_\alpha^\beta e^{-kx^{1/k}}\,dx$. Substituting $z = kx^{1/k}$ gives $\int_\alpha^\beta e^{-kx^{1/k}}\,dx = \frac{1}{k^{k-1}}\int_{k\alpha^{1/k}}^{k\beta^{1/k}} z^{k-1}e^{-z}\,dz$. Thus, for finite α and β, $\int_\alpha^\beta e^{-kx^{1/k}}\,dx = \left(\Gamma(k, k\alpha^{1/k}) - \Gamma(k, k\beta^{1/k})\right)/k^{k-1}$.

Returning to our problem,

$$W = \int_0^\infty e^{-kx^{1/k}}\,dx = \frac{1}{k^{k-1}}\int_0^\infty z^{k-1}e^{-z}\,dz = \frac{\Gamma(k)}{k^{k-1}} = \frac{(k-1)!}{k^{k-1}}.$$

The relation between a and b is given by

$$be^{-kb^{1/k}} = \int_a^b e^{-kx^{1/k}}\,dx = \frac{\Gamma(k, ka^{1/k}) - \Gamma(k, kb^{1/k})}{k^{k-1}}.$$

Let $r = ka^{1/k}, t = kb^{1/k}$. Then $da = \frac{r^{k-1}}{k^{k-1}}\,dr$. Substituting r and t in the previous equation and simplifying gives

$$k\Gamma(k,t) + t^k e^{-t} = k\Gamma(k,r).$$

Applying the recurrence to both sides of this equation and rearranging terms yields

$$\Gamma(k+1,r) - \Gamma(k+1,t) = r^k e^{-r}.$$

We also get (directly)

$$\Gamma(k,r) - \Gamma(k,t) = \left(\frac{t}{k}\right)t^{k-1}e^{-t}.$$

Let us explore the relationship between r and t. Clearly, $r \le t$. It is easy to see that the function $x^k e^{-x}$ increases in $[0,k)$ and decreases in (k,∞). Thus, referring to Figure 2(a), we see that for $r \ge k$, $\Gamma(k+1,r) - \Gamma(k+1,t) = \int_r^t x^k e^{-x}\,dx$ is the area of the region marked "X" between r and t, and $r^k e^{-r}$ is the area of the dotted rectangle between r and $r+1$. Since both areas are equal and $x^k e^{-x}$ decreases in this region, $t \ge r+1$.

Next consider $r \le k-2$. Referring to Figure 2(b) and applying the same reasoning we see that $t \le r+1 \le k-1$. Let us now consider the function $x^{k-1}e^{-x}$. Its maximum occurs at $x = k-1$. Thus, since $t \le k-1$, the function increases in the interval $[r,t]$. Referring to Figure 2(c) and appealing to $\Gamma(k,r) - \Gamma(k,t) = \left(\frac{t}{k}\right)t^{k-1}e^{-t}$, we see that $r \le t - \frac{t}{k} = t(1-\frac{1}{k})$.

To summarize,

$$r \ge k \Rightarrow\ \ t \ge r+1 \Rightarrow\ \ \ e^{-t} \le \frac{1}{e}e^{-r},$$
$$r \le k-2 \Rightarrow t \ge \frac{r}{1-1/k} \Rightarrow e^{-t} \le e^{-\frac{r}{1-1/k}}.$$

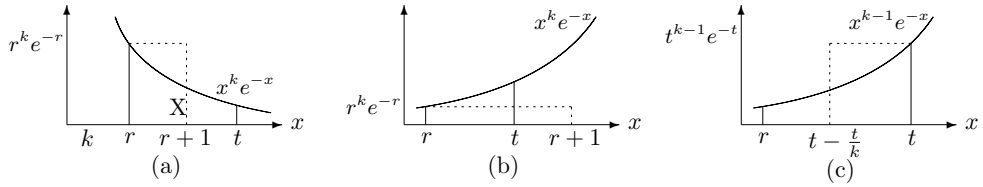


FIG. 2. *The relationship between r and t. The graphs are not drawn to the same scale.* (a) *Using $x^k e^{-x}$ to show $r \geq k \Rightarrow t \geq r+1$.* (b) *Using $x^k e^{-x}$ to show $r \leq k-2 \Rightarrow t \leq k-1$.* (c) *Using $x^{k-1}e^{-x}$ to show $t \leq k-1 \Rightarrow r \leq t - t/k$.*

By differentiating we get $\max_{x \geq 0} \left\{x^{k-1}e^{-x}\right\} = \left(\frac{k-1}{e}\right)^{k-1}$, which implies

$$\int_{k-2}^{k} r^{k-1}e^{-t}\,dr \leq \int_{k-2}^{k} r^{k-1}e^{-r}\,dr \leq 2\left(\frac{k-1}{e}\right)^{k-1}.$$

Putting all the pieces together,

$$\begin{aligned}
\frac{W'}{W} &= \frac{1}{W}\int_0^\infty e^{-kb^{1/k}}\,da = \frac{1}{W}\int_0^\infty e^{-t}\,da = \frac{k^{k-1}}{(k-1)!}\int_0^\infty e^{-t}\frac{r^{k-1}}{k^{k-1}}\,dr \\
&= \frac{1}{(k-1)!}\left(\int_0^{k-2} r^{k-1}e^{-t}\,dr + \int_{k-2}^{k} r^{k-1}e^{-t}\,dr + \int_k^\infty r^{k-1}e^{-t}\,dr\right) \\
&\leq \frac{1}{(k-1)!}\left(\int_0^{k-2} r^{k-1}e^{-\frac{r}{1-1/k}}\,dr + 2\left(\frac{k-1}{e}\right)^{k-1} + \frac{1}{e}\int_k^\infty r^{k-1}e^{-r}\,dr\right).
\end{aligned}$$

Substituting $z = \frac{r}{1-(1/k)}$ in the first integral gives

$$\int_0^{k-2} r^{k-1}e^{-\frac{r}{1-(1/k)}}\,dr = \left(1-\frac{1}{k}\right)^k \int_0^{k-\frac{1}{1-(1/k)}} z^{k-1}e^{-z}\,dz \leq \frac{1}{e}\int_0^k z^{k-1}e^{-z}\,dz.$$

Thus,

$$\frac{W'}{W} \leq \frac{1}{(k-1)!}\left(\frac{1}{e}\int_0^\infty z^{k-1}e^{-z}\,dz + 2\left(\frac{k-1}{e}\right)^{k-1}\right) = \frac{1}{e} + 2\frac{\left(\frac{k-1}{e}\right)^{k-1}}{(k-1)!}.$$

Hence, $\lim_{k\to\infty}\frac{W'}{W} \leq \frac{1}{e}$, and we obtain the lower bound $\lim_{k\to\infty}\frac{W}{W'} \geq e$. ∎

3.4. Application to the discrete models. Returning to the discrete setting, we claim that lower bounds obtained using our method in the continuous model apply in the discrete models as well. This is intuitively correct, since the continuous model may be viewed as the limiting case of the integral models with $n \to \infty$ and arbitrarily long input sequences. We omit the proofs for lack of space.

We also claim that these lower bounds are valid even against randomized algorithms. Whereas typical deterministic lower bound constructions are adversarial, that is, a different input sequence is tailored for each algorithm, our lower bound technique provides a single sequence fit for all algorithms. Consequently, the bounds we derive are also valid for randomized algorithms. This can be seen easily, either directly or via Yao's principle (see, e.g., [13]).

4. Temporary jobs. In this section we allow temporary jobs in the input. We restrict our attention to the integral model, as we have already seen (in section 2.5) an optimal e-competitive algorithm for the fractional model that admits temporary jobs. We present an algorithm that is 4-competitive in the unweighted case and 5-competitive in the weighted case. We also show a lower bound of 3 for the unweighted integral model.

Recall the definition of H in the optimum lemma (Lemma 1). Consider the jobs which are active upon job j's arrival (including job j) and denote by $H(j)$ the value of H defined with respect to these jobs. A server is *saturated* on the arrival of job j if its load is at least $kH(j)$, where k is a constant to be determined later.

Algorithm PushRight

Assign each job it to its rightmost unsaturated eligible server.

PROPOSITION 12. *If $k \geq 4$, then whenever a job arrives, at least one of its eligible servers is unsaturated. Thus, by taking $k = 4$ Algorithm PushRight becomes 4 (resp., 5)-competitive in the unweighted (resp. weighted) models.*

Proof. We begin by considering the properties of certain sequences of numbers whose role will become evident later. Consider the infinite sequence defined by the recurrence $a_{i+2} = k(a_{i+1} - a_i)$ with initial conditions $a_0 = 0$ and $a_1 = 1$. We are interested in the values of k for which this sequence increases monotonically. Solving the recurrence reveals the following.

- If $k = 4$, then $a_i = i2^{i-1}$.
- If $k > 4$, then $a_i = \frac{\lambda_1^i - \lambda_2^i}{\sqrt{k^2-4k}}$, where $\lambda_1 > \lambda_2$ are the two roots of the quadratic polynomial $\lambda^2 - k\lambda + k$.
- If $k < 4$, then $a_i = \frac{\mathrm{Im}(\lambda^i)}{\mathrm{Im}(\lambda)}$, where $\lambda = \frac{1}{2}(k + \sqrt{k^2 - 4k})$.

It is easy to see that the sequence increases monotonically in the first two cases but not in the third.

Now consider some infinite sequence $\{s_i\}$ obeying $s_0 = 0$, $s_1 > 0$, and $s_{i+2} \geq k(s_{i+1} - s_i)$ for all i. It is not difficult to show that if k is chosen such that $\{a_i\}$ increases monotonically, i.e., $k \geq 4$, then $s_{i+2} \geq k(s_{i+1} - s_i) > s_{i+1} > s_i$ for all i.

Returning to our proposition, let $k \geq 4$ and suppose that some job arrives, only to find all of its eligible servers saturated. Let j_0, be the first such job, and let s_1 be the server requested by it. We show how to construct two sequences, $\{s_i\}_{i=0}^{\infty}$ and $\{j_i\}_{i=0}^{\infty}$, with the following properties.

1. $s_0 = 0$, $s_1 > 0$, and $s_{i+2} \geq k(s_{i+1} - s_i)$ for all i.
2. For all i, although the servers $s_i + 1, \ldots, s_{i+1}$ are all eligible for job j_i, the algorithm does not assign this job to the right of s_i.
3. The jobs $\{j_i\}_{i=1}^{\infty}$ are all distinct and they arrive before j_0.

Property 3 states that job j_0 is preceded by an infinite number of jobs, yielding a contradiction.

We have already defined j_0, s_0, and s_1. Having defined $s_0, \ldots, s_{i+1}$ and $j_0, \ldots, j_i$ we define s_{i+2} and j_{i+1} as follows. Property 1 implies that $s_i < s_{i+1} < k(s_{i+1} - s_i)$. By property 2 we know that when job j_i arrives, the total weight of active jobs assigned to servers $s_i + 1$ through s_{i+1} is at least $k(s_{i+1} - s_i)H(j_i)$. By the optimum lemma (Lemma 1), at least one of these jobs must have requested a server whose number is at least $k(s_{i+1} - s_i)$. Define j_{i+1} as any one such job and s_{i+2} as the server it requests. □

4.1. A lower bound. We show a lower bound of 3 on the competitive ratio of deterministic algorithms. We use essentially the same construction as was used in [6] in the context of *related machines*. Some of the details are different, though, owing to the different nature of the two problems. For completeness, we present the construction in full detail.

We consider the unweighted model. To motivate the construction, suppose the value of OPT is known in advance and consider the algorithm that assigns each job to its rightmost eligible server whose current load is less than $k \cdot OPT$, where $k \geq 1$ is an appropriately chosen constant. We design an input sequence targeted specifically at this algorithm. As we shall see, the lower bound of 3 obtained using this input sequence is valid for *any* on-line algorithm; we use only this algorithm to motivate our construction.

Recall that a *right-to-left* input sequence is a sequence such that for all $i < j$, all requests for server j are made before any of the requests for server i. Our input sequence will be right-to-left. For now, we focus on the principles at the cost of rigor. We shall refer to either of the servers s, $s+1$, or $s-1$ simply as server s. We will also refer to server "x" without worrying about the fact that x may be noninteger. To simplify matters, we design the sequence with $OPT = 1$. (This will be changed later.) Figure 3 depicts the first few steps in the ensuing construction.

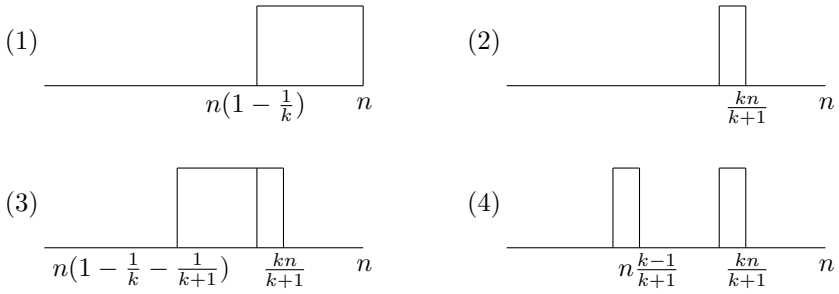


FIG. 3. *The first two rounds in the input sequence.*

We start by making requests to server n. Since OPT is already known to the algorithm, we lose nothing by making n requests, which is the maximum permitted by $OPT = 1$. The algorithm assigns these jobs to servers $n(1-\frac{1}{k}), \ldots, n$. We now remove αn jobs. (α will be determined shortly.) Naturally, the jobs we remove will be the ones assigned rightmost by the algorithm. The remaining $n(1-\alpha)$ jobs will be the ones assigned to servers $n(1-\frac{1}{k}), \ldots, n(1-\frac{1}{k}) + \frac{n(1-\alpha)}{k}$. The adversary assigns these jobs to servers $\alpha n, \ldots, n$. The value of α is determined by our desire that the remaining jobs be assigned by the algorithm strictly to the left of their assignment by the adversary. To that end, we select $\alpha = \frac{k}{k+1}$, which solves the equation $n(1-\frac{1}{k}) + \frac{n(1-\alpha)}{k} = \alpha n$.

We proceed with a second round of jobs. The logical choice is to request server $\alpha n = \frac{k}{k+1} n$. We make αn requests, again, the maximum permitted by $OPT = 1$. The algorithm assigns these jobs in the range $n(1-\frac{1}{k}-\frac{1}{k+1}), \ldots, n(1-\frac{1}{k})$. We terminate βn of these jobs, the ones assigned rightmost by the algorithm. The remaining $n(\alpha-\beta)$ jobs are assigned by the adversary in the range $\beta n, \ldots, \alpha n$ and by the algorithm in the range $n(1-\frac{1}{k}-\frac{1}{k+1}), \ldots, n(1-\frac{1}{k}) - \frac{\beta n}{k}$. To determine β we solve $n(1-\frac{1}{k}) - \frac{\beta n}{k} =$

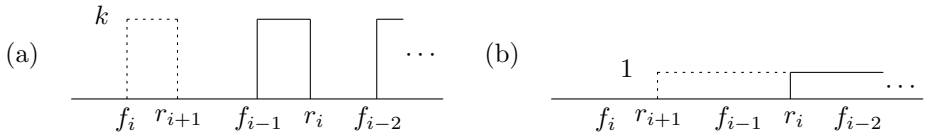


FIG. 4. *The ith round. Solid rectangles represent the assignment of the active jobs at the beginning of the round; the dotted rectangle represents the assignment of the jobs that arrive in the ith round and do not subsequently depart.* (a) *The algorithm's assignment.* (b) *The adversary's assignment.*

βn, arriving at $\beta = n\frac{k-1}{k+1}$.

To generalize the procedure, note that the number of jobs that arrive in a given round is chosen equal to the number of jobs that depart in the round preceding it. Let us introduce some notation. Denote by r_i the server to which requests are made in the ith round and by f_i the leftmost server to which any of these jobs gets assigned by the algorithm. We have already chosen $r_1 = n$ and $r_2 = \frac{k}{k+1}n$, and we have seen that $f_1 = n(1-\frac{1}{k})$ and $f_2 = n(1-\frac{1}{k}-\frac{1}{k+1})$. Define $f_0 = n$. For the ith round of jobs, suppose the following two conditions hold at the end of round $i-1$ (see Figure 4).

1. In the adversary's assignment the active jobs are all assigned in the range $r_i, \ldots, n$.

2. In the algorithm's assignment the active jobs that arrived in round $i-1$ occupy servers $f_{i-1}, \ldots, r_i$, and no jobs are assigned to the left of f_{i-1}.

In the ith round, r_i requests are made to server r_i. They are assigned by the algorithm in the range $f_i, \ldots, f_{i-1}$. Thus,

$$f_i = f_{i-1} - \frac{1}{k}r_i.$$

Next, r_{i+1} of these jobs depart, where r_{i+1} is chosen such that the $r_i - r_{i+1}$ remaining jobs occupy servers $f_i, \ldots, r_{i+1}$. Thus, $k(r_{i+1} - f_i) = r_i - r_{i+1}$, or, equivalently,

$$r_{i+1} = \frac{k}{k+1}f_{i-1}.$$

The actual lower bound construction follows. Let $\mathcal{A}$ be an on-line algorithm purporting to be k-competitive for some $k < 3$. Without loss of generality, k is a rational number arbitrarily close to 3. Consider the two sequences $\{\rho_i\}$ and $\{\varphi_i\}$ defined simultaneously below.

$$\begin{aligned} \varphi_0 &= 1, \\ \rho_1 &= 1, \\ \rho_{i+1} &= \frac{k}{k+1}\varphi_{i-1} \qquad (i = 1, 2, \ldots), \\ \varphi_{i+1} &= \varphi_i - \frac{1}{k}\rho_{i+1} \qquad (i = 0, 1, \ldots). \end{aligned}$$

By substituting $\frac{k}{k+1}\varphi_{i-1}$ for ρ_{i+1} in the second recurrence, we get

$$\varphi_{i+1} = \varphi_i - \frac{1}{k+1}\varphi_{i-1}.$$

It can be shown (see [6]) that there exists a minimal integer p such that $\varphi_p < 0$ and that ρ_i and φ_i are rational for all i. The number of servers we use is n such that $n\rho_i$ and $n\varphi_i$ are integers for $1 \le i \le p$. Define $r_i = n\rho_i$ and $f_i = n\varphi_i$. The recurrences defining $\{\varphi_i\}$ and $\{\rho_i\}$ hold for $\{f_i\}$ and $\{r_i\}$ as well. Let c be any positive integer; we construct an input sequence of unit weight jobs such that $OPT = c$.

1. cr_1 jobs request server r_1.
2. For $i = 2, \ldots, p$:
    3. Of the cr_{i-1} jobs which have requested server r_{i-1}, the cr_i jobs that were assigned rightmost by $\mathcal{A}$ depart. (Ties are broken arbitrarily.)
    4. cr_i new jobs request server r_i.

The lower bound proof proceeds as follows. (We omit the proofs for lack of space.)

For the input sequence to be well defined we must have $r_{i+1} < r_i$ for all $1 \le i < p$.

CLAIM 13. *For $1 \le i < p$, $f_p < 0 \le f_i < r_{i+1} < f_{i-1} \le r_1$ (see Figure 4).*

Denote by J_i the set of jobs requesting server r_i and by J_i' the set of the cr_{i+1} jobs in J_i that eventually depart. Let $W(s,t)$ be the number of active jobs assigned to the left of server s at time t and denote by t_i the moment in time immediately prior to the arrival of the jobs J_i.

CLAIM 14. *$OPT \le c$.*

Observe that the recurrence $f_{i+1} = f_i - \frac{1}{k} r_{i+1}$ is equivalent to $r_i = k(f_{i-1} - f_i)$.

CLAIM 15. *Suppose algorithm $\mathcal{A}$ is k-competitive. Then $W(r_i, t_i) \ge ck(r_i - f_{i-1})$ for all $1 \le i \le p$.*

COROLLARY 16. *Algorithm $\mathcal{A}$ is not k-competitive.*

5. The tree hierarchy. In this section we study a generalization of the problem in which the servers form a rooted tree. A job requesting some server t may be assigned to any of t's ancestors in the tree.

Let us introduce some terminology. A server is said to be *lower* than its proper ancestors. The *trunk* defined by a set of servers U is the set $U \cup \{\, s \mid s \text{ is an ancestor of some server in } U \,\}$. The servers eligible for a given job form a path which is also a trunk. We refer to it interchangeably as the job's *eligible path* or *eligible trunk*.

For a given input sequence, denote by W_T the total weight of jobs requesting servers in trunk T, and let $\mu_T = W_T/|T|$. Define $H = \max\{\, \mu_T \mid T \text{ is a trunk} \,\}$. Denote by $w_{\max}$ the maximum weight of a job in the sequence. Note the analogy with the linear hierarchy. The following lemma can be proved in a manner similar to the proof of the optimum lemma for the linear hierarchy (Lemma 1).

LEMMA 17 (optimum lemma: tree hierarchy).

- *In the fractional model, $OPT = H$.*
- *In the unweighted integral model, $OPT = \lceil H \rceil$.*
- *In the weighted integral model, $\max\{H, w_{\max}\} \le OPT < H + w_{\max}$.*

5.1. A doubling algorithm. The off-line algorithm used in the proof of the optimum lemma (Lemma 17) is nearly a valid on-line algorithm; its only off-line feature is the requirement that the value of H be known at the outset. Thus, employing the standard doubling technique (see, e.g., [4]) we can easily construct an on-line algorithm which is respectively 4-, 4-, and 7-competitive for the fractional, unweighted integral, and weighted integral models. The algorithm we present here is based on the more sophisticated doubling approach pioneered in [12]. It is 4-, 4-, and 5-competitive in the respective cases. The randomized version of this algorithm is, respectively, e-, e-, and $(e+1)$-competitive.

We start by describing the algorithm for the weighted integral model. The algorithm uses two variables: *GUESS* holds the current estimate of H, and *LIMIT* deter-

mines the saturation threshold; a server is *saturated* if its load is at least *LIMIT*. We say that a set of servers U is *saturated* if every server $s \in U$ is saturated. The set U is *overloaded* if the total weight assigned to servers in U is greater than $|U| \cdot LIMIT$. A newly arrived job is *dangerous* if assigning it to its lowest unsaturated eligible server will overload some trunk. In particular, if its eligible trunk is saturated, the job is dangerous. The algorithm avoids overloading any trunk by incrementing *LIMIT* whenever a dangerous job arrives. This, in turn, guarantees that whenever a job arrives, at least one of its eligible servers is unsaturated. Note that assigning the job may saturate the server to which it is assigned.

Algorithm Doubling

Initialize (upon arrival of the first job):

1. Let w be the first job's weight and T its eligible trunk.
2. $GUESS \leftarrow w/|T|$; $LIMIT \leftarrow GUESS$.

For each job:

3. While the job is dangerous:
    4. $GUESS \leftarrow 2 \cdot GUESS$.
    5. $LIMIT \leftarrow LIMIT + GUESS$.
6. Assign the job to its lowest unsaturated eligible server.

We divide the algorithm's execution into phases. A new phase begins whenever lines 4–5 are executed. (The arrival of a heavy job may trigger a succession of several empty phases.) Let p be the number of phases and denote by $GUESS_i$ and $LIMIT_i$ the respective values of *GUESS* and *LIMIT* during the ith phase. For consistency define $LIMIT_0 = 0$. Note that the initial value of *GUESS* ensures that $GUESS_1 \le H$.

PROPOSITION 18. *If $GUESS_i \ge H$ for some i, then the ith phase is the last one. Consequently, $GUESS_p < 2H$.*

Proof. Suppose $GUESS_i \ge H$ and consider the beginning of the ith phase. We claim that from this moment onward, the algorithm will not encounter any dangerous jobs. Suppose this is not true. Let us stop the algorithm when the first such dangerous job is encountered and assign the job manually to its lowest unsaturated eligible server. This overloads some trunk R. Let T be the maximal trunk containing R such that $T - R$ is saturated. Clearly, T is overloaded as well; the total weight assigned to it is greater than $|T| \cdot LIMIT_i$. On the other hand, T was not overloaded at the end of the $(i-1)$st phase, since the algorithm never overloads a trunk. Thus, the total weight of jobs assigned to T during the ith phase (including the job we have assigned manually) is greater than $|T|(LIMIT_i - LIMIT_{i-1}) = |T|GUESS_i \ge |T|H$. By T's maximality, all of these jobs must have requested servers in T. Thus, the total weight of jobs requesting servers in T is greater than $|T|H$, yielding a contradiction. □

COROLLARY 19. *$COST < 4OPT + w_{\max}$.*

Proof. The claim follows since $COST < LIMIT_p + w_{\max}$ and

$$LIMIT_p = GUESS_p \sum_{i=1}^{p} \frac{1}{2^{p-i}} < GUESS_p \sum_{i=0}^{\infty} \frac{1}{2^i} < 4H \le 4OPT. \qquad \square$$

Thus, Algorithm Doubling is 4-competitive in the unweighted integral model and 5-competitive in the weighted integral model. In the fractional model we modify the algorithm as follows. A job is called dangerous iff its eligible path is saturated. When assigning the job we may have to split it, as in the proof of the optimum lemma for

the linear hierarchy (Lemma 1). This algorithm achieves $COST \leq LIMIT_p < 4H = 4OPT$.

5.2. Randomized doubling. We consider randomization against oblivious adversaries. The randomization technique we use is fairly standard by now; a similar idea has been used several times in different contexts (see, e.g., [11, 19, 15, 12, 10]). The idea is to randomize the initial value of *GUESS* and to tweak the doubling parameter. Specifically, let r be a random variable uniformly distributed over $(0, 1]$ and select any constant $k > 1$. We replace lines 2 and 4 with

$2'$. $GUESS \leftarrow k^{r-1}w/|T|$; $LIMIT \leftarrow GUESS$;

$4'$. $GUESS \leftarrow k \cdot GUESS$.

It can be shown that $E(LIMIT_p) \leq H \cdot k/\ln k$ (see [12] or [10] for details). This expression is minimized at $e \cdot H$ by putting $k = e$. Thus, for $k = e$, the algorithm is e-competitive in the fractional and unweighted integral models and $(e+1)$-competitive in the weighted integral model.

5.3. Lower bounds for temporary jobs. In contrast to the linear hierarchy, allowing temporary jobs in the tree hierarchy has a drastic effect on the competitiveness of the solutions. For the unweighted integral model, we show deterministic and randomized lower bounds of $\sqrt{n}$ and $\frac{1}{2}(1 + \sqrt{n+1})$, respectively. These bounds are tight, up to a multiplicative constant, as demonstrated by the upper bound shown in [6] for the general problem with unrestricted eligibility constraints. Our randomized lower bound construction applies in the fractional model as well.

5.3.1. A deterministic lower bound for the integral models. Let $\mathcal{A}$ be a deterministic on-line algorithm, and let $n = k^2$ for some integer $k > 1$. We show an input sequence for which $\mathcal{A}$'s assignment satisfies $COST \geq k \cdot OPT = \sqrt{n}OPT$.

The server tree we use has a flower-like structure. It is composed of a *stem* and *petals*. The stem consists of k servers $s_1, s_2, \ldots, s_k$; s_1 is the root, and s_i is the parent of s_{i+1} for $1 \leq i < k-1$. The petals, $p_1, p_2, \ldots, p_{k^2-k}$, are all children of s_k. Server s_k is called the *calyx*.

Suppose that the competitiveness of $\mathcal{A}$ is better than k for the given n. Consider the following request sequence. Let c be an arbitrarily large integer.

1. For $i = 1, 2, \ldots, k^2 - k$:
    2. $c(k+1)$ jobs, each of unit weight, request the petal p_i.
    3. Of these jobs, ck now depart. (The rest are permanent.) The choice of which jobs depart is made so as to maximize the number of jobs assigned by $\mathcal{A}$ to p_i which depart.
4. ck jobs request the calyx.

During the first stage (lines 1–3), the adversary always assigns the permanent jobs to the petal and the temporary ones to the servers in the stem, c jobs to each server. Thus, at the beginning of the second stage, no jobs are assigned to the stem servers, and the adversary assigns c new jobs to each of them. Thus, $OPT = c$.

Consider the jobs requesting p_i. Since $\mathcal{A}$ is better than k-competitive, it assigns fewer than ck jobs to p_i. Thus, in each iteration more than c permanent jobs are assigned by $\mathcal{A}$ to servers in the stem. Hence, at the beginning of the second stage there are more than $c(k^2-k)$ jobs assigned to servers in the stem. Since the additional ck jobs must be assigned to servers in the stem, at least one server must end up with a load greater than ck, contradicting the assumption that $\mathcal{A}$ is better than k-competitive.

5.3.2. A randomized lower bound. Let us generalize the previous construction. We use a flower-like server tree with p petals and a stem of s servers ($n = s+p$). As before, the input sequence has two stages. The first consists of p iterations, where in the ith iteration, $c \cdot (s+1)$ jobs request the petal p_i and then $c \cdot s$ of them depart. In the second stage cs jobs request the calyx. The goal in the first stage is to "push" enough jobs into the stem so that the algorithm will fail in the second stage.

Consider the ith iteration. Let X_j be the random variable denoting the contribution of the jth job to the load on servers in the stem. (In the integral model this is a 0–1 variable; in the fractional model X_j may assume any value between 0 and 1.) Since the algorithm is better than k-competitive, the expected total weight assigned to p_i is less than ck, so $E(\sum_j X_j) > c(s+1-k)$. Thus, there exist c jobs $j_1, j_2, \ldots, j_c$ such that $\sum_{i=1}^{c} E(X_{j_i}) > c(s+1-k)/(s+1)$. The adversary makes these jobs permanent and terminates the rest.

Consequently, at the beginning of the second stage the expected total load on the servers in the stem is greater than $cp(s+1-k)/(s+1)$, and at the end of the second stage the expectation grows to more than $cs + cp(s+1-k)/(s+1)$. Since the algorithm is better than k-competitive, the expected maximum load on a server in the stem must be less than ck, and thus the expected total load on servers in the stem must be less than cks. To reach a contradiction we choose s and p to satisfy

$$cks = cs + \frac{cp(s+1-k)}{s+1}.$$

Solving the equation yields

$$p = \frac{s(s+1)(k-1)}{s+1-k}, \qquad n = s+p = \frac{ks^2}{s+1-k}.$$

Minimizing n (for fixed k) subject to the last equation yields $s = 2(k-1)$, $p = 2(2k-1)(k-1)$, and $n = 4k(k-1)$. Thus, $k = \frac{1}{2}(1+\sqrt{n+1})$.

6. Other eligibility restrictions. In the *hierarchical servers* problem the sets of eligible servers for a job have a restricted form. For example, in the linear hierarchy they have the following form: all servers to the left of some server s. This is a special case of the problem considered in [7], where eligible sets may be arbitrary. In this section we study various other restrictions on the eligible sets. We focus on the following three.

1. The servers form a path and the set of servers eligible for a given job must be contiguous on the path.

2. The servers form a rooted tree and each job specifies a node v, all of whose descendents (including v itself) are eligible.

3. The number of servers eligible for any given job is at most k for some fixed $2 \le k \le n$.

We show how to extend the $\Omega(\log n)$ lower bound of [7] to these three variants of the problem. This shows that the greedy algorithm, which is $O(\log n)$-competitive for the general problem, remains optimal (up to a multiplicative constant) in many restrictive scenarios.

For the first variant, consider the following input sequence. For convenience assume n is a power of 2 (otherwise consider only the first $2^{\lfloor \log n \rfloor}$ servers on the path). All jobs have unit weight, and they arrive in $\log n$ *rounds*. The ith round consists of $m_i = n/2^i$ jobs, all of which specify the same set of eligible servers S_i. The sets S_i are chosen such that $|S_i| = 2n/2^i$. In the first round all servers are eligible.

Having defined S_i, we construct S_{i+1} as follows. Suppose the total weight assigned to servers in S_i at the end of the ith round is at least $ni/2^i$ (as is certainly the case for $i = 1$). The set S_i is contiguous, i.e., its servers form a path. At least half of the total weight assigned to S_i is assigned to either the first half of this path or to its second half. We define S_{i+1} as the half to which the majority of the weight is assigned (breaking ties arbitrarily). Thus, the total weight assigned to S_{i+1} at the end of the $(i+1)$st round is at least $n/2^{i+1} + \frac{1}{2}ni/2^i = n(i+1)/2^{i+1}$. We define $S_{\log n+1}$ in the same manner and call the single server which it comprises the *leader*. The load on the leader at the end of the last round is at least $\frac{1}{2}\log n$. The adversary assigns the jobs in the ith round to the servers $S_i - S_{i+1}$, one job to each server. Thus, $OPT \leq 1$, and the lower bound follows.

For the second variant we use a very similar construction. The servers are arranged in a complete binary tree. The number of jobs in the ith round is defined by the recurrence $m_i = \frac{1}{2}(m_{i-1} - 1)$ with $m_1 = \frac{1}{2}(n-1)$. The sets of eligible servers are defined as follows. In the first round all servers are eligible. Let v_i be the root of the subtree S_i; we define S_{i+1} as the subtree rooted at the child of v_i to which more weight is assigned at the end of the ith round.

For the third variant we use a recursive construction. Partition the servers into n/k subsets of k servers each and apply the construction of the first variant to each subset. The load on the leader of each subset is now $\Omega(\log k)$. Continue recursively on the set of leaders. Each level of the recursion increases the load on the leaders in that level by $\Omega(\log k)$, and there are $\Theta(\log_k n) = \Theta(\log n/\log k)$ levels. Thus, $COST = \Omega(\log n)$. At each level of the recursion, the adversary assigns no weight to the leaders and at most one job to the other servers. Hence, $OPT \leq 1$, and the lower bound follows.

REFERENCES

[1] M. Abramowitz and I. A. Stegun eds., *Handbook of Mathematical Functions*, Dover, New York, 1965.

[2] S. Albers, *Better bounds for online scheduling*, SIAM J. Comput., 29 (1999), pp. 459–473.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waartz, *On-line machine scheduling with applications to load balancing and virtual circuit routing*, J. ACM, 44 (1997), pp. 486–504.

[4] Y. Azar, *On-line load balancing*, in On-line Algorithms: The State of the Art, A. Fiat and G. Woeginger, eds., Lecture Notes in Comput. Sci. 1442, Springer-Verlag, Berlin, 1998, pp. 178–195.

[5] Y. Azar, A. Broder, and A. Karlin, *On-line load balancing*, Theoret. Comput. Sci., 130 (1994), pp. 73–84.

[6] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts, *On-line load balancing of temporary tasks*, J. Algorithms, 22 (1997), pp. 93–110.

[7] Y. Azar, J. Naor, and R. Rom, *The competitiveness of on-line assignments*, J. Algorithms, 18 (1995), pp. 221–237.

[8] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, *New algorithms for an ancient scheduling problem*, in Proceedings of the 24th ACM Symposium on Theory of Computing, 1992, pp. 51–58.

[9] Y. Bartal, H. J. Karloff, and Y. Rabani, *A better lower bound for on-line scheduling*, Inform. Process. Lett., 50 (1994), pp. 113–116.

[10] A. Bar-Noy, A. Freund, and J. Naor, *New algorithms for related machines with temporary jobs*, J. Sched., 3 (2000), pp. 259–272.

[11] A. Beck and D. Newman, *Yet more on the linear search problem*, Israel J. Math., 8 (1970), pp. 419–429.

[12] P. Berman, M. Charikar, and M. Karpinski, *On-line load balancing for related machines*, J. Algorithms, 35 (2000), pp. 108–121.

[13] A. Borodin and R. El-Yaniv, *On-line Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.

[14] L. Epstein and J. Sgall, *A lower bound for on-line scheduling on uniformly related machines*, Oper. Res. Lett., 26 (2000), pp. 17–22.

[15] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 152–157.

[16] R. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[17] D. R. Karger, S. J. Phillips, and E. Torng, *A better algorithm for an ancient scheduling problem*, J. Algorithms, 20 (1996), pp. 400–430.

[18] A. J. Kleywegt, V. S. Nori, M. W. P. Savelsbergh, and C. A. Tovey, *Online resource minimization*, in Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 576–585.

[19] R. Motwani, S. Phillips, and E. Torng, *Non-clairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.

[20] J. Sgall, *On-line scheduling*, in On-line Algorithms: The State of the Art, A. Fiat and G. Woeginger, eds., Lecture Notes in Comput. Sci. 1442, Springer-Verlag, Berlin, 1998, pp. 196–231.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 550–576

CHECKING APPROXIMATE COMPUTATIONS OF POLYNOMIALS AND FUNCTIONAL EQUATIONS*

FUNDA ERGÜN†, S. RAVI KUMAR‡, AND RONITT RUBINFELD§

Abstract. A majority of the results on self-testing and correcting deal with programs which purport to compute the correct results precisely. We relax this notion of correctness and show how to check programs that compute only a numerical approximation to the correct answer. The types of programs that we deal with are those computing polynomials and functions defined by certain types of functional equations. We present results showing how to perform approximate checking, self-testing, and self-correcting of polynomials, settling in the affirmative a question raised by [P. Gemmell et al., *Proceedings of the* 23*rd ACM Symposium on Theory of Computing,* 1991, pp. 32–42; R. Rubinfeld and M. Sudan, *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms,* Orlando, FL, 1992, pp. 23–43; R. Rubinfeld and M. Sudan, *SIAM J. Comput.,* 25 (1996), pp. 252–271]. We obtain this by first building approximate self-testers for linear and multilinear functions. We then show how to perform approximate checking, self-testing, and self-correcting for those functions that satisfy addition theorems, settling a question raised by [R. Rubinfeld, *SIAM J. Comput.,* 28 (1999), pp. 1972–1997]. In both cases, we show that the properties used to test programs for these functions are both robust (in the approximate sense) and stable. Finally, we explore the use of reductions between functional equations in the context of approximate self-testing. Our results have implications for the stability theory of functional equations.

Key words. program testing, approximate testing, property testing, polynomials, functional equations

AMS subject classifications. 68Q25, 68Q60, 68W20, 65Y99, 13P99

PII. S0097539798337613

1. Introduction.

Program checking was introduced by Blum and Kannan [7] in order to allow one to use a program safely without having to know a priori that the program is correct on all inputs. Related notions of self-testing and self-correcting were further explored in [8, 24]. These notions are seen to be powerful from a practical point of view (cf. [9]) and from a theoretical angle (cf. [5, 4]) as well. The techniques used usually consist of tests performed at run-time which compare the output of the program either to a predetermined value or to a function of outputs of the same program at different inputs. In order to apply these powerful techniques to programs computing real-valued functions, several issues dealing with *precision* need to be dealt with. The standard model, which considers an output to be wrong even if it is off by a very small margin, is too strong to make practical sense due to reasons such as the following: (i) In many cases, the algorithm is only intended to compute an approximation, e.g., Newton's method. (ii) Representational limitations

*Received by the editors April 20, 1998; accepted for publication (in revised form) December 20, 1999; published electronically September 26, 2001. This work was partially supported by NSF Career grant CCR-9624552, the Alfred P. Sloan Research Award, and ONR grant N00014-97-1-0505. The first and second authors were also supported by NSF grant DMI-91157199. The third author was also supported by grant 92-00226 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel. Parts of this research were conducted while the authors were at the Computer Science Department of Cornell University and visiting the MIT Lab for Computer Science. A preliminary version of this paper appeared in *Proceedings of the* 37*th IEEE Conference on Foundations of Computer Science,* 1996.

http://www.siam.org/journals/sicomp/31-2/33761.html

†Case Western Reserve University, Cleveland, OH 44106 (afe@eecs.cwru.edu).

‡IBM Almaden Research Center, San Jose, CA 95120 (ravi@almaden.ibm.com).

§NEC Research Institute, Princeton, NJ 08540 (ronitt@research.nj.nec.com).

and round-off/truncation errors are inevitable in real-valued computations. (iii) The representation of some fundamental constants (e.g., $\pi = 3.14159\ldots$) is inherently imprecise.

The framework presented by [20, 3] accommodates these inherently inevitable or acceptably small losses of information by overlooking small precision errors while detecting actual "bugs," which manifest themselves with greater magnitude. Given a function f, a program P that purports to compute f, and an error bound Δ, if $|P(x) - f(x)| \leq \Delta$ (denoted $P(x) \approx_\Delta f(x)$) under some appropriate notion of norm, we say $P(x)$ is *approximately correct on input* x. *Approximate result checkers* test if P is approximately correct for a given input x. *Approximate self-testers* are programs that test if P is approximately correct for most inputs. *Approximate self-correctors* take programs that are approximately correct on most inputs and turn them into programs that are approximately correct on every input.

Domains. We work with finite subsets of fixed point arithmetic that we refer to as *finite rational domains*. For $n, s \in \mathbb{Z}^+$, $\mathcal{D}_{n,s} \stackrel{\text{def}}{=} \{\frac{i}{s} : |i| \leq n, i \in \mathbb{Z}\}$. Usually, $s = 2^l$ where l is the precision. We allow s and n to vary for generality. For a domain $\mathcal{D}$, let $\mathcal{D}^+$ and $\mathcal{D}^-$ denote the positive and negative elements in $\mathcal{D}$.

Testing using properties. There are many approaches to building self-testers. We illustrate one paradigm that has been particularly useful. In this approach, in order to test if a program P computes a function f on most inputs, we test if P satisfies certain properties of f.

As an example, consider the function $f(x) = 2x$ and the property "$f(x+1) = f(x)+2$" that f satisfies. One might pick random inputs x and verify that $P(x+1) = P(x)+2$. Clearly, if for some x, $P(x+1) \neq P(x)+2$, then P is incorrect. The program, however, might be quite incorrect and still satisfy $P(x+1) = P(x)+2$ for most choices of random inputs. In particular, there exists a P (for instance, $P(x) = 2x \bmod K$)[1] such that (i) with high probability, P satisfies the property at random x and hence will pass the test, and (ii) there is no function that satisfies the property for all x such that P agrees with this function on most inputs. Thus we see that this method, when used naively, does not yield a self-tester that works according to our specifications. Nevertheless, this approach has been used as a good heuristic to check the correctness of programs [13, 14, 35].

As an example of a property that does yield a good tester, consider the linearity property "$f(x+y) = f(x) + f(y)$," satisfied only by functions mapping $\mathcal{D}_{n,s}$ to $\mathbb{R}$ of the form $f(x) = cx, c \in \mathbb{R}$. If, by random sampling, we conclude that the program P satisfies this property for most x, y, it can be shown that P agrees with a linear function g on most inputs [8, 28]. We call the linearity property, and any property that exhibits such behavior, a *robust property*.

We now describe more formally how to build a self-tester for a *class* $\mathcal{F}$ of functions that can be characterized by a robust property. The two-step approach, which was introduced in [8], is as follows: (i) test that P satisfies the robust property (*property testing*) and (ii) check if P agrees with a *specific* member of $\mathcal{F}$ (*equality testing*). The success of this approach depends on finding robust properties which are both easy to test and lead to efficient equality tests.

A *property* is a pair $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$, consisting of an equation $\mathcal{I}^f(x_1, \ldots, x_k) = 0$ that relates the values of function f at various tuples of locations $\langle x_1, \ldots, x_k \rangle$, and

[1] We naturally extend the mod function to $\mathcal{D}_{n,s}$ by letting $x \bmod K$ stand for $\frac{j \bmod k}{s}$, for $x, K \in \mathcal{D}_{n,s}$, and $x = \frac{j}{s}$, $K = \frac{k}{s}$.

a distribution $\mathcal{E}_{\tau(n,s)}$ over $\mathcal{D}^k_{\tau(n,s)}$ from which the locations are picked. The property $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$ is said to characterize a function family $\mathcal{F}$ in the following way. A function f is a member of $\mathcal{F}$ if and only if $\mathcal{I}^f(x_1, \ldots, x_k) = 0$ for every $\langle x_1, \ldots, x_k \rangle$ that has nonzero support under $\mathcal{E}_{\tau(n,s)}$. For instance, the linearity property can be written as $\mathcal{I}^f(x_1, x_2, x_3) \equiv f(x_1) + f(x_2) - f(x_3) = 0$, and $\mathcal{E}^{\mathrm{Lin}}_{\tau(n,s)}$ is a distribution on $\langle x_1, x_2, x_1 + x_2 \rangle$, where x_1 and x_2 are chosen randomly from some distribution[2] over the domain $\mathcal{D}_{\tau(n,s)}$. In this case $\langle \mathcal{I}, \mathcal{E}^{\mathrm{Lin}}_{\tau(n,s)} \rangle$ characterizes $\mathcal{F} = \{f(x) = cx \mid c \in \mathbb{R}\}$, the set of all linear functions over $\mathcal{D}_{\tau(n,s)}$. We will adhere to this definition of a property throughout the paper; however, for simplicity of notation, when appropriate, we will talk about the distribution and the equality together. For instance, we express the linearity property as $f(x + y) = f(x) + f(y)$, giving the distributions of x, y.

We first consider robust properties in more detail. Suppose we want to infer the correctness of the program on inputs from the domain $\mathcal{D}_{n,s}$. Then we allow calls to the program on a larger domain $\mathcal{D}_{\tau(n,s)}$, where $\tau : \mathbb{Z}^2 \to \mathbb{Z}^2$ is a fixed function that depends on the structure of $\mathcal{I}$. Ideally, we would like $\tau(n, s) = (n, s)$, i.e., $\mathcal{D}_{\tau(n,s)} = \mathcal{D}_{n,s}$. However, for technical reasons, we allow $\mathcal{D}_{\tau(n,s)}$ to be a proper, but not too much larger, superset of $\mathcal{D}_{n,s}$ (in particular, the description size of an element in $\mathcal{D}_{\tau(n,s)}$ should be polynomial in the description size of an element in $\mathcal{D}_{n,s}$).[3]

To use a property in a self-tester, one must prove that the property is robust. Informally, the $(\delta, \epsilon, \mathcal{D}_{\tau(n,s)}, \mathcal{D}_{n,s})$-*robustness* of the property $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$ implies that if, for a program P, $\mathcal{I}^P(x_1, \ldots, x_k) = 0$ is satisfied with probability at least $1 - \epsilon$ when $\langle x_1, \ldots, x_k \rangle$ is chosen from the distribution $\mathcal{E}_{\tau(n,s)}$, then there is a function $g \in \mathcal{F}$ that agrees with P on $1 - \delta$ fraction of the inputs in $\mathcal{D}_{n,s}$. In the case of linearity, it can be shown that there is a distribution $\mathcal{E}^{\mathrm{Lin}}_{11n,s}$ on $\langle x_1, x_2, x_1 + x_2 \rangle$ where $x_1, x_2 \in \mathcal{D}_{11n,s}$ such that the property is $(2\epsilon, \epsilon, \mathcal{D}_{11n,s}, \mathcal{D}_{n,s})$-robust for all $\epsilon < 1/48$ [8, 28]. Therefore, once it is tested that P satisfies $P(x_1) + P(x_2) = P(x_1 + x_2)$ with large enough probability when the inputs are picked randomly from $\mathcal{E}^{\mathrm{Lin}}_{11n,s}$, it is possible to conclude that P agrees with some linear function on most inputs from $\mathcal{D}_{n,s}$. A somewhat involved definition of *robust* is given in [28]. Given a function τ such that for all n, s, $\mathcal{D}_{n,s}$ is a large enough subset of $\mathcal{D}_{\tau(n,s)}$, in this paper we say that a property is *robust* if for all $0 < \delta < 1$, there is an ϵ such that for all n, s the property is $(\delta, \epsilon, \mathcal{D}_{\tau(n,s)}, \mathcal{D}_{n,s})$-robust.

We now consider equality testing. Recall that once it is determined that P satisfies the robust property, then equality testing determines that P agrees on most inputs with a *specific* member of $\mathcal{F}$. For instance, in the case of linearity, to ensure that P computes the specific linear function $f(x) = x$ on most inputs, we perform the equality test which ensures that $P(x + \frac{1}{s}) = P(x) + \frac{1}{s}$ for most x. Neither the property test nor the equality test on its own is sufficient for testing the program. However, since $f(x) = x$ is the *only* function that satisfies both the linearity property and the above equality property, the combination of the property test and the equality test can be shown to be sufficient for constructing self-testers.

This combined approach yields very efficient testers (that make only $O(\epsilon^{-1} \log 1/\delta)$ calls to the program for fixed δ and ϵ) for programs computing homomorphisms (e.g., multiplication of integers and matrices, exponentiation, logarithm). This idea is further generalized in [28], where the class of functional equations called *addition theorems* is shown to be useful for self-testing. An addition theorem is a mathematical

[2] For example, choosing x_1 and x_2 uniformly from $\mathcal{D}_{\tau(n,s)}$ suffices for characterizing linearity. To prove robustness, however, [28] uses a more complicated distribution that we do not describe here.

[3] Alternatively, one could test the program over the domain $\mathcal{D}_{n,s}$ and attempt to infer the correctness of the program on most inputs from $\mathcal{D}_{n',s'}$, where $\mathcal{D}_{n',s'}$ is a large subdomain of $\mathcal{D}_{n,s}$.

TABLE 1
Some addition theorems of the form $f(x+y) = G[f(x), f(y)]$.

$G[f(x), f(y)]$	$f(x)$
$f(x) + f(y)$	Ax
$\frac{f(x)+f(y)}{1-f(x)f(y)}$	$\tan Ax$
$\frac{f(x)f(y)-1}{f(x)+f(y)}$	$\cot Ax$
$\frac{f(x)+f(y)-1}{2f(x)+2f(y)-2f(x)f(y)-1}$	$\frac{1}{1+\tan Ax}$
$\frac{f(x)+f(y)-2f(x)f(y)}{1-f(x)f(y)}$	$\frac{-Ax}{1-Ax}$
$\frac{f(x)+f(y)}{1+[f(x)f(y)]/A^2}$	$A\tanh Bx$
$\frac{f(x)f(y)}{f(x)+f(y)}$	$\frac{A}{x}$

$G[f(x), f(y)]$	$f(x)$
$f(x)f(y) - \sqrt{1-f(x)^2}\sqrt{1-f(y)^2}$	$\cos Ax$
$\frac{f(x)+f(y)-2f(x)f(y)}{1-2f(x)f(y)}$	$\frac{1}{1+\cot Ax}$
$\frac{f(x)+f(y)-2f(x)f(y)\cos a}{1-f(x)f(y)}$	$\frac{\sin Ax}{\sin Ax+a}$
$\frac{f(x)+f(y)-2f(x)f(y)\cosh a}{1-f(x)f(y)}$	$\frac{\sinh Ax}{\sinh Ax+a}$
$\frac{f(x)+f(y)+2f(x)f(y)\cosh a}{1-f(x)f(y)}$	$\frac{-\sinh Ax}{\sinh Ax+a}$
$\frac{f(x)+f(y)+2f(x)f(y)}{1-f(x)f(y)}$	$\frac{Ax}{1-Ax}$
$f(x)f(y) + \sqrt{f(x)^2-1}\sqrt{f(y)^2-1}$	$\cosh Ax$

identity of the form for all x, y, $f(x+y) = G[f(x), f(y)]$. Addition theorems characterize many useful and interesting mathematical functions [1, 11]. When G is algebraic, they can be used to characterize families of functions that are rational functions of x, e^{cx}, and doubly periodic functions (see Table 1 for examples of functional equations and the families of functions that they characterize over the reals). Polynomials of degree d can be characterized via several different robust functional equations (e.g., [6, 26, 4, 30]).

Approximate robustness and stability. When the program works with finite precision, the properties upon which the testers are built will rarely be satisfied, even by a program whose answers are correct up to the required (or hardware-wise maximal) number of digits, since they involve strict equalities. Thus, when testing, one might be willing to pass programs for which the properties are only approximately satisfied. This relaxation in the tests, however, leads to some difficulties, for in the approximate setting (i) it is harder to analyze which function families are solutions to the robust properties, and (ii) equality testing is more difficult. For instance, it is not obvious which family of functions would satisfy both $P(x_1) + P(x_2) \approx P(x_1 + x_2)$, for all $x, y \in \mathcal{D}_{\tau(n,s)}$ (approximate linearity property), and $P(x + \frac{1}{s}) \approx P(x) + \frac{1}{s}$ for all $x \in \mathcal{D}_{\tau(n,s)}$ (approximate equality property).

To construct approximate self-testers, our approach is to first investigate a notion of *approximate robustness* of the property to be used. We first require a notion of distance between two functions.

DEFINITION 1 (Chebyshev norm). *For a function f on a domain $\mathcal{D}$, $\|f\|_{\mathcal{D}} = \|f\| = \sup_{x\in\mathcal{D}}\{|f(x)|\}$.*

When the domain is obvious from the context, we drop it. Given functions f, g, the *distance* between them is $\|f-g\|$. Next, we define the *approximation* of a function by another function.

DEFINITION 2. *The function P (Δ, ϵ)-approximates f on domain $\mathcal{D}$ if $\|P-f\| \leq \Delta$ on at least $1-\epsilon$ fraction of $\mathcal{D}$.*

Approximate robustness is a natural extension of the robustness of a property. We say that a *program satisfies a property approximately* if the property is true of the program when exact equalities are replaced by approximate equalities. Once again consider the linearity property and a program P that satisfies the property approximately (i.e., $P(x_1 + x_2) \approx_\Delta P(x_1) + P(x_2)$) for all but an ϵ fraction of the choices of $\langle x_1, x_2, x_1 + x_2\rangle \in \mathcal{E}^{\text{Lin}}_{\tau(n,s)}$. The approximate robustness of linearity implies that there exists a function g and a choice of Δ', Δ'' such that $g(x+y) \approx_{\Delta'} g(x) +$

$g(y)$ for all inputs $x, y \in \mathcal{D}_{n,s}$, and g $(\Delta'', 2\epsilon)$-approximates P on $\mathcal{D}_{n,s}$ [20, 28]. In general, we would like to define approximate robustness of a property $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$ as the following: If a program P satisfies the equation $\mathcal{I}$ approximately on most choices of inputs according to the distribution $\mathcal{E}_{\tau(n,s)}$, then there exists a function g that (i) satisfies $\mathcal{I}$ approximately on all inputs chosen according to $\mathcal{E}_{n,s}$ and (ii) approximates P on most inputs in $\mathcal{D}_{n,s}$, the support of $\mathcal{E}_{\tau(n,s)}$. The function τ relates the distributions used for describing the behaviors of P and G and depends on $\mathcal{I}$.

We now give a formal definition of approximate robustness.

DEFINITION 3 (approximate robustness). *Let $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$ characterize the family of functions $\mathcal{F}$ over the domain $\mathcal{D}_{\tau(n,s)}$. Let $\mathcal{F}'$ be the family of functions satisfying $\mathcal{I}$ approximately on all inputs chosen according to $\mathcal{E}_{n,s}$. Let ϵ, δ be constants independent of n. A property $\langle \mathcal{I}, \mathcal{E}_{\tau(n,s)} \rangle$ for a function family $\mathcal{F}'$ is* $(\delta, \epsilon, \mathcal{D}_{\tau(n,s)}, \mathcal{D}_{n,s}, \Delta, \Delta', \Delta'')$-approximately robust *if for all $P, \Pr_{\langle x_1,\ldots,x_k \rangle \sim \mathcal{E}_{\tau(n,s)}}[\mathcal{I}^P(x_1, \ldots, x_k) \approx_\Delta 0] \geq 1 - \epsilon$ implies there is a $g \in \mathcal{F}'$ that (Δ'', δ)-approximates P on $\mathcal{D}_{n,s}$ and $\mathcal{I}^g(x_1, \ldots, x_k) \approx_{\Delta'} 0$ for all tuples $\langle x_1, \ldots, x_k \rangle$ with nonzero support in $\mathcal{E}_{n,s}$.*

Once we know that the property is approximately robust, the second step is to analyze the *stability* of the property, i.e., to characterize the set of functions $\mathcal{F}'$ that satisfy the property approximately and compare it to $\mathcal{F}$, the set of functions that satisfy the property exactly (*Hyers–Ulam stability* [21]). In our linearity example, the problem is the following: given g satisfying $g(x+y) \approx_\Delta g(x) + g(y)$ for all x, y in the domain, is there a homomorphism h that $(\Delta', 0)$-approximates g with Δ' depending only on Δ and *not* on the size of the domain? If the answer is affirmative, we say that the property is *stable*. In the following definition, $\mathcal{D}_{n',s'} \subseteq \mathcal{D}_{n,s}$.

DEFINITION 4 (stability). *A property $\langle \mathcal{I}, \mathcal{E}_{n,s} \rangle$ for a function family $\mathcal{F}$ is* $(\mathcal{D}_{n,s}, \mathcal{D}_{n',s'}, \Delta, \Delta')$-stable *if for all g that satisfies $\mathcal{I}^g \approx_\Delta 0$ for all tuples with nonzero support according to $\mathcal{E}_{n,s}$, there is a function h that satisfies $\mathcal{I}^h = 0$ for all tuples with nonzero support according to $\mathcal{E}_{n',s'}$ with $\|h - g\|_{\mathcal{D}_{n',s'}} \leq \Delta'$.*

If a property is both approximately robust and stable, then it can be used to determine whether P approximates some function in the desired family. Furthermore, if we have a method of doing approximate equality testing, then we can construct an approximate self-tester. Here, we assume that the distributions associated with approximate robustness and stability are samplable.

Previous work. Previously, not many of the known checkers have been extended to the approximate case. Often it is rather straightforward to extend the robustness results to show approximate robustness. However, the difficulty with extending the checkers appears to lie in showing the stability of the properties. The issue is first mentioned in [20], where approximate checkers for mod, exponentiation, and logarithm are constructed. The domain is assumed to be closed in all of these results. A domain is said to be closed under an operation if the range of the operation is a subset of the domain. For instance, a finite precision rational domain is not closed under addition. In [3] approximate checkers for sine, cosine, matrix multiplication, matrix inversion, linear system solving, and determinant are given. The domain is assumed to be closed in the results on sine and cosine. In [10] an approximate checker for floating-point division is given. In [32], a technique which uses approximation theory is presented to test univariate polynomials of degree at most 9. It is left open in [20, 3, 30, 28] whether the properties used to test polynomial, hyperbolic, and other trigonometric functions can be used in the approximate setting. For instance, showing the stability of such functional equations is not obvious; if the functional equation involves division with a large numerator and a small denominator, a small additive error in the denominator

leads to a large additive error in the output.

There has been significant work on the stability of specific functional equations. The stability of linearity and other homomorphisms is addressed in [21, 16, 18, 12]. The techniques used to prove the above results, however, cease to apply when the domain is not closed. The stronger property of stability in a nonclosed space, called *local stability*, is addressed by Skof [31], who proves that Cauchy functional equations are locally stable on a finite interval in $\mathbb{R}$. The problem of stability of univariate polynomials over continuous domains is first addressed in [2] and the problem of local stability on $\mathbb{R}$ is solved in [19]. See [17] for a survey. These results do not extend in an obvious way to finite subsets of $\mathbb{R}$ and thus cannot be used to show the correctness of self-testers. For those that can be extended, the error bounds obtained by naive extensions are not optimal. Our different approach allows us to operate on $\mathcal{D}_{n,s}$ *and* obtain tight bounds.

Results. In this paper, we answer the questions of [20, 3, 30, 28] in the affirmative by giving the first approximate versions of most of their testers. We first present an approximate tester for linear and multilinear functions with tight bounds. These results apply to several functions, including multiplication, exponentiation, and logarithm, over nonclosed domains. We next present the first approximate testers for multivariate polynomials. Finally, we show how to approximately test functions satisfying addition theorems. Our results apply to many algebraic functions of trigonometric and hyperbolic functions (e.g., sinh, cosh). All of our results apply to nonclosed discrete domains.

Since a functional equation over $\mathbb{R}$ has more constraints than the same functional equation over $\mathcal{D}_{n,s}$, it may happen that the functional equation over $\mathbb{R}$ characterizes a family of functions that is a proper subset of the functions characterized by the same functional equation over $\mathcal{D}_{n,s}$. This does not limit the ability to construct self-testers for programs for these functions, due to the equality testing performed by self-testers.

To show our results, we prove new local stability results for discrete domains. Our techniques for showing the stability of multilinearity differ from those used previously in that (i) we do not require the domain to be discrete and (ii) we do not require the range to be a complete metric space. This allows us to apply our results to multivariate polynomial characterizations. In addition to new combinatorial arguments, we employ tools from approximation theory and stability theory. Our techniques appear to be more generally applicable and cleaner to work with than those previously used.

Self-correctors are built by taking advantage of the random self-reducibility of polynomials and functional equations [8, 24] in the exact case. As in [20], we employ a similar idea for the approximate case by making several guesses at the answer and returning their median as the output. We show that if each guess is within Δ of the correct answer with high probability, then the median yields a good answer with high probability. To build an approximate checker for all of these functions, we combine the approximate self-tester and approximate self-corrector as in [8].

Subsequent to our work, our results have been extended to the case of relative error in a recent paper of [22].

Organization. Section 2 addresses the stability of the properties used to test linear and multilinear functions. Using these results, section 3 considers approximate self-testing of polynomials. Section 4 addresses the stability and robustness of functional equations. Section 5 illustrates the actual construction of approximate self-testers and self-correctors.

2. Linearity and multilinearity. In this section, we consider the stability of the robust properties used to test linearity and multilinearity over the finite rational domain $\mathcal{D}_{n,s}$. The results in this section, in addition to being useful for the testing of linear and multilinear functions, are crucial to our results in section 3.

As in [20], approximate robustness is easy to show by appropriately modifying the proof of robustness [28]. This involves replacing each exact equality by an approximate equality and keeping track of the error accrued at each step of the proof. To show stability, we use two types of bootstrapping arguments: the first shows that an error bound on a small subset of the domain implies the same error bound on a larger subset of the domain; the second shows that an error bound on the whole domain implies a tighter error bound over the same domain. These results can be applied to give the first approximate self-testers for several functions over $\mathcal{D}_{n,s}$ including multiplication, exponentiation, and logarithm (section 2.2).

2.1. Approximate linearity. The following defines formally what it means for a function to be approximately linear.

DEFINITION 5 (approximate linearity). *A function g is* Δ-approximately linear *on $\mathcal{D}_{n,s}$ if for all $x, y \in \mathcal{D}_{n,s}, g(x+y) \approx_\Delta g(x) + g(y)$.*

Hyers [21] and Skof [31] obtain a linear approximation to an approximately linear function when the domain is $\mathbb{R}$. (See Appendix A for their approach.) Their methods are not extendible to discrete domains.

Suppose we define h such that $h(\frac{1}{s}) \stackrel{\text{def}}{=} g(\frac{1}{s})$ and h is linear. In the 0-approximately linear case (exact linearity), since $g(\frac{i}{s}) = g(\frac{i-1}{s}) + h(\frac{1}{s})$ and $h(\frac{i}{s}) = h(\frac{i-1}{s}) + h(\frac{1}{s})$, by induction on the elements in $\mathcal{D}_{n,s}$, we can show that $h(x) = g(x)$ for all x. This approach is typically used to prove the sufficiency of the equality test. However, in the Δ-approximately linear case for $\Delta \neq 0$, using the same inductive argument will only yield a linear function h such that $h(\frac{i}{s}) \approx_{i\cdot\Delta} g(\frac{i}{s})$. This is quite unattractive since the error bound depends on the domain size. The problem of obtaining a linear function h whose discrepancy from g is *independent* of the size of the domain is nontrivial.

In [20], a solution is given for when the domain is a finite group. Their technique requires that the domain be closed under addition and therefore does not work for $\mathcal{D}_{n,s}$. We give a brief overview of the scheme in [20] and point out where it breaks down for nonclosed domains. The existence of a linear h that is close to g is done in [20] by arguing that if $\mathcal{D}$ is sufficiently large, then an error of at least Δ at the maximum error point x^* would imply an even bigger error at $2x^*$, contradicting the maximality assumption about error at x^*. Here, the crucial assumption is that $x \in \mathcal{D}$ implies $2x \in \mathcal{D}$. This step fails for domains which are not closed under addition.

Instead, we employ a different constructive technique to obtain a linear h on $\mathcal{D}_{n,s}$ given a Δ-approximately linear g. Our technique yields a tight bound of 2Δ on the error $e \equiv h - g$ (instead of 4Δ in [31]) and does not require that the domain be closed under addition. It is important to achieve the best (lowest) constants possible on the error because these results are used in section 3.2 where the constants affect the error in an exponential way.

The following lemma shows how to construct a linear function h that is within $2\Delta + \rho$ of a Δ-approximately linear function g in $\mathcal{D}^+_{n,s}$.

LEMMA 6. *Let g be a Δ-approximately linear function on $\mathcal{D}^+_{n,s}$, and let h be linear on $\mathcal{D}_{n,s}$. Define $e(x) = h(x) - g(x)$. If $|e(\frac{n}{s})| = \rho$, then for all $x \in \mathcal{D}^+_{n,s}, |e(x)| \leq 2\Delta + \rho$.*

Proof. We prove by contradiction that for all $x \in \mathcal{D}^+_{n,s}, e(x) \leq 2\Delta + \rho$. A symmetric argument can be made to show that $e(x) \geq -(2\Delta + \rho)$.

Recall that $\frac{n}{s}$ is the greatest positive element of the domain, and note that e is a Δ-approximately linear function. Assume that there exists a point in $\mathcal{D}^+_{n,s}$ with error greater than $2\Delta + \rho$. Let p be the maximal such element. p has to lie between $\frac{n}{2s}$ and $\frac{n}{s}$, otherwise $2p \in \mathcal{D}^+_{n,s}$ would have error greater than $2\Delta + \rho$, contradicting the maximality of p. Let $q = \frac{n}{s} - p$. Then, $e(q) + e(p) \approx_\Delta e(\frac{n}{s})$; therefore $e(q) < -\Delta$. Also, for any $x \in (p, \frac{n}{s}] \subseteq \mathcal{D}^+_{n,s}$, by definition of p, $e(x) \leq 2\Delta + \rho$. Note that any such x can be written as $x = x' + p$, where $x' \in (0, q]$. To satisfy the approximate linearity property that $e(x') + e(p) \approx_\Delta e(x)$, x' must have error strictly less than $\Delta + \rho$.

We now know that the points in the interval $(0, q]$ have error strictly less than $2\Delta + \rho$ (in fact, less than $\Delta + \rho$) and that the point q itself has error strictly less than $-\Delta$. Putting these two facts and approximate linearity together, and since any $x \in (q, 2q]$ can be written as $q + y$ where $y \in (0, q]$, we can conclude that at any point in $(q, 2q]$, the error is at most $2\Delta + \rho$. Now we can repeat the same argument by taking y from $(0, 2q]$ rather than $(0, q]$ to bound the error in the interval $(0, 3q]$ by $2\Delta + \rho$. By continuing this argument, eventually the interval contains the point p, which means that p has error at most $2\Delta + \rho$. This contradicts our initial assumption that $e(p)$ was greater than $2\Delta + \rho$. ∎

In addition, since $e(0) \approx_\Delta e(0) + e(0)$, $|e(0)| \leq \Delta$. We now generalize the error bound on $\mathcal{D}^+_{n,s}$ to $\mathcal{D}_{n,s}$.

LEMMA 7. *If a function g is Δ-approximately linear on $\mathcal{D}_{n,s}$, with h and e defined as in Lemma 6, and if $|e(\frac{n}{s})| = \rho$, then for all $x \in \mathcal{D}_{n,s}, |e(x)| \leq 2\Delta + \rho$.*

Proof. Observe that if the error $e(x)$ is upper bounded by σ when $x \in [0, \frac{n}{s}]$, then $|e(x)| \leq (\sigma + \Delta)/2$ whenever $0 \leq x \leq \frac{n}{2s}$, since $e(2x) \leq \sigma$. Also, if $|e(x)| \leq \mu$, then $|e(-x)| \leq \mu + 2\Delta$ since $e(0) \leq \Delta$. By Lemma 6, $e(x) \leq 2\Delta + \rho$ for all $x \in \mathcal{D}^+_{n,s}$. We will bound the error in $\mathcal{D}^-_{n,s}$ first by $3\Delta + \rho$ and then by $2\Delta + \rho$. From the above observations, we have $e(x) \leq 4\Delta + \rho$ for $x \in \mathcal{D}^-_{n,s}$, $e(x) \leq (3\Delta + \rho)/2$ for $x \in [0, \frac{n}{2s}]$ and $e(x) \leq (5\Delta + \rho)/2$ for $x \in [-\frac{n}{2s}, 0]$.

Assume that $\exists x \in \mathcal{D}^-_{n,s}$ such that $e(x) = 3\Delta + \rho + \epsilon > 3\Delta + \rho$. Let p be such a point with minimal absolute value. Then $p < -\frac{n}{2s}$, otherwise the error at $2p$ would exceed $3\Delta + \rho$. Let t be the point with the highest error in $\mathcal{D}^+_{n,s}$ (the maximal such one if there is a tie). We consider the possible locations for t to bound $e(t)$: (i) if $t \leq \frac{n}{2s}$, then to ensure that $e(2t) \leq e(t)$, $e(t) \leq \Delta$; (ii) if $\frac{n}{2s} < t \leq |p|$, then $t + p \in [-\frac{n}{2s}, 0]$; therefore, to satisfy the bound above on $e(t + p)$, $e(t) \leq \Delta/2 - \epsilon \leq \Delta$; (iii) if $t > |p|$, then $t + p \in (0, \frac{n}{2s}]$; therefore to satisfy the bound above, $e(t) \leq -\Delta/2 - \epsilon \leq \Delta$.

Regardless of where t lies, $e(t) \leq \Delta \leq \Delta + \rho$; hence the error in $\mathcal{D}^+_{n,s}$ is bounded by $\Delta + \rho$. However, $e(\frac{n}{s} + p) \geq 3\Delta + 2\rho + \epsilon - \Delta > 2\Delta + \rho$. Since $\frac{n}{s} + p \in \mathcal{D}^+_{n,s}$, this contradicts the bound we established before. Therefore, there cannot be a point in $\mathcal{D}^-_{n,s}$ with error greater than $3\Delta + \rho$. A symmetric argument can be used to bound negative error.

Now we reduce the error bound to $2\Delta + \rho$. Assume that p is the minimal point in $\mathcal{D}^-_{n,s}$ with error at least $2\Delta + \rho$. The proof is similar to the previous stage, using the tighter bound $e(x) \leq 2\Delta + \rho/2$ for $x \in [-\frac{n}{2s}, 0]$. Cases (i) and (iii) stay the same; for case (ii) we have $e(t + p) \leq -\epsilon \leq \Delta$. Therefore, the error cannot exceed $\Delta + \rho$ in $\mathcal{D}^+_{n,s}$. However, $e(\frac{n}{s} + p) \geq 2\Delta + \epsilon + \rho - \Delta$, which is a contradiction. ∎

The following special case proves the stability result for linearity.

COROLLARY 8. *The linearity property is $(\mathcal{D}_{n,s}, \mathcal{D}_{n,s}, \Delta, 2\Delta)$-stable.*

Proof. Suppose function g is Δ-approximately linear on $\mathcal{D}_{n,s}$. Set $h(\frac{n}{s}) = g(\frac{n}{s})$ in Lemma 7. This uniquely defines a linear h with $\rho = 0$. ∎

The intuition that drives us to set $h(\frac{n}{s}) = g(\frac{n}{s})$ in the proof of Corollary 8 is as

follows. Consider the following function of n, s: $g(\frac{n}{s}) = (\frac{n}{s} + \frac{[(n-1)/3]}{s})\Delta$ ($[x]$ denotes integer part of x). It is easy to see that $g(x+y) \approx_\Delta g(x) + g(y)$. Note that setting $h(\frac{1}{s}) = g(\frac{1}{s})$ instead of $h(\frac{n}{s}) = g(\frac{n}{s})$ does not work in general. If we set $h(\frac{1}{s}) = g(\frac{1}{s})$, then we obtain $h(\frac{n}{s}) = \frac{n}{s}\Delta$. However, $\|g-h\|$ is a growing function of n and so there is no way to bound the error at all points.

The following example shows that the error bound obtained in Corollary 8 using our technique is tight: we have shown how to construct a linear function h so that $\|h-g\| \leq 2\Delta$. We now show that there is a function g that, given our method of constructing h, asymptotically approaches this bound from below. Define g as follows: $g(n) = 0$; $g(x) = (3x/n - 1)\Delta$ for $0 \leq x \leq n-1$; $g(-x) = -g(x)$ for $0 < x \leq n$. It is easy to see that g is Δ-approximately linear: If $x+y < n$, $g(x+y) - g(x) - g(y) = \Delta$. If $x + y = n$, then $g(x+y) = 0$ and so $g(x) + g(y) = \Delta$. Our construction sets $h(n) = 0$; thus, $h \equiv 0$, the zero function. However, $\|g-h\| = |g(n-1) - h(n-1)| = (2 - 3/n)\Delta \longrightarrow 2\Delta$ for large enough n.

2.2. Approximate multilinearity. In this section we focus our attention on *multilinear functions*. A multivariate function is multilinear if it is linear in any one input when all the other inputs are fixed. A multilinear function of k variables is called a k-linear function. An example of a *bilinear* function is multiplication, and bilinearity property can be stated concisely as $f(x_1 + x_1', x_2 + x_2') = f(x_1, x_2) + f(x_1', x_2) + f(x_1, x_2') + f(x_1', x_2')$. Note that distributivity of multiplication over addition is a special case of multilinearity.

A natural extension of this class of functions is the class of approximately multilinear functions, which are formally defined below.

DEFINITION 9 (approximate multilinearity). *A k-variate function g is* Δ-approximately k-linear *on $\mathcal{D}_{n,s}^k$ if it is Δ-approximately linear on $\mathcal{D}_{n,s}$ in each variable.*

For instance, for $k = 2$, a function g is Δ-approximately bilinear if for all $x_1, x_1', x_2, x_2' \in \mathcal{D}$, $g(x_1 + x_1', x_2) \approx_\Delta g(x_1, x_2) + g(x_1', x_2)$ and $g(x_1, x_2 + x_2') \approx_\Delta g(x_1, x_2) + g(x_1, x_2')$.

Now we generalize Lemma 7 to Δ-approximately k-linear functions. Let g be a Δ-approximately k-linear function and h be the symmetric multilinear function uniquely defined by the condition $h(\frac{n}{s}, \ldots, \frac{n}{s}) = g(\frac{n}{s}, \ldots, \frac{n}{s})$. Let $e \equiv h - g$. e is a Δ-approximately k-linear function.

Since g takes k inputs from $\mathcal{D}_{n,s}$, if we consider each input to g as a coordinate, the set of all possible k-tuples of inputs of g form a $(2n+1) \times \cdots \times (2n+1)$ cube of dimension k. We show that for any point $(x_1, \ldots, x_k)$ in this cube, $|e(x_1, \ldots, x_k)|$ is bounded.

THEOREM 10. *The approximate k-linearity property is $(\mathcal{D}_{n,s}^k, \mathcal{D}_{n,s}^k, \Delta, 2k\Delta)$-stable. In other words, if a function g is Δ-approximately k-linear on $\mathcal{D}_{n,s}^k$, then there exists a k-linear h on $\mathcal{D}_{n,s}^k$ such that $\|h-g\| \leq 2k\Delta$.*

Proof. With h defined as above, $e(\frac{n}{s}, \ldots, \frac{n}{s}) = 0$. First, we argue about points that have one coordinate that is different from $\frac{n}{s}$. Fix $k-1$ of the inputs to be $\frac{n}{s}$ (hard-wire into g) and vary one (say, x_i). This operation transforms g from a Δ-approximately k-linear function of $x_1, \ldots, x_k$ to a Δ-approximately linear function of x_i. By Lemma 7, this function cannot have an error of more than 2Δ in $\mathcal{D}_{n,s}$. Therefore, $|e(\frac{n}{s}, \ldots, \frac{n}{s}, x_i, \frac{n}{s}, \ldots, \frac{n}{s})| \leq 2\Delta$, if $|x_i| < \frac{n}{s}$. Next we consider points which have two coordinates that are different from $\frac{n}{s}$. Consider without loss of generality an input $a, b, \frac{n}{s}, \ldots, \frac{n}{s}$. By the result we just argued, we know that $e(\frac{n}{s}, b, \frac{n}{s}, \ldots, \frac{n}{s}) \leq 2\Delta$. By fixing inputs 2 through k to be $b, \frac{n}{s}, \ldots, \frac{n}{s}$, and varying

the first input, by Lemma 7, we have $|e(a, b, \frac{n}{s}, \ldots, \frac{n}{s})| \leq 4\Delta$ for any $a \in \mathcal{D}_{n,s}$. Via symmetric arguments, we can bound the error by 4Δ if any two inputs are different from $\frac{n}{s}$. Continuing this way, it can be shown that for all inputs, the error is at most $2k\Delta$. □

The following theorem shows that the error can be reduced to $(1+\mu)\Delta$ for any constant $\mu > 0$ by imposing the multilinearity condition on a larger domain $\mathcal{D}'$ and fitting the multilinear function h on $\mathcal{D}$, where $|\mathcal{D}'|/|\mathcal{D}| = \lceil 2k/\mu \rceil$. Note that doubling the domain size only involves adding one more bit to the representation of a domain element.

THEOREM 11. *For any $\mu > 0$, the approximate multilinearity property is $(\mathcal{D}^k_{2kn/\mu,s}, \mathcal{D}^k_{n,s}, \Delta, (1+\mu)\Delta)$-stable.*

Proof. By Theorem 10, g is $2k\Delta$-close to a k-linear h on $\mathcal{D}_{2kn/\mu,s}$. For any $x = x_1, \ldots, x_k$, we fix all coordinates except x_i and argue in the ith coordinate as below.

For any $\mathcal{D}_{m,s}$, first we show that if $|e(x)|_{\mathcal{D}_{m,s}} \leq \rho$, then $|e(x)|_{\mathcal{D}_{m/2,s}} \leq (\rho+\Delta)/2$. To observe this, note that if $x \in \mathcal{D}_{m/2,s}$, then $2x \in \mathcal{D}_{m,s}$. Therefore the function should satisfy $e(x) + e(x) \approx_\Delta e(2x)$, which implies that $|e(x)| \leq (\rho+\Delta)/2$. Thus, in general, the maximum error in $\mathcal{D}_{m/2^i,s}$ is $\leq \rho/2^i + \Delta(1 - 1/2^i)$. Since the error in $\mathcal{D}_{2kn/\mu,s}$ is at most $2k\Delta$, the error in $\mathcal{D}_{n,s}$ is at most $(1+\mu)\Delta$ by our choice of parameters. In the multilinear case, we can make a similar argument by using points which have at least one coordinate x_i within the smaller half of the axis. □

3. Polynomials. To test programs purportedly computing polynomials, it is tempting to (i) interpolate the polynomial from randomly chosen points and then (ii) verify that the program is approximately equal to the interpolated polynomial for a large fraction of the inputs. Since a degree d k-variate polynomial can have $(d+1)^k$ terms, this leads to exponential running times. Furthermore, it is not obvious how error bounds that are *independent* of the domain size can be obtained.

Our test uses the same "evenly spaced" interpolation identity as that in [30]: f is a degree d polynomial if and only if for all $x, t \in \mathcal{D}$, $\sum_{i=0}^{d+1} (-1)^{d+1-i} \binom{d+1}{i} f(x + it) = 0$. This identity is computed by the method of successive differences which never explicitly interpolates the polynomial computed by the program, thus giving a particularly simple and efficient ($O(d^2)$ operations) test.

We can show that the interpolation identity is approximately robust by modifying the robustness theorem in [29] (section 3.3). Our proof of stability of the interpolation identity (section 3.2), however, uses a characterization of polynomials in terms of multilinear functions that previously has not been applied to program checking. This in turn allows us to use our results on the stability of multilinearity (section 2.2) and other ideas from stability theory. Section 3.4 extends these techniques to multivariate polynomials.

3.1. Preliminaries. In this section, we present the basic definitions and theorems that we will use. Define

$$\nabla_t f(x) \stackrel{\text{def}}{=} f(x+t) - f(x)$$

to be the standard *forward difference operator*. Let

$$\nabla_t^d f(x) \stackrel{\text{def}}{=} \overbrace{\nabla_t \cdots \nabla_t}^{d} f(x) = \sum_{k=0}^{d} (-1)^{d-k} \binom{d}{k} f(x + kt)$$

and $\nabla_{t_1,t_2} f(x) \stackrel{\text{def}}{=} \nabla_{t_1} \nabla_{t_2} f(x)$. The following are simple facts concerning this operator.

FACT 12. *The following are true for the difference operator* ∇*:*

1. ∇ *is linear:* $\nabla(f+g) = \nabla f + \nabla g$*;*
2. ∇ *is commutative:* $\nabla_{t_1,t_2} = \nabla_{t_2,t_1}$*; and*
3. $\nabla_{t_1+t_2} - \nabla_{t_1} - \nabla_{t_2} = \nabla_{t_1,t_2} = \nabla_{t_2,t_1}$.

Let $x^{[k]}$ denote $\overbrace{x,\ldots,x}^{k}$. For any k-ary symmetric f, let $f^*(x) = f(x^{[k]})$ denote its diagonal restriction. We use three different characterizations of polynomials [27, 15].

FACT 13. *Let* $\mathcal{D}$ *be a ring. The following are equivalent:*

1. *there exist* $a_0,\ldots,a_d \in \mathcal{D}$ *such that for all* $x \in \mathcal{D}$, $f(x) = \sum_{k=0}^{d} a_k x^k$*;*
2. *for all* $x,t \in \mathcal{D}$, $\nabla_t^{d+1} f(x) = 0$*;*
3. *there exist symmetric* k*-linear functions* F_k*,* $0 \le k \le d$*, such that for all* $x \in \mathcal{D}$, $f(x) = \sum_{k=0}^{d} F_k^*(x)$.

The above fact remains true for nonclosed domains so long as we insist that the arguments to f are from the domain.

The following definitions are motivated by the notions of using evenly and unevenly spaced points in interpolation.

DEFINITION 14 (strong approximate polynomial). *A function* g *is called* strongly Δ-approximately degree d polynomial on $\mathcal{D}$ *if for all* $x,t_1,\ldots,t_{d+1} \in \mathcal{D}$ *such that* $x + t_1 + \cdots + t_{d+1} \in \mathcal{D}$, $|\nabla_{t_1,\ldots,t_{d+1}} g(x)| \le \Delta$.

DEFINITION 15 (weak approximate polynomial). *A function* g *is called* weakly Δ-approximately degree d polynomial on $\mathcal{D}$ *if for all* $x,t \in \mathcal{D}$ *such that* $x+t(d+1) \in \mathcal{D}$, $|\nabla_t^{d+1} g(x)| \le \Delta$.

3.2. Stability for polynomials. First, we prove that if a function is strongly Δ-approximately polynomial, then there is a polynomial that $(2^{d \lg d}\Delta, 0)$-approximates it. Next, we show that if a function is weakly approximately polynomial on a domain, then there is a coarser subdomain on which the function is strongly approximately polynomial. Combining these two, we can show that if a function is weakly approximately polynomial on a domain, then there is a subdomain on which the function approximates a polynomial. By using Theorem 11, we can bring the above error arbitrarily close to Δ by assuming the hypothesis on a large enough domain. In order to pass programs that err by at most Δ', we need to set $\Delta \ge (d+1) \cdot 2^d \Delta'$.

Strongly approximate case. One must be careful in defining polynomial h that is close to g. For instance, defining h based on the values of g at some $d+1$ points will not work. We proceed by modifying techniques in [2, 19] using the following fact.

FACT 16. *If a function* f *is symmetric and* k*-linear, then* $\nabla_{t_1,\ldots,t_d} f^*(x) = k! f(t_1,\ldots,t_k)$ *if* $k = d$ *and* 0 *if* $k < d$.

The following theorem shows the stability of the strong approximate polynomial property.

THEOREM 17. *The strong approximate polynomial property is* $(\mathcal{D}_{n(d+2),s}, \mathcal{D}_{n,s}, \Delta, O(2^{d \lg d})\Delta)$*-stable. In other words, if* g *is a strongly* Δ*-approximately degree* d *polynomial on* $\mathcal{D}_{n(d+2),s}$*, then there is a degree* d *polynomial* h_d *such that* $\|g - h_d\|_{\mathcal{D}_{n,s}} \le O(2^{d \lg d})\Delta$.

Proof. Note that if $x,t_1,\ldots,t_{d+1} \in \mathcal{D}_{n,s}$, then $x + t_1 + \cdots + t_{d+1} \in \mathcal{D}_{(d+2)n,s}$. Now, the hypothesis that g is a strongly Δ-approximately degree d polynomial on $\mathcal{D}_{n(d+2),s}$ guarantees that for all $x,t_1,\ldots,t_{d+1} \in \mathcal{D}_{n,s}$, $|\nabla_{t_1,\ldots,t_{d+1}} g(x)| \le \Delta$. The rest of the proof uses this "modified hypothesis" and works with $\mathcal{D}_{n,s}$.

We induct on the degree. Let $e_d \stackrel{\text{def}}{=} |g - h_d|$. When $d = 0$, by the modified hypothesis, we have for all $x, t \in \mathcal{D}_{n,s}, |\nabla_t g(x)| \leq \Delta$, i.e., $|\nabla_t g(0)| = |g(t) - g(0)| \leq \Delta$ for all $t \in \mathcal{D}_{n,s}$. Setting $h_0 = g(0)$, a constant, we are done.

Suppose the lemma holds when the degree is strictly less than $d + 1$. Now, by the modified hypothesis, we have for all $t_1, \ldots, t_{d+1} \in \mathcal{D}_{n,s}$, $|\nabla_{t_1,\ldots,t_{d+1}} g(x)| \leq \Delta$. Using Fact 12 and then our modified hypothesis, we have $|\nabla_{t_1+t'_1,t_2,\ldots,t_d} g(x) - \nabla_{t_1,t_2,\ldots,t_d} g(x) - \nabla_{t'_1,t_2,\ldots,t_d} g(x)| = |\nabla_{t_1,t'_1,\ldots,t_d} g(x)| \leq \Delta$. By symmetry of the difference operator, we have a Δ-approximate symmetric d-linear function on $\mathcal{D}_{n,s}$, say, $G(t_1, \ldots, t_d) \stackrel{\text{def}}{=} \nabla_{t_1,\ldots,t_d} g(0)$. Theorem 10 on multilinearity guarantees a symmetric d-linear H with $\|G - H\| \leq 2d\Delta$. Let $H_d(x_1, \ldots, x_d) = H(x_1, \ldots, x_d)/d!$. Let $g'(x) = g(x) - H^*_d(x)$ for $x \in \mathcal{D}_{n,s}$.

Now, we have for all $x, t_1, \ldots, t_d \in \mathcal{D}_{n,s}$,

$$\begin{aligned}
|\nabla_{t_1,\ldots,t_d} g'(x)| &= |\nabla_{t_1,\ldots,t_d}(g(x) - H^*_d(x))| \quad \text{(definition of } g') \\
&\leq |\nabla_{t_1,\ldots,t_d} g(x) - \nabla_{t_1,\ldots,t_d} g(0)| \\
&\quad + |\nabla_{t_1,\ldots,t_d} g(0) - \nabla_{t_1,\ldots,t_d} H^*_d(x)| \quad \text{(triangle inequality)} \\
&= |\nabla_{t_1,\ldots,t_d,x} g(0)| + |\nabla_{t_1,\ldots,t_d} g(0) - \nabla_{t_1,\ldots,t_d} H^*_d(x)| \quad \text{(definition of } \nabla) \\
&= |\nabla_{t_1,\ldots,t_d,x} g(0)| + |G(t_1, \ldots, t_d) - \nabla_{t_1,\ldots,t_d} H^*_d(x)| \quad \text{(definition of } G) \\
&= |\nabla_{t_1,\ldots,t_d,x} g(0)| + |G(t_1, \ldots, t_d) - d! H_d(t_1, \ldots, t_d)| \quad \text{(Fact 16)} \\
&= |\nabla_{t_1,\ldots,t_d,x} g(0)| + |G(t_1, \ldots, t_d) - H(t_1, \ldots, t_d)| \quad \text{(definition of } H_d) \\
&\leq \Delta + |G(t_1, \ldots, t_d) - H(t_1, \ldots, t_d)| \quad \text{(modified hypothesis on } g) \\
&\leq (2d + 1)\Delta \quad \text{(since } \|G - H\| \leq 2d\Delta).
\end{aligned}$$

Now we apply the induction hypothesis. g' satisfies the hypothesis above for d and larger error $\Delta' = (2d+1)\Delta$ and so by induction, we are guaranteed the existence of a degree $d-1$ polynomial h_{d-1} such that $\|g' - h_{d-1}\| \leq e_{d-1}\Delta'$. Set $h_d = h_{d-1} + H^*_d$. By Fact 13(3) about the characterization of polynomials, h_d is a degree d polynomial. Now, $e_d = \|g - h_d\| = \|g - h_{d-1} - H^*_d\| = \|g' - h_{d-1}\| \leq e_{d-1}\Delta' = e_{d-1}(2d + 1)\Delta$.

Unwinding the recurrence, the final error $\|g - h_d\| = \Delta \prod_{i=1}^{d}(2i + 1)$. □

Weakly approximate case. We first need the following useful fact [15] which helps us to go from equally spaced points to unequally spaced points.

FACT 18. *For any $\lambda_1, \ldots, \lambda_d \in \{0, 1\}$, if $t'_{\lambda_1,\ldots,\lambda_d} = -\sum_{i=1}^{d} \lambda_i t_i/i$ and $t''_{\lambda_1,\ldots,\lambda_d} = \sum_{i=1}^{d} \lambda_i t_i$, then*

$$\nabla_{t_1,\ldots,t_d} f(x) = \sum_{\lambda_1,\ldots,\lambda_d \in \{0,1\}} (-1)^{\lambda_1+\cdots+\lambda_d} \nabla^d_{t'_{\lambda_1,\ldots,\lambda_d}} f(x + t''_{\lambda_1,\ldots,\lambda_d}).$$

Using this fact, we obtain the following theorem. Let $\mu(d) = \text{lcm}\{1, 2, \ldots, d\}$.

THEOREM 19. *If g is weakly $(\Delta/2^{d+1})$-approximately degree d polynomial on $\mathcal{D}_{n(d+1),s\mu(d+1)}$, then g is strongly Δ-approximately degree d polynomial on $\mathcal{D}_{n,s}$.*

Proof. For $t_1, \ldots, t_{d+1} \in \mathcal{D}_{n,s}$, and for any $\lambda_1, \ldots, \lambda_{d+1} \in \{0, 1\}$, we have by our choice of parameters that $t'_{\lambda_1,\ldots,\lambda_{d+1}}, t''_{\lambda_1,\ldots,\lambda_{d+1}} \in \mathcal{D}_{n(d+1),s\mu(d+1)}$. Therefore, for $x \in \mathcal{D}_{n,s}$,

$$\begin{aligned}
|\nabla_{t_1,\ldots,t_{d+1}} g(x)| &\leq \sum_{\lambda_1,\ldots,\lambda_{d+1} \in \{0,1\}} |\nabla^{d+1}_{t'_{\lambda_1,\ldots,\lambda_{d+1}}} g(x + t''_{\lambda_1,\ldots,\lambda_{d+1}})| \\
&\leq 2^{d+1}(\Delta/2^{d+1}) \leq \Delta. \quad \square
\end{aligned}$$

3.3. Approximate robustness for polynomials. This section shows that the interpolation equation for degree d polynomials is in some sense approximately robust. All the results in this subsection are modifications of the exact robustness of polynomials given in [29]. Let $\alpha_k = (-1)^{d+1-k}\binom{d+1}{k}$. To self-test P on $\mathcal{D}_{n,s}$, we use the following domains. These domains are used for technical reasons that will become apparent in the proofs of the theorems in this section.

1. $\mathcal{D}_{(d+2)n,s}$.
2. $\mathcal{T} = \mathcal{D}_{K_n,L_s}$ where $K_n = n(d+2)(n(d+1)!)^3$ and $L_s = s((d+1)!)^3$.
3. $\mathcal{T}_j = \{jx : x \in \mathcal{T}\}$ for $0 \le j \le d+1$.
4. $\mathcal{T}_{i,j} = \{ix : x \in \mathcal{T}_j\}$ for $0 \le i, j \le d+1$.

All $\mathcal{T}_j, \mathcal{T}_{i,j}$ contain $\mathcal{D}_{(d+2)n,s}$. Now, assume that P satisfies the following properties, which are similar to the low-degree test in an approximate setting and over different domains. Note that these properties can be tested by sampling. We use $\Pr_{x\in\mathcal{D}}[\cdot]$ to denote the probability of an event when x is chosen uniformly from domain $\mathcal{D}$.

1. $\displaystyle\Pr_{0\le k\le d+1, x\in\mathcal{D}_{n,s}, t\in\mathcal{T}_k}\left[\sum_{i=0}^{d+1}\alpha_i P(x+it) \approx_\Delta 0\right] \ge 1-\epsilon;$
2. for each $0 \le j \le d+1$, $\displaystyle\Pr_{0\le k,l\le d+1, x\in\mathcal{T}_{k,j}, t\in\mathcal{T}_l}\left[\sum_{i=0}^{d+1}\alpha_i P(x+it) \approx_\Delta 0\right] \ge 1-\epsilon;$ and
3. for each $0 \le i, j \le d+1$, $\displaystyle\Pr_{0\le k\le d+1, x\in\mathcal{T}_{i,j}, t\in\mathcal{T}_k}\left[\sum_{l=0}^{d+1}\alpha_l P(x+lt) \approx_\Delta 0\right] \ge 1-\epsilon.$

Define $g(x) = \text{median}_{0\le k\le d+1, t\in\mathcal{T}_k}\{\sum_{i=1}^{d+1}\alpha_i P(x+it)\}$. We obtain the following theorem that shows the approximate robustness of polynomials. Let $\mathcal{E}_{\tau(n,s)}$ be the distribution that flips a fair three-sided die and on outcome $i \in \{1,2,3\}$ chooses inputs according to distribution given in the ith property above. Let $\mathcal{D}_{\tau(n,s)}$ be the union of the domains used in the above properties.

THEOREM 20. *The interpolation equation, where inputs are picked according to the distribution $\mathcal{E}_{\tau(n,s)}$, is $(2\epsilon, \epsilon, \mathcal{D}_{\tau(n,s)}, \mathcal{D}_{n,s}, \Delta, 2^{d+3}\Delta, \Delta)$-approximately robust.*

The rest of this section is devoted to proving the above theorem.

By Markov's inequality, g's definition, and properties (1) and (3) of P, it is easy to show that P $(\Delta, 2\epsilon)$-approximates g.

THEOREM 21. *If program P satisfies the above three properties, then, for all $i, j \in \{0, \ldots, d+1\}$, $\Pr_{x\in\mathcal{T}_{i,j}}[P(x) \approx_\Delta g(x)] \ge 1-2\epsilon$ and $\Pr_{x\in\mathcal{D}_{n,s}}[P(x) \approx_\Delta g(x)] \ge 1-2\epsilon$.*

Now, we set out to prove that g is a weakly approximate polynomial. Let $\delta(p_1,p_2) = p_1$ if $p_1 = p_2$ and 0 otherwise. For two domains $\mathcal{A}, \mathcal{B}$, subsets of a universe $\mathcal{X}$, let $\delta(\mathcal{A},\mathcal{B}) = \sum_{s\in\mathcal{X}}\delta(\Pr_{x\in\mathcal{A}}[x=s], \Pr_{y\in\mathcal{B}}[y=s])$ and call the domains ϵ-*close* if $\delta(\mathcal{A},\mathcal{B})$ is at least $1-\epsilon$. Using the definitions of $\mathcal{T}, \mathcal{T}_j, \mathcal{T}_{i,j}$, the following fact can be shown.

FACT 22. *For any $x \in \mathcal{D}_{(d+2)n,s}$, the domains $\mathcal{T}_j$ and $\{x+t : t \in \mathcal{T}_j\}$ are $\epsilon_1 = O(1/n^3)$-close. For any x, the domains $\mathcal{T}_{i,j}$ and $\{x+t : t \in \mathcal{T}_{i,j}\}$ are $\epsilon_2 = O(1/n^3)$-close.*

The following lemma shows that, in some sense, g is well-defined and links it to an interpolation obtained from P.

LEMMA 23. *For all $x \in \mathcal{D}_{(d+2)n,s}$, $\Pr_{0\le k\le d+1, t\in\mathcal{T}_k}[g(x) \approx_{2^{d+2}\Delta} \sum_{j=1}^{d+1}\alpha_j P(x+jt)] \ge 1-\epsilon_3$ and for all i, $\Pr_{t\in\mathcal{T}_i}[g(x) \approx_{2^{d+2}\Delta} \sum_{j=1}^{d+1}\alpha_j P(x+jt)] \ge 1-\epsilon_4$, where $\epsilon_3 = 2(d+1)(\epsilon+\epsilon_2)$ and $\epsilon_4 = (d+1)\epsilon_3$.*

Proof. Consider $0 \le k, l \le d+1$, and $t_1 \in \mathcal{T}_k, t_2 \in \mathcal{T}_l$. For a fixed $0 \le j \le d+1$, using properties of P, and since $\mathcal{T}_{j,k}$ and $\{x + jt_1 : t_1 \in \mathcal{T}_k\}$ are ϵ_2-close (Fact 22), we get $\Pr[P(x+jt_1) \approx_\Delta \sum_{i=1}^{d+1} \alpha_i P(x+jt_1+it_2)] \ge 1-\epsilon-\epsilon_2$ and $\Pr[P(x+it_2) \approx_\Delta \sum_{j=1}^{d+1} \alpha_j P(x+jt_1+it_2)] \ge 1-\epsilon-\epsilon_2$. Summing over all $0 \le i,j \le d+1$ and noting that $\sum_{i=1}^{d+1} \alpha_j \Delta \le 2^{d+1}\Delta$, $\Pr[\sum_{j=1}^{d+1} \alpha_j P(x+jt_1) \approx_{2^{d+1}\Delta} \sum_{i=1}^{d+1} \alpha_i P(x+it_2)] \ge 1 - 2(d+1)(\epsilon+\epsilon_2) = 1-\epsilon_3$. Using Lemma 43 (see section 4.3), we can show that with a relaxation of twice the error, this probability lower bounds the probability in the first part of the lemma. The second part of the lemma follows from the first via a simple averaging argument. ∎

Now, the following theorem completes the proof that g is a weakly approximate degree d polynomial.

THEOREM 24. *For all $x \in \mathcal{D}_{(d+2)n,s}$, for all $i \in \{0,\dots,d+1\}$, $\Pr_{t\in\mathcal{T}_i}[g(x) \approx_{2^{d+3}\Delta} \sum_{j=1}^{d+1} \alpha_j g(x+jt)] \ge 1-\epsilon_5$, where $\epsilon_5 = \epsilon_4 + (d+1)(2\epsilon+\epsilon_2)$, and for all $x,t \in \mathcal{D}_{n,s}$, $\Pr_{t_1\in\mathcal{T}}[|\nabla_t^{d+1} g(x)| \le 2^{d+3}\Delta] \ge 1-(d+1)(2\epsilon_5+\epsilon_1)$.*

Proof. It is implied by Theorem 21, Lemma 23, and the closeness of the domains $\mathcal{T}_{i,j}$ and $\{x+t : t \in \mathcal{T}_{i,j}\}$ that for all $x \in \mathcal{D}_{(d+2)n,s}$, for all i, $\Pr_{t\in\mathcal{T}_i}[g(x) \approx_{2^{d+2}\Delta} \sum_{j=1}^{d+1} \alpha_j P(x+jt)] \ge 1-\epsilon_4$ and $\Pr_{t\in\mathcal{T}_i}[g(x+jt) \approx_\Delta P(x+jt)] \ge 1-2\epsilon-\epsilon_2$. Summing the latter expression and putting them together, we have the first part of the lemma. The second part follows from the first part and the fact that $\mathcal{T}_j$ and $\{t+jt_1 : t_1 \in \mathcal{T}\}$ are ϵ_1-close (Fact 22). ∎

For an appropriate choice of $\epsilon, \epsilon_1, \epsilon_2$, we have a g that is a weakly $(2^{d+3}\Delta)$-approximately degree d polynomial on $\mathcal{D}_{n,s}$ with g $(\Delta, 2\epsilon)$-approximating P on $\mathcal{D}_{n,s}$.

3.4. Multivariate polynomials. The following approach is illustrated for bivariate polynomials. We can easily generalize this to multivariate polynomials. It is easy to show that the approximate robustness holds when the interpolation equation [30] is used as in section 3.3, i.e., for any k-variate polynomial P of total degree d, the following interpolation equation is satisfied for all $\bar{x}, \bar{t} \in \mathcal{D}_{n,s}^k$:

$$\sum_{i=0}^{d+1} \alpha_i P(\bar{x} + i\bar{t}) = 0.$$

A *horizontal axis parallel line* for a fixed y is the set of points $l_{x,h} = \{(x+kh, y) : k \in \mathbb{Z}\}$. A vertical axis parallel line is defined analogously. As a consequence of approximate robustness, we have a bivariate function $g(x,y)$ that is a strongly approximately degree d polynomial along every horizontal and vertical line. We use this consequence to prove stability.

The characterization we will use is as follows: $f(x,y)$ is a bivariate polynomial (assume degree in both x and y is d) if and only if there are $d+1$ symmetric k-linear functions $F_k(y_1,\dots,y_k) : \mathcal{D}^k \to \mathcal{P}_\mathcal{D}[x]$, where the range is the space of all degree d univariate polynomials in x.

For each value of y, $g_y(x)$ is a strongly approximately degree d polynomial. Using the univariate case (Theorem 17), there is an exact degree d polynomial $P_y(x)$ such that for all x, $g(x,y) \approx_{2^{d\lg d}\Delta} P_y(x)$. Construct the function $g'(x,y) = P_y(x)$. Let $\Delta' = 2^{d\lg d}\Delta$. Now, for a fixed x (i.e., on vertical line) for any y, using $\nabla_{t_1,\dots,t_{d+1}} g(x,y) \approx_\Delta 0$, we have $\nabla_{t_1,\dots,t_{d+1}} g'(x,y) \approx_{\Delta'} 0$. Thus, $g'(x,y)$ is a bivariate function where along every horizontal line, it is an *exact* degree d polynomial and along every vertical line, it is a strongly Δ'-approximate degree d polynomial. Interpreting $g'(x,y)$ as $g'_x(y)$ and using the same idea as in univariate case, we can conclude that $\nabla(t_1,\dots,t_d)$:

$\mathcal{D}^d \to \mathcal{P}_{\mathcal{D}}[x]$ is a symmetric approximate d-linear function (here, we used the fact that $g'_x(y) \in \mathcal{P}_{\mathcal{D}}[x]$). The rest of the argument in Theorem 17 goes through because our proofs of approximate linearity (Lemma 7) and multilinearity (Theorem 10) assume that the range is a metric space (which is true for $\mathcal{P}_{\mathcal{D}}[x]$ with, say, the Chebyshev norm). The result follows from the above characterization of bivariate polynomials.

4. Functional equations. Extending the technique in Lemma 7 to addition theorems $f(x+y) = G[f(x), f(y)]$ is not straightforward, since G can be an arbitrary function. In order to prove approximate robustness (section 4.3) and stability (section 4.2), several related properties of G are required. Proving that G satisfies each individual one is tedious; however, the notion of *modulus of continuity* from approximation theory gives a general approach to this problem. We show that bounds on the modulus of continuity imply bounds on all of the quantities of G that we require. The stability of G is shown by a careful inductive technique based on a canonical generation of the elements in $\mathcal{D}_{n,s}$ (section 4.2). The scope of our techniques is not only limited to addition theorems; we also show that Jensen's equation is approximately robust and stable (section 4.2.4).

4.1. Preliminaries. For addition theorems, we can assume that G is algebraic and a symmetric function (the latter is true, in general, under some technical assumptions as in [28]). We need a notion of "smoothness" of G. The following notions are well known in approximation theory [25, 33].

DEFINITION 25 (moduli of continuity). *The* modulus of continuity *of the function $f : \mathcal{D} \to \mathbb{R}$ is the following function of $\delta \in [0, \infty)$:*

$$\omega(f; \delta) = \sup_{\substack{|x_1 - x_2| \le \delta \\ x_1, x_2 \in \mathcal{D}}} \{|f(x_1) - f(x_2)|\}.$$

The modulus of continuity *of the function $f : \mathcal{D}^2 \to \mathbb{R}$ is the following function of $\delta_x, \delta_y \in [0, \infty)^2$:*

$$\omega(f; \delta_x, \delta_y) = \sup_{\substack{|x_1 - x_2| \le \delta_x, |y_1 - y_2| \le \delta_y \\ x_1, y_1, x_2, y_2 \in \mathcal{D}}} \{|f(x_1, y_1) - f(x_2, y_2)|\}.$$

The partial moduli of continuity *of the function $f : \mathcal{D}^2 \to \mathbb{R}$ are the following functions of $\delta \in [0, \infty)$:*

$$\omega(f; \delta, 0) = \sup_{y \in \mathcal{D}} \sup_{\substack{|x_1 - x_2| \le \delta \\ x_1, x_2 \in \mathcal{D}}} \{|f(x_1, y) - f(x_2, y)|\} \text{ and}$$

$$\omega(f; 0, \delta) = \sup_{x \in \mathcal{D}} \sup_{\substack{|y_1 - y_2| \le \delta \\ y_1, y_2 \in \mathcal{D}}} \{|f(x, y_1) - f(x, y_2)|\}.$$

We now present some facts which are easily proved.

FACT 26. *The following are true of the modulus of continuity:*

1. $0 \le \omega(f; \delta) \le \omega(f; \delta')$ *if* $\delta \le \delta'$;
2. *if f', the derivative of f, exists and is bounded in $\mathcal{D}$, then $\omega(f; \delta) \le \delta \|f'\|_{\mathcal{D}}$;*
3. $\omega(f; \delta, \delta) \le \omega(f; 0, \delta) + \omega(f; \delta, 0)$, *and if $f(\cdot, \cdot)$ is symmetric, then $\omega(f; \delta, \delta) \le 2\omega(f; \delta, 0)$; and*
4. *if f'_x is the partial derivative of f with respect to x, then $\omega(f; \delta, 0) \le \delta \|f'_x\|_{\mathcal{D}}$.*

We need a notion of an "inverse" of G. If $G[x, y] = z$, denote $G_1^{-1}[z, y] = x, G_2^{-1}[x, z] = y$. Since G is symmetric, $G_1^{-1} \equiv G_2^{-1}$ and we denote $G^{-1}[z, y] = x$.

An example. Wherever necessary, we will illustrate our scheme using the functional equation $f(x+y) = \frac{f(x)f(y)}{f(x)+f(y)}$, i.e., $G[x,y] = xy/(x+y)$. The solution to this functional equation is $f(x) = C/x$ for some constant C. The following fact [34] is useful in locating the maxima of analytic functions.

FACT 27 (maximum modulus principle). *If f is analytic in a compact set $\mathcal{D}$, then f attains extremum only on the boundary of $\mathcal{D}$.*

Over a bounded rectangle $\mathcal{D} = [L,U]^2$, where $0 < L \le U$, G is analytic, and hence by Fact 27 attains its maximum on the boundary. $G \in C^1[L,U]$ in $\mathcal{D}$ (i.e., is continuously differentiable). We have $G'_x[x,y] = y^2/(x+y)^2$ which is a decreasing function of x. By Fact 27, $\|G'_x\|$ attains a maximum when $x = L$, giving $\widehat{G} = \|G'_x(\cdot,y)\| = y^2/(L+y)^2$. Therefore, using Fact 26(4), $\omega(G;\delta,0) \le \sup_{y\in[L,U]} \frac{\delta y^2}{(L+y)^2} = \frac{\delta U^2}{(L+U)^2}$.

4.2. Stability for functional equations. In this section, we prove (under some assumptions) that if a function g satisfies a functional equation approximately everywhere, then it is close to a function h that satisfies the functional equation exactly everywhere. Our functional equations are of the form $g(x+y) = G[g(x), g(y)]$, where G is a symmetric algebraic function.

Example. If g satisfies $g(x+y) \approx_\Delta \frac{g(x)g(y)}{g(x)+g(y)}$ for some $\Delta > 0$ and for all valid x, y, then there is a function h such that $h(x+y) = \frac{h(x)h(y)}{h(x)+h(y)}$ for all valid x, y, and $h(x) \approx_{\Delta'} g(x)$ for some $\Delta' > 0$ and all valid x. The domains for the valid values of x, y, as well as the relationship between Δ and Δ', will be discussed later.

In the following sections we show how to construct the function h that is close to g, satisfying a particular functional equation. Given such an h, let $e(x)$ denote $|h(x) - g(x)|$, i.e., $h(x) \approx_{e(x)} g(x)$. For simplicity, let $H_1(x) \stackrel{\text{def}}{=} G[x,x]$. Note that $H_1(h(x)) = h(2x)$. We assume that $\omega(H_1;\delta) \le c\delta$; our results thus hold for functions where the modulus of continuity is linear in δ. We will be making this assumption for our moduli of continuity when appropriate.

We consider the cases when $c < 1$, $c = 1$, and $c > 1$. We first show how to obtain h and then obtain bounds on $e(x)$. Then, we can conclude that h, which satisfies the functional equation everywhere, also approximates g; i.e., the functional equation is stable.

Call $\frac{x}{s}$ even (resp., odd) if x is even (resp., odd).

4.2.1. When $c < 1$. We begin by assuming that n is a power of 2, i.e., let $n = 2^k$ in $\mathcal{D}_{n,s}$. We first construct h by setting $h(\frac{1}{s}) = g(\frac{1}{s})$. This determines h for all values in $\mathcal{D}$ by the fact that h satisfies the functional equation.

We obtain a relationship between the error at x and $2x$ using the functional equation.

LEMMA 28. $e(2x) \le ce(x) + \Delta$.

Proof. $e(2x) = |g(2x) - h(2x)| \le \Delta + |G[g(x),g(x)] - G[h(x),h(x)]|$. However, rewriting, and using the definition of the modulus of continuity, $|H_1(g(x)) - H_1(h(x))| \le \omega(H_1; e(x)) \le ce(x)$. ∎

We explore the relationship between $e(x + \frac{1}{s})$ and $e(x)$. For simplicity, let $H_2(x) \stackrel{\text{def}}{=} G[x, g(\frac{1}{s})]$. Note that $H_2(h(x)) = h(x+1)$. We again consider functions where the modulus of continuity is bounded by a linear function in δ, i.e., $\omega(H_2;\delta) = |H'_2(\cdot, g(\frac{1}{s}))| \le d\delta$ for some constant d. Now, we introduce the following lemma.

LEMMA 29. $e(x+\frac{1}{s}) \le de(x) + \Delta$.

Proof. $e(x+\frac{1}{s}) = |g(x+\frac{1}{s}) - h(x+\frac{1}{s})| \le \Delta + |G[g(x), g(\frac{1}{s})] - G[h(x), h(\frac{1}{s})]|$. However, $|G[g(x), g(\frac{1}{s})] - G[h(x), h(\frac{1}{s})] = |H_2[g(x)] - H_2[h(x)]| \le \omega(H_2; e(x)) \le de(x)$. □

We will show a scheme to bound $e(x)$ for all x when $d < 1$. This scheme can be thought of as an enumeration scheme, where at each step of the process, certain constraint equations have to be satisfied. We construct a binary tree T_k with nodes labeled with elements from $\mathcal{D}^+_{n,s}$ where $2^k = n$. The root is labeled $\frac{1}{s}$. If x is the label of a node, then $2x$ is the label of its left child (if $2x$ is not already in the tree), and $x + \frac{1}{s}$ is the label of its right child (if $x + \frac{1}{s}$ is not already in the tree). It is easy to see that if x is even (except root), then x is a left child; if x is odd, then x is a right child.

LEMMA 30. *Let $\omega(H_1; \delta) \le c\delta, \omega(H_2; \delta) \le d\delta$ with $c, d < 1$. For all $x \in \mathcal{D}^+_{n,s}$, if x is even, then $e(x) \le \frac{1+c}{1-c}\Delta$; and if x is odd, then $e(x) \le \frac{2}{1-c}\Delta$.*

Proof. We will prove this by induction on the preorder enumeration of T_k. Let x be the next element to be enumerated. By preorder listing, its parent has already been enumerated, and hence its error is known. If $x = 2y$ is even, it is a left child and hence generated by a $2y$ operation. $e(y) \le \frac{2}{1-c}\Delta$ by the induction hypothesis. This together with Lemma 28 yields $e(x) \le ce(y) + \Delta \le c\frac{2}{1-c}\Delta + \Delta \le \frac{1+c}{1-c}\Delta$, preserving the induction hypothesis. If $x = y + \frac{1}{s}$ is odd, it is a right child, and hence generated by a $y + 1$ operation. However, y is even, so $e(y) \le \frac{1+c}{1-c}\Delta$ by the induction hypothesis. This together with Lemma 29 and $d \le 1$ yields $e(x) \le de(y) + \Delta \le e(y) + \Delta \le \frac{1+c}{1-c}\Delta + \Delta \le \frac{2}{1-c}\Delta$, preserving the induction hypothesis. □

This yields the following theorem.

THEOREM 31. *Let $\omega(H_1; \delta) \le c\delta, \omega(H_2; \delta) \le d\delta$ with $c, d < 1$ and let n be a power of 2. Then, the addition theorem is $(\mathcal{D}^+_{n,s}, \mathcal{D}^+_{n,s}, \Delta, \frac{2}{1-c}\Delta)$-stable.*

With our example, we have $H_1(x) = G[x, x] = x/2$ and so $c = 1/2$. Also, $H_2(x) = G[x, g(\frac{1}{s})]$ from which $H_2'(x) \le 1$ as $0 < L \le x, g(\frac{1}{s})$. Thus, $d \le 1$. By Theorem 31, we have $e(x) \le 4\Delta$ for all $x \in \mathcal{D}^+_{n,s}$.

When n is not a power of 2, we can argue in the following manner. From our proof, we see that we use very specific values of x, y in the approximate functional equation. Let i be such that $2^{i-1} \le n \le 2^i$ and let $\mathcal{D}' = \mathcal{D}_{2^i,s}$. We extend $\mathcal{D}^+_{n,s}$ to $\mathcal{D}'$ and define values of g at $\mathcal{D}' \backslash \mathcal{D}$: at even x $(= 2y)$ let $g(x) = H_1(g(y))$ and at odd x $(= y + 1)$ let $g(x) = H_2(g(y))$. These can be thought of as new assumptions on g which are satisfied "exactly" (i.e., without error Δ). We can use Lemma 30 to conclude that there is a linear h on $\mathcal{D}'$ that is $\frac{2}{1-c}\Delta$ close to g. Hence, h is close to g even on $\mathcal{D}^+_{n,s}$. To argue about $\mathcal{D}^-_{n,s}$, we pick a "pivot" point in $\mathcal{D}_{n,s}$ (0 for simplicity). Now, we have $h(-x) = G^{-1}[h(x), h(0)]$. Therefore, as in Theorem 31, we have $e(-x) \le \omega(G^{-1}; \frac{2c}{1-c})$.

When $d \ge 1$, the error can no longer be bounded. In this case, we have $c \le 1 < d$. Let $r = cd$. We can see from the structure of T_k that the maximum error can occur at $\frac{2^k-1}{s}$. By simple induction on the depth of the tree, the error is given by $e(\frac{2^k-1}{s}) \le \sum_{i=0}^{k-2}(d^{i+1} + d^i)c^i\Delta = (d+1)\sum_{i=0}^{k-2} r^i\Delta = (d+1)\frac{r^{k-1}-1}{r-1}\Delta$. If $r < 1$, we obtain a constant error bound of $e(x) \le (d+1)\frac{1}{1-r}\Delta$ by geometric summation. Otherwise, we obtain $e(x) = O(r^{\lg n})$.

4.2.2. When $c > 1$. In this case, we require additional assumptions. We define the quantity

$$\omega^{-1}(f;\delta) = \sup_{\substack{|f(x_1)-f(x_2)|\le\delta \\ x_1,x_2\in\mathcal{D}}} \{|x_1 - x_2|\}.$$

Note that $\omega(f;\delta) \le c\delta$ implies $\omega^{-1}(f;\delta) \ge \delta/c$. Now, we assume that $\omega^{-1}(f;\delta) \le \delta/c'$ for some $c' > 1$.

Set $h(\frac{2^k}{s}) = g(\frac{2^k}{s})$. Since h satisfies the addition theorem, this can be used to fix all of h if H_1^{-1} is well-defined. Let $e(x) = |g(x) - h(x)|$ as before.

As before, we first obtain a relationship between the error at x and at $2x$ using the addition theorem.

LEMMA 32. $e(x) \le (e(2x) + \Delta)/c'$.

Proof. We have as in Lemma 28, $|H_1(g(x)) - H_1(h(x))| \le e(2x) + \Delta$. By definition of ω^{-1} and our assumption, we get $e(x) \le \omega^{-1}(H_1; e(2x) + \Delta) \le (e(2x) + \Delta)/c'$. ∎

For simplicity, let $H_3(x) \stackrel{\text{def}}{=} G^{-1}[g(\frac{2^k}{s}), \frac{2^k}{s} - x]$. We assume that $\omega(H_3;\delta) \le d\delta$ for some constant d. The following lemma can be proved easily.

LEMMA 33. $e(x) \le de(\frac{2^k}{s} - x) + \Delta$.

$e(\frac{2^k}{s}) = 0$ by our construction. We adopt a scheme similar to the one in the previous section. Construct a binary tree T_k with nodes labeled with elements from $\mathcal{D}^+_{n,s}$. The root is labeled $\frac{2^k}{s}$. If x is the label of a node and x is even, then $x/2$ is the label of its left child (if $x/2$ is not already in the tree), and $\frac{2^k}{s} - x$ is the label of its right child (if $\frac{2^k}{s} - x$ is not already in the tree). It is easy to see that if $x \le \frac{2^{k-1}}{s}$ (except the root), then x is a left child, and if $x > \frac{2^{k-1}}{s}$, then x is a right child. We use the preorder enumeration of $\mathcal{D}^+_{n,s}$ using T_k to prove the following lemma in the spirit of the proof of Lemma 30.

LEMMA 34. *For all $x \in \mathcal{D}^+_{n,s}$, if $x \le \frac{2^{k-1}}{s}$ and $d \le 1$, then $e(x) \le \frac{2c'}{1-c'}\Delta$; and if $x > \frac{2^{k-1}}{s}$, then $e(x) \le \frac{1+c'}{1-c'}\Delta$.*

This yields (under the assumptions on $\omega^{-1}(H_1;\delta)$ and $\omega(H_3;\delta)$) the following theorem.

THEOREM 35. *Let $\omega^{-1}(H_1;\delta) \le \delta/c', \omega(H_3,\delta) \le d\delta$ with $d \le 1$ and let n be a power of 2. Then, the addition theorem is $(\mathcal{D}^+_{n,s}, \mathcal{D}^+_{n,s}, \Delta, \frac{1+c'}{1-c'}\Delta)$-stable.*

This case arises for linearity where $H_1(x) = G[x,x] = 2x$ and so $c' = 2$. Using the above theorem, we get a weaker bound of $e(x) \le 3\Delta$ (as opposed to $\le 2\Delta$ by Corollary 8). Similar techniques as in the previous section can be used to argue about $\mathcal{D}^-_{n,s}$ and when n is not a power of 2.

The case when $d > 1$ can be handled by schemes as in the previous section.

4.2.3. When $c = 1$. In this case, it means that $\omega(H_1;\delta) = \delta$ or in other words, by Fact 26(2), $\|H_1'\| = 1$. By Fact 27, the maximum occurs only at the boundary of the domain. Hence, we can test by looking at a subdomain in which the maximum is less than 1.

4.2.4. Jensen's equation. Jensen's equation is the following: for all $x, y \in \mathcal{D}_{n,s}, f(\frac{x+y}{2}) = \frac{f(x)+f(y)}{2}$. The solution to this functional equation is the set of affine linear functions, i.e., $f(x) = ax + b$ for some constants a, b. Jensen's equation can be proved approximately robust by modifying the proof of its robustness in [28]. We will show a modified version of our technique for proving its stability. As before, we have

for all $x, y \in \mathcal{D}_{n,s}, g(\frac{x+y}{2}) \approx_\Delta \frac{g(x)+g(y)}{2}$. To prove the stability of this equation, we construct an affine linear h. Note that two points are necessary and sufficient to fully determine h. We set $h(\frac{n}{s}) = g(\frac{n}{s})$ and $h(0) = g(0)$.

LEMMA 36. $e(\frac{x+y}{2}) \le e(x)/2 + e(y)/2 + \Delta$.

Proof. $e(\frac{x+y}{2}) = |g(\frac{x+y}{2}) - h(\frac{x+y}{2})| \le \Delta + |\frac{g(x)+g(y)}{2} - \frac{h(x)+h(y)}{2}|$. However, $|\frac{g(x)-h(x)}{2} + \frac{g(y)-h(y)}{2}| = e(x)/2 + e(y)/2$. ∎

The following corollary is immediate.

COROLLARY 37. $e(\frac{x}{2}) \le \Delta + e(x)/2$ *and* $e(\frac{x+\frac{n}{s}}{2}) \le \Delta + e(x)/2$.

Proof. Since for $y = 0$ and $y = \frac{n}{s}$, $e(y) = 0$ in Lemma 36. ∎

We construct a slightly different tree T_k in this case. The root of T_k is labeled by $\frac{n}{s}$ and if x is the label of a node, then $x/2$ (if integral and not already present) is label of its left child and $(\frac{n}{s} + x)/2$ (if integral and not already present) is the label of its right child.

THEOREM 38. *The Jensen equation is* $(\mathcal{D}^+_{n,s}, \mathcal{D}^+_{n,s}, \Delta, 2\Delta)$*-stable.*

Proof. The proof is by induction on an enumeration order of T_k given by, say, a breadth-first traversal. Clearly, at the root, $e(\frac{n}{s}) = 0 \le 2\Delta$. Now, if $e(x) \le 2\Delta$, then consider its children. Its left (resp., right) child (if exists) is $x/2$ (resp., $(x + \frac{n}{s})/2$). Thus, by Corollary 37, we have $e(\frac{x}{2}) \le \Delta + e(x)/2 \le 2\Delta$ (resp. $e(\frac{x+\frac{n}{s}}{2}) \le \Delta + e(x)/2 \le 2\Delta$). ∎

4.3. Approximate robustness for functional equations. As in [20, 29], we test the program on $\mathcal{D}_{2p,s}$ and make conclusions about its correctness on $\mathcal{D}_{n,s}$. The relationship between p and n will be determined later. The domain has to be such that G is analytic in it. Therefore, we consider the case when f is bounded on $\mathcal{D}_{2p,s}$, i.e., $0 < L \le f(x) \le U$. Let $\mathcal{G}$ be the family of functions f that satisfy the following conditions:

(1) $\Pr_{x \in \mathcal{D}_{2p,s}}[f(x) \ge L] \ge 1 - \epsilon$,
(2) $\Pr_{x \in \mathcal{D}_{2p,s}}[f(x) \le U] \ge 1 - \epsilon$,
(3) $\Pr_{x,y \in \mathcal{D}_{2p,s}}[G[f(x), f(y)] \ge L] \ge 1 - \epsilon$, and
(4) $\Pr_{x,y \in \mathcal{D}_{2p,s}}[G[f(x), f(y)] \le U] \ge 1 - \epsilon$.

Note that the membership in $\mathcal{G}$ is easy to determine by sampling. We can define a distribution $\mathcal{E}_{\tau(n,s)}$ such that if P satisfies the functional equation on $\mathcal{E}_{\tau(n,s)}$ with probability at least $1 - \epsilon$, then P also satisfies the following four properties:

(1) $\Pr_{x,y \in \mathcal{D}_{p,s}}[P(x + y) \approx_\Delta G[P(x), P(y)]] \ge 1 - \epsilon$,
(2) $\Pr_{x,y \in \mathcal{D}_{p,s}}[P(x) \approx_\Delta G[P(x - y), P(y)]] \ge 1 - \epsilon$,
(3) $\Pr_{x,y \in \mathcal{D}_{p,s}}[P(x) \approx_\Delta G[P(y), P(x - y)]] \ge 1 - \epsilon$, and
(4) $\Pr_{x \in \mathcal{D}_{n,s}, y \in \mathcal{D}_{p,s}}[P(x) \approx_\Delta G[P(x - y), P(y)]] \ge 1 - \epsilon$.

$\mathcal{E}_{\tau(n,s)}$ is defined by flipping a fair four-sided die and on outcome $i \in \{1, 2, 3, 4\}$ choosing inputs according to the distribution given in the ith property above. Let $\widehat{G} = \|G'_x\|_{\mathcal{D}}$. We can then show the following.

THEOREM 39. *The addition theorem with the distribution* $\mathcal{E}_{\tau(n,s)}$ *is* $(2\epsilon, \epsilon, \mathcal{D}_{2p,s}, \mathcal{D}_{n,s}, \Delta, (9\widehat{G}^2 + 5\widehat{G})\Delta, \Delta)$*-approximately robust.*

Define for $x \in \mathcal{D}_{p,s}$, $g(x) = \text{median}_{y \in \mathcal{D}_{p,s}}\{G[P(x - y), P(y)]\}$. By Markov's inequality, definition of g, and the properties of P, it is easy to show the following.

LEMMA 40. $\Pr_{x \in \mathcal{D}_{n,s}}[g(x) \approx_\Delta P(x)] > 1 - 2\epsilon$.

Proof. Consider the set of elements $x \in \mathcal{D}_{n,s}$ such that $\Pr_{y \in \mathcal{D}_{p,s}}[P(x) \approx_\Delta G[P(x - y), P(y)]] < \frac{1}{2}$. If the fraction of such elements is more than 2ϵ, then it contradicts hypothesis (4) on P that $\Pr_{x \in \mathcal{D}_{n,s}, y \in \mathcal{D}_{p,s}}[P(x) \approx_\Delta G[P(x - y), P(y)]] \ge 1 - \epsilon$. For the

rest, for at least half of the y's, $P(x) \approx_\Delta G[P(x-y), P(y)]$. By defining g to be the median (over y's in $\mathcal{D}_{p,s}$), we have for these elements $g(x) \approx_\Delta P(x)$. □

For simplicity of notation, let P_x denote $P(x)$ for any $x \in \mathcal{D}_{p,s}$ and $G_{x,y}$ denote $G[P(x), P(y)]$ for any $x, y \in \mathcal{D}_{p,s}$. Since G is fixed, we will drop G from the modulus of continuity.

A distribution U' on $\mathcal{D}$ is said to be *ϵ-uniform on $\mathcal{D}$* if $\sum_{x\in\mathcal{D}} |U'(x) - 1/|\mathcal{D}|| \le \epsilon$. Let $\gamma = n/2p$.

FACT 41. (1) *For all $x \in \mathcal{D}_{2n,s}$, the distribution of $x + y$ is γ-uniform on $\mathcal{D}_{p,s}$.*

(2) *For any event $E(x)$ and for an ϵ-uniform distribution U' on $\mathcal{D}$,* $|\Pr_{x\sim U'}[E(x)] - \Pr_{x\in\mathcal{D}}[E(x)]| \le \epsilon$.

LEMMA 42. *For $x \in \mathcal{D}_{2n,s}$,* $\Pr_{y,z\in\mathcal{D}_{p,s}}[G_{x-y,y} \approx_{2\omega(\Delta,0)} G_{x-z,z}] \ge 1 - 12\epsilon - 4\gamma$.

Proof. $\Pr_{y,z\in\mathcal{D}_{p,s}}[G_{x-y,y} \approx_{\omega(\Delta,0)} G[G_{x-z,z-y}, P_y] = G[P_{x-z}, G_{z-y,y}] \approx_{\omega(0,\Delta)} G_{x-z,z}] > 1 - 12\epsilon - 4\gamma$. The error in the first step (due to computation of P_{x-y}) is $\omega(\Delta, 0)$ and the equation holds with probability at least $1 - \epsilon - \gamma$ by property (3) and Fact 41. The bounds on $G_{x-z,z-y}$ also hold with probability at least $1 - 2\epsilon - 2\gamma$ by properties (3), (4), and Fact 41 and so the error is just $\omega(\Delta, 0)$. The next line is just rewriting. In a similar manner, the final equation holds with probability at least $1 - \epsilon - \gamma$ by property (2) and Fact 41 and the error bound is $\omega(0, \Delta)$. The bounds on random points $P_y, P_z, P_{x-z}, P_{z-y}$ hold with probability at least $1 - 8\epsilon$ by properties (1), (2) on P to make the error $\omega(0, \Delta)$. Hence, the total error is $\omega(\Delta, 0) + \omega(0, \Delta) = 2\omega(\Delta, 0)$ by Fact 26(3) and the equality holds with probability at least $1 - 12\epsilon - 4\gamma$. □

The following lemma, which helps us to bound the error, is from [23]. The proof uses the observation that the clique number of G^2 is at least as big as the maximum degree in G. Hence, for a random node x, probability that x is present in the largest clique in G^2 is more than the probability that x is connected to the maximum degree vertex (say, y) in G.

LEMMA 43 (see [23]). *If $G = \langle V, E\rangle$ is a random graph with edges inserted with probability $1 - \epsilon$, then $G^2 = \langle V, \{(x, y) : \exists z \in V, (x, z) \in E \wedge (z, y) \in E\}\rangle$ is a graph where the largest clique is of size at least $(1 - \epsilon)|V|$.*

The following shows, in some sense, that g is well-defined.

LEMMA 44. *For all $x \in \mathcal{D}_{2n,s}$,* $\Pr_{y\in\mathcal{D}_{p,s}}[g(x) \approx_{2\Delta'} G_{x-y,y}] \ge 1 - 12\epsilon - 4\gamma$, *where* $\Delta' = 2\omega(\Delta, 0)$.

Proof. We have the following: for all $x \in \mathcal{D}_{2n,s}$, $\Pr_{y,z\in\mathcal{D}_{p,s}}[G_{x-y,y} \approx_{\Delta'} G_{x-z,z}] \ge 1 - 12\epsilon - 4\gamma$. Now, we use Lemma 43. If G denotes a graph in which (y, z) is an edge if and only if $G_{x-y,y} \approx_{\Delta'} G_{x-z,z}$, then G^2 denotes the graph in which (y, z) is an edge if and only if $G_{x-y,y} \approx_{2\Delta'} G_{x-z,z}$. Now, using Lemma 43, we have that the number of elements that are $2\Delta'$ away from the largest clique is at most 2ϵ. Thus, at least $1 - 2\epsilon$ of elements are within $2\Delta'$ of each other. If $\epsilon < 1/2$ and since $g(x)$ is the median, the lemma follows. □

Now, the following theorem completes the proof that g satisfies the addition theorem approximately.

THEOREM 45. *For all $x, y \in \mathcal{D}_{n,s}$, $g(x + y) \approx_{\Delta''} G[g(x), g(y)]$ with probability at least $1 - 56\epsilon - 14\gamma$, where $\Delta'' = (9\widehat{G}^2 + 5\widehat{G})\Delta$.*

Proof.

$$\begin{aligned}
\Pr_{u,v\in\mathcal{D}_{p,s}}[G[g(x),g(y)] &\approx_{\omega(2\Delta',2\Delta')} G[G_{u,x-u},G_{v,y-v}]\\
&= && G[P_u, G[P_{x-u}, G_{v,y-v}]]\\
&= && G[P_u, G[G_{x-u,v}, P_{y-v}]]\\
&\approx_{\omega(0,\omega(\Delta,0))} && G[P_u, G_{x-u+v,y-v}]\\
&\approx_{\omega(0,\Delta)} && G_{u,x+y-u}\\
&\approx_{2\Delta'} && g(x+y)] \geq 1-56\epsilon-14\gamma.
\end{aligned}$$

By Lemma 44, the first equality holds with probability $1-24\epsilon-8\gamma$ and error $\omega(2\Delta',2\Delta')$. By property (4), the bounds on $G_{u,x-u}, G_{v,y-v}$ hold with probability at least $1-4\epsilon$ to make the error $\omega(2\Delta',2\Delta') \leq 2\omega(2\Delta',0) = 4\omega(\Delta',0) = 8\omega(\omega(\Delta,0),0)$ by Fact 26(3). The second and third equalities are always true. The fourth equality holds with probability at least $1-\epsilon-\gamma$ by property (1) and Fact 41 on P and the error accrued is $\omega(0,\omega(\Delta,0))$. The bounds on $P_u, P_{x-u}, P_v, P_{y-v}, G_{x-u,v}$ hold with probability at least $1-10\epsilon$ by properties (1)–(4) to make the error $\omega(0,\omega(\Delta,0)) = \omega(\omega(\Delta,0),0)$. The fifth equality also holds with probability at least $1-\epsilon-\gamma$ by property (1) on P and the error accrued is $\omega(0,\Delta) = \omega(\Delta,0)$, after bounds on P_u, P_{x+y-u} (with probability at least $1-4\epsilon$). The final equality holds with probability at least $1-12\epsilon-4\gamma$ by Lemma 44 and error is $2\Delta' = 4\omega(\Delta,0)$. Thus, the total error is $9\omega(\omega(\Delta,0),0) + 5\omega(\Delta,0)$. However, $\omega(\Delta,0) \leq \Delta\widehat{G}$ by Fact 26(4). Hence, $\omega(\omega(\Delta,0),0) \leq \Delta\widehat{G}^2$. □

If $\epsilon < 1/112$, $p > 14n$, we have $1-56\epsilon-14\gamma > 0$ and so the above lemma is true with probability 1. In the case of our example function, we already calculated $\widehat{G} = U^2/(L+U)^2$. Hence, $\Delta'' = \Delta(\frac{9U^4}{(L+U)^4} + \frac{5U^2}{(L+U)^2})$.

5. Approximate self-testing and self-correcting. In this section we briefly show how to apply our techniques that we developed in this paper to construct approximate self-tester and self-correctors. The approaches in this section follow [8, 20].

5.1. Definitions. The following modifications of definitions from [20] capture the idea of approximate checking, self-testing, and self-correcting in a formal manner. Let P be a program for f, $x \in \mathcal{D}_{n,s}$ an input to P, and β the confidence parameter.

DEFINITION 46. *A* $(\Delta_1,\Delta_2,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate result checker *for* f *is a probabilistic oracle program* T *that, given* P, $x \in \mathcal{D}_{n,s}$, *and* β, *satisfies the following:*

(1) P $(\Delta_1,0)$-*approximates* f *on* $\mathcal{D}_{\tau(n,s)} \Rightarrow \Pr[T^P$ *outputs "PASS"*$] \geq 1-\beta$.

(2) $P(x) \not\approx_{\Delta_2} f(x) \Rightarrow \Pr[T^P$ *outputs "FAIL"*$] \geq 1-\beta$.

DEFINITION 47. *A* $(\Delta_1,\Delta_2,\epsilon,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate self-tester *for* f *is a probabilistic oracle program* T *that, given* P *and* β, *satisfies the following:*

(1) P $(\Delta_1,0)$-*approximates* f *on* $\mathcal{D}_{\tau(n,s)} \Rightarrow \Pr[T^P$ *outputs "PASS"*$] \geq 1-\beta$.

(2) P *does not* (Δ_2,ϵ)-*approximate* f *on* $\mathcal{D}_{n,s} \Rightarrow \Pr[T^P$ *outputs "FAIL"*$] \geq 1-\beta$.

Observe that if a property is $(\delta,\epsilon,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s},\Delta_1,\Delta_2,\Delta_3)$-approximately robust, $(\mathcal{D}_{n,s},\mathcal{D}_{n',s'},\Delta_2,\Delta_4)$-stable, and it is possible to do equality testing for the function family satisfying the property, then it is possible to construct a $(\Delta_1,\Delta_3+\Delta_4,\epsilon,\mathcal{D}_{n,s},\mathcal{D}_{n',s'})$-approximate self-tester.

DEFINITION 48. *A* $(\Delta,\epsilon,\Delta',\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate self-corrector *for* f *is a probabilistic oracle program* SC_f^P *that, given* P *that* (Δ,ϵ)-*approximates* f *on* $\mathcal{D}_{\tau(n,s)}$, $x \in \mathcal{D}_{n,s}$, *and* β, *outputs* $\mathrm{SC}_f^P(x)$ *such that* $\Pr[\mathrm{SC}_f^P(x) \approx_{\Delta'} f(x)] \geq 1-\beta$.

Note that a $(\Delta_1,\Delta_2,\epsilon,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate self-tester and $(\Delta_2,\epsilon,\Delta_3,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate self-corrector yield a $(\Delta_1,\Delta_3,\mathcal{D}_{\tau(n,s)},\mathcal{D}_{n,s})$-approximate result

checker [8].

5.2. Constructing approximate self-correctors. We illustrate how to build approximate self-correctors for functional equations. Suppose P (Δ, ϵ)-approximates f for $\epsilon < 1/8$ and $f(x+y) = G[f(x), f(y)]$. Then the self-corrector SC_f^P at input x is constructed as follows. To obtain a confidence of β,

1. choose random points $y_1, y_2, \ldots, y_N (N = O(\ln 1/\beta))$;
2. let $\mathrm{SC}_f^P(x)$ be the median of $G[P(x-y_1), P(y_1)], \ldots, G[P(x-y_N), P(y_N)]$.

By the assumption on ϵ, both the calls to P on $x - y_i$ and y_i are within Δ of f with probability greater than 3/4. In this case, the value of $G[P(x-y_i), P(y_i)]$ is $\Delta' = 2\Delta\widehat{G}$ away from $f(x)$ (see section 4.1 for $\widehat{G}$). Using Chernoff bounds, we can see that at least half of the values $G[P(x-y_i), P(y_i)]$ are at most Δ' away from $f(x)$. Thus, their median $\mathrm{SC}_f^P(x)$ is also at most Δ' away from $f(x)$.

For degree d polynomials, a similar self-corrector works with $\Delta' = O((d+1)2^d\Delta)$. In order to pass good programs, this is almost the best Δ' possible using the evenly spaced interpolation equation since the coefficients of the interpolation equation are $\Omega(2^d)$. Using interpolation equations that do not use evenly spaced points seems to require Δ' that is dependent on the size of the domain.

5.3. Constructing approximate self-testers. The following is a self-tester for any function satisfying an addition theorem $f(x+y) = G[f(x), f(y)]$ computing the function family $\mathcal{F}$ over $\mathcal{D}_{n,s}$. We use the notation from section 4.1. To obtain a confidence of β, we choose random points $x_1, y_1, \ldots, x_N, y_N (N = O(1/\epsilon \ln 1/\beta))$ and verify the assumptions on program P in the beginning of section 4.3. If P passes the test, then using Chernoff bounds, approximate robustness, and stability of the property, we are guaranteed that P approximates some function in $\mathcal{F}$. We next perform the equality test to ensure that P approximates the given $f \in \mathcal{F}$. Assume that $f(\frac{1}{s})$ when $c < 1$ (resp., $f(\frac{n}{s})$ when $c > 1$) is given. Using the proofs in section 4.2, one can show that if there is a constant Δ such that $\mathrm{SC}_f^P(\frac{1}{s}) \approx_\Delta f(\frac{1}{s})$ when $c < 1$ ($\mathrm{SC}_f^P(\frac{n}{s}) \approx_\Delta f(\frac{n}{s})$ when $c > 1$), the error between SC_f^P and f can be bounded by a constant on the rest of $\mathcal{D}_{n,s}$. Since SC_f^P approximates P, the correctness of the self-tester follows.

For polynomials, we use random sampling to verify the conditions on program P required for approximate robustness that are given in the beginning of section 3.3. If P satisfies the conditions, then using the approximate robustness and stability of the evenly spaced interpolation equation, P is guaranteed to approximate some degree d polynomial h. To perform the equality test that determines if P approximates the correct polynomial f, we assume that the tester is given the correct value of the polynomial f at $\ell = (d+1)/\epsilon$ evenly spaced points $x_1 = -\frac{n}{s}, \ldots, x_\ell = \frac{n}{s} \in \mathcal{D}_{n,s}$. Using the self-corrector SC_f^P from section 5.2, we have $\|\mathrm{SC}_f^P - h\| \le \Delta' = (d+1)2^d 2^{d \lg d}\Delta$. The equality tester now tests that for all x_i, $|f(x_i) - \mathrm{SC}_f^P(x_i)| \le (d+1)2^d\Delta$. Call an input x *bad* if $|f(x) - h(x)| > \Delta'' = \Delta' + (d+1)2^d\Delta$. If x is bad, then $|f(x) - \mathrm{SC}_f^P(x)| > (d+1)2^d\Delta$. If x is a sample point, and x is bad, then the test would have failed. Define a *bad interval* to be a sequence of consecutive bad points. If the test passes, then any bad interval in the domain can be of length at most $(2n+1)/\ell$, because any longer interval would contain at least one sample point. The two sample points immediately preceding and following the bad interval satisfy $|f(x) - h(x)| \le \Delta''$. This implies that there must be a local maximum of $f - h$ (a degree d polynomial) inside the bad interval. Since there are only d extrema of $f - h$, there can be at most d bad

intervals, and so the total number of bad points is at most $d(2n+1)/\ell$. Thus, on $1-\epsilon$ fraction of $\mathcal{D}_{n,s}$, SC_f^P's error is at most $\Delta' + \Delta''$. These arguments can be generalized to the k-variate case by partitioning the k-dimensional space into $((d+1)/\epsilon)^k$ cubes.

We have thus shown how to construct approximate self-testers and self-correctors. It is straightforward to construct approximate result-checkers using these.

5.4. Reductions between functional equations. This section explores the idea of using reductions among functions (as in [7, 3]) to obtain approximate self-testers for new functions. Consider any pair of functions f_1, f_2 that are interreducible via functional equations. Suppose we have an approximate self-tester for f_1 and let there exist continuous computable functions F, F^{-1} such that $f_2(x) = F[f_1(x)]$ and $f_1(x) = F^{-1}(f_2(x))$. Given a program P_2 computing f_2, construct program P_1 computing f_1 via F^{-1}. We can then self-test P_1. Suppose P_1 is Δ-close to f_1 on a large portion of the domain. Then for every x for which $P_1(x)$ is Δ-close to $f_1(x)$, we bound the deviation of $P_2(x)$ from $f_2(x)$ by $\Delta' = F[f_1(x) + \Delta] - f_2(x)$. Then $\Delta' = F[f_1(x) + \Delta] - F[f_1(x)] \leq \omega(F; \Delta)$. If we can bound the right-hand side by a constant (at least for a portion of the domain), we can bound the maximum deviation Δ' of P_2 from f_2. This idea can be used to give simple and alternative approximate self-testers for functions like $\sin x, \cos x, \sinh x, \cosh x$ which can be reduced to e^x.

For example, suppose we are given a $(\delta_1, \epsilon_1, \delta_2, \epsilon_2, \mathcal{D}, \mathcal{D}')$-approximate self-tester for $f_1(x) = e^x$ and we want an approximate self-tester for the function f_2 given by $f_2(x) = \cos x$. By the Euler identity, $f_1(ix) = f_2(x) + i f_2(x + 3\pi/2)$. Given a program P_2 that supposedly computes f_2, we can build a program P_1 (for e^{ix}) out of the given P_2 (for $\cos x$) and self-test P_1. $P_1(ix) = P_2(x) + iP_2(x + 3\pi/2)$.

Let the range of f_1 be equipped with the following metric: $|P_1(x) - f_1(x)| = |\Re(P_1(x) - f_1(x))| + |\Im(P_1(x) - f_1(x))|$. In other words, in our case, we have $|P_1(x) - e^{ix}| = |P_2(x) - \cos x| + |P_2(x + 3\pi/2) - \cos(x + 3\pi/2)|$. This metric ensures that P_1 is erroneous on x if and only if P_2 is erroneous on at least one of $x, x + 3\pi/2$. Alternatively, there is no "cancellation" of errors.

Suppose P_1 is (δ_1, ϵ_1)-good. Then, what can we say about P_2? For δ_1 fraction of the "bad" domain for P_1, the errors can occur in both the places where P_2 is invoked. Hence, at most $2\delta_1$ fraction of the domain for P_2 is bad. The rest of the domain for P_1 is ϵ_1-close to f_1, which by our metric implies P_2 is also ϵ_1-close to f_2. Thus, P_2 is $(2\delta_1, \epsilon_1)$-good.

Similarly, suppose P_1 is not (δ_2, ϵ_2)-good. P_1 is not good on at least δ_2 fraction of the domain, where P_1 is not ϵ_2-close to f_1. Thus, at these points in the domain, at least one of the points where P_2 is called is definitely not $\epsilon_2/2$-close to f_2. Thus, P_2 is not $(\delta_1, \epsilon_2/2)$-good.

Therefore, we have a $(2\delta_1, \epsilon_1, \delta_2, \epsilon_2/2, \mathcal{D}, \mathcal{D}')$-approximate self-tester for f_2 from a $(\delta_1, \epsilon_1, \delta_2, \epsilon_2, \mathcal{D}, \mathcal{D}')$-approximate self-tester for f_1, given by [20].

Appendix A. Proofs of some theorems for linearity.

THEOREM 49 (Hyers' theorem). *Let S be an Abelian semigroup and B be a Banach space and let $g : S \to B$ be such that for some $\Delta > 0$, g is Δ-approximately linear on S; then, for every $x \in S$, $h(x) = \lim_{n\to\infty} g(2^n x)/2^n$ exists, h is linear, and $\|g - h\| \leq \Delta$.*

Proof (see [17]). By induction on n, $|g(x)/2^n - g(x/2^n)| < \Delta(1 - 1/2^n)$. Let $q_n(x) = g(2^n x)/2^n$. Then, $q_n(x) - q_m(x) = (g(2^{m-n}2^n x) - 2^{m-n} g(2^n x))/2^m$. If $m < n$, we can obtain $|q_n(x) - q_m(x)| < \Delta(1 - 2^{m-n})/2^m$. Thus, for $x \in S$, $\{q_n(x)\}$ is a Cauchy sequence and by completeness of B, it has a limit function

$h(x) = \lim_{n\to\infty} g(2^n x)/2^n$. The properties of h are easily proved. ∎

THEOREM 50 (Skof's theorem). *Let $n > 0$ and let $g : [0, n) \to \mathbb{R}$ be such that for some $\Delta \geq 0$, $|g(x+y) - g(x) - g(y)| \leq \Delta$ for all $0 \leq x, y < n$ (such that $x + y < n$); then, there exists a linear $h : \mathbb{R} \to \mathbb{R}$ such that $\|g - h\|_{[0,n)} \leq 3\Delta$.*

Proof. For any $x \in \mathbb{R}^+$, write $x = p(n/2) + q$ where $0 \leq q < n/2$. Define $g' : \mathbb{R}^+ \to \mathbb{R}$ such that $g'(x) = pg(n/2) + g(q)$. Clearly, $\|g' - g\|_{[0,n)} \leq \Delta$. Now, the claim is $g'(x+y) \approx_{2\Delta} g'(x) + g'(y)$. As before, $x = p(n/2) + q, y = r(n/2) + s$ with $0 \leq q, s < n/2$.

If $0 \leq q + s < n/2]$, then we have $g'(x+y) = g(q+s) + (p+r)g(n/2) \approx_\Delta g(q) + g(s) + pg(n/2) + rg(n/2) = g'(x) + g'(y)$.

If $n/2 \leq q + s < n]$, then let $q + s = t + n/2$. We have $g'(x+y) = g(t) + (p + r)g(n/2) + g(n/2) \approx_{2\Delta} g(q) + g(s) + pg(n/2) + rg(n/2) = g'(x) + g'(y)$.

To extend g' to $\mathbb{R}$, define for $x < 0$, $g'(x) = -g'(-x)$. Thus, for all $x, y \in \mathbb{R}, g'(x+y) \approx_{2\Delta} g'(x) + g'(y)$. By Theorem 49, there is a linear h such that $\|g' - h\| \leq 2\Delta$. Therefore, $\|g - h\|_{[0,n)} \leq \|g - g'\|_{[0,n)} + \|g' - h\|_{[0,n)} \leq 3\Delta$. ∎

Appendix B. Proofs of some theorems for polynomials.

B.1. Stability for polynomials.

FACT 12. $(\nabla_{t_1+t_2} - \nabla_{t_1} - \nabla_{t_2})f(x) = \nabla_{t_1,t_2} = \nabla_{t_2,t_1}$.

Proof. $(\nabla_{t_1+t_2} - \nabla_{t_1} - \nabla_{t_2})f(x) = f(x+t_1+t_2) - f(x) - f(x+t_1) + f(x) - f(x+t_2) + f(x) = f(x+t_1+t_2) - f(x+t_1) - f(x+t_2) + f(x) = \nabla_{t_1} f(x+t_2) - \nabla_{t_1} f(x) = \nabla_{t_1}(f(x+t_2) - f(x)) = \nabla_{t_1,t_2} f(x) = \nabla_{t_2,t_1} f(x)$. ∎

Difference operators act on multilinear functions in a nice manner, which is captured in the following fact.

FACT 51. *If f is a k-linear function, then $\nabla_{t_1,\ldots,t_d} f^*(x) = k! f(t_1, \ldots, t_k)$ if $k = d$ and 0 if $k < d$.*

Proof. Recall that, due to multilinearity, f is also symmetric. By induction on k and chasing definitions, we have $\nabla_{t_1,\ldots,t_d} f^*(x) = \nabla_{t_1,\ldots,t_{d-1}}(f^*(x+t_d) - f^*(x)) = \nabla_{t_1,\ldots,t_{d-1}}(f((x+t_d)^{[d-1]}, x) + f((x+t_d)^{[d-1]}, t_d) - f(x^{[d]}))$, which by linearity of ∇ yields $\nabla_{t_1,\ldots,t_{d-1}} f((x+t_d)^{[d-1]}, t_d) + \nabla_{t_1,\ldots,t_{d-1}}(f((x+t_d)^{[d-1]}, x) - f(x^{[d]}))$. Observe that for any constant t, the restriction of k-linear f to any of its arguments being t (denoted f_t) results in a $(k-1)$-linear function. By induction, the first term in the above expression evaluates to $(d-1)! f_{t_d}(t_1, \ldots, t_{d-1}) = (d-1)! f(t_1, \ldots, t_d)$. Now, using the symmetry and linearity (in each variable) of f, we can write the second term as $\nabla_{t_1,\ldots,t_{d-1}}(\sum_{i=0}^{d-1} \binom{d-1}{i} f(x^{[i+1]}, {t_d}^{[d-i-1]}) - f(x^{[d]}))$ which is $(d-1)\nabla_{t_1,\ldots,t_{d-1}} f(x^{[d-1]}, t_d) + \sum_{i=0}^{d-3} \nabla_{t_1,\ldots,t_{d-1}} \binom{d-1}{i} f(x^{[i+1]}, {t_d}^{[d-i-1]})$. By induction, the first term evaluates to $(d-1)(d-1)! f_{t_d}(t_1, \ldots, t_{d-1}) = (d-1)(d-1)! f(t_1, \ldots, t_d)$, which combined with the earlier result yields $d! f(t_1, \ldots, t_d)$. The second term evaluates to 0 since each of the terms inside the sum are restrictions of f to more than 1 variable, which evaluates to 0 after applying $\nabla_{t_1,\ldots,t_{d-1}}$. ∎

FACT 13. *Let $\mathcal{D}$ be a ring. The following characterizations of polynomials, are equivalent:*

1. *for all $x \in \mathcal{D}, f(x) = \sum_{k=0}^{d} a_k x^k$;*
2. *for all $x, t \in \mathcal{D}, \nabla_t^{d+1} f(x) = 0$;*
3. *there exists symmetric k-linear functions F_k, $0 \leq k \leq d$ such that for all $x \in \mathcal{D}, f(x) = \sum_{k=0}^{d} F_k^*(x)$.*

Proof. (1) ⇔ (2) follows from Lagrangian interpolation. We first prove (1) ⇒ (3). Given (1), just set $F_k(x_1, \ldots, x_k) = a_k \prod_{i=0}^{k} x_i$. It is easy to see that F_k's are symmetric, k-linear. We now prove (3) ⇒ (2). Given (3), $\nabla_{t_1,\ldots,t_{d+1}} f(x) =$

TABLE 2
An illustration of Fact 18.

λ	t'_λ	t''_λ	Term
00	0	0	0
01	$-t_2/2$	t_2	$-[f(x) - 2f(x + t_2/2) + f(x + t_2)]$
10	$-t_1$	t_1	$-[f(x - t_1) - 2f(x) + f(x + t_1)]$
11	$-t_1 - t_2/2$	$t_1 + t_2$	$+[f(x - t_1) - 2f(x + t_2/2) + f(x + t_1 + t_2)]$

$\nabla_{t_1,\ldots,t_{d+1}} \sum_{k=0}^{d} F_k^*(x) = \sum_{k=0}^{d} \nabla_{t_1,\ldots,t_{d+1}} F_k^*(x) = 0$ by Fact 16 about difference operators. □

FACT 18. *For any* $\lambda_1, \ldots, \lambda_d \in \{0,1\}$, *if*

$$t'_{\lambda_1,\ldots,\lambda_d} = -\sum_{i=1}^{d} \lambda_i t_i / i, \qquad t''_{\lambda_1,\ldots,\lambda_d} = \sum_{i=1}^{d} \lambda_i t_i,$$

then

$$\nabla_{t_1,\ldots,t_d} f(x) = \sum_{\lambda_1,\ldots,\lambda_d \in \{0,1\}} (-1)^{\lambda_1+\cdots+\lambda_d} \nabla^d_{t'_{\lambda_1,\ldots,\lambda_d}} f(x + t''_{\lambda_1,\ldots,\lambda_d}).$$

Proof. The proof follows by a pairing argument. First, it is easy to prove that the left-hand side can be expressed as $\nabla_{h_1,\ldots,h_d} = \sum_{\lambda_1,\ldots,\lambda_d \in \{0,1\}} (-1)^{d+\lambda_1+\cdots+\lambda_d} f(x + t''_{\lambda_1,\ldots,\lambda_d})$. Now, we can expand the right-hand side as

$$\sum_{\lambda_1,\ldots,\lambda_d \in \{0,1\}} (-1)^{\lambda_1+\cdots+\lambda_d} \sum_{k=0}^{d} (-1)^{d-k} \binom{d}{k} f(x + t''_{\lambda_1,\ldots,\lambda_d} + k t'_{\lambda_1,\ldots,\lambda_d}).$$

When $k = 0$, the left-hand side is obtained. Therefore, we have to prove that for $k > 0$, the right-hand side vanishes. The terms inside $f(\cdot)$ are linear combinations of t_i's by our construction. Note that for each $k > 0$, each term inside $f(\cdot)$ on the right-hand side has exactly one t_i absent because of its cancellation between t' and t''. Therefore, for each $\lambda_1, \ldots, \lambda_d \in \{0,1\}$, construct its conjugate $\lambda'_1, \ldots, \lambda'_d \in \{0,1\}$ with $\lambda'_i = 1 - \lambda_i$ and $\lambda'_j = \lambda_j$ otherwise. It is easy to see that the terms $(-1)^{\lambda_1+\cdots+\lambda_d} f(x + t''_{\lambda_1,\ldots,\lambda_d} + k t'_{\lambda_1,\ldots,\lambda_d})$ and $(-1)^{\lambda'_1+\cdots+\lambda'_d} f(x + t''_{\lambda'_1,\ldots,\lambda'_d} + k t'_{\lambda'_1,\ldots,\lambda'_d})$ cancel. An illustration of this fact is given below. □

To illustrate with an example, consider the case when $d = 2$. Then, the left-hand side is given by $\nabla_{t_1,t_2} f(x) = f(x + t_1 + t_2) - f(x + t_1) - f(x + t_2) + f(x)$. The right-hand side is given by the sum of the entries in the last column of Table 2.

It is easy to see that appropriate cancellations take place so that the left-hand side equals the right-hand side.

Acknowledgments. We would like to thank Janos Aczel (U. of Waterloo), Peter Borwein (Simon Fraser U.), Gian Luigi Forti (U. of Milan), D. Sivakumar (SUNY, Buffalo), Madhu Sudan (IBM, Yorktown), and Nick Trefethen (Cornell U.), for their suggestions and pointers. We would also like to thank the two anonymous referees for improving the presentation of this paper.

REFERENCES

[1] J. Aczel, *Lectures on Functional Equations and Their Applications*, Academic Press, New York, 1966.

[2] M. Albert and J.A. Baker, *Functions with bounded n-th differences*, Ann. Polon. Math., 43 (1983), pp. 93–103.

[3] S. Ar, M. Blum, B. Codenotti, and P. Gemmell, *Checking approximate computations over the reals*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 786–795.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[5] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[6] L. Babai, L. Fortnow, and C. Lund, *Nondeterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[7] M. Blum and S. Kannan, *Designing programs that check their work*, J. ACM, 42 (1995), pp. 269–291.

[8] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[9] M. Blum and H. Wasserman, *Software reliability via run-time result-checking*, J. ACM, 44 (1997), pp. 826–849.

[10] M. Blum and H. Wasserman, *Reflections on the Pentium division bug*, IEEE Trans. Comput., 45 (1996), pp. 385–393.

[11] E. Castillo and M.R. Ruiz-Cobo, *Functional Equations and Modeling in Science and Engineering*, Marcel Dekker, New York, 1992.

[12] P.W. Cholewa, *The stability problem for a generalized Cauchy type functional equation*, Rev. Roumaine Math. Pures Appl., 29 (1984), pp. 457–460.

[13] W.J. Cody, *Performance evaluation of programs related to the real gamma function*, ACM Trans. Math. Software, 17 (1991), pp. 46–54.

[14] W.J. Cody and I. Stoltz, *The use of Taylor series to test accuracy of function programs*, ACM Trans. Math. Software, 17 (1991), pp. 55–63.

[15] D.Z. Djokovic, *A representation theorem for* $(X_1-1)(X_2-1)\dots(X_n-1)$ *and its applications*, Ann. Polon. Math., 22 (1969), pp. 189–198.

[16] G.L. Forti, *An existence and stability theorem for a class of functional equations*, Stochastica, 4 (1980), pp. 23–30.

[17] G.L. Forti, *Hyers–Ulam stability of functional equations in several variables*, Aequationes Math., 50 (1995), pp. 143–190.

[18] G.L. Forti and J. Schwaiger, *Stability of homomorphisms and completeness*, C.R. Math. Rep. Acad. Sci. Canada, 11 (1989), pp. 215–220.

[19] Z. Gajda, *Local stability of the functional equation characterizing polynomial functions*, Ann. Polon. Math., 52 (1990), pp. 119–137.

[20] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991, pp. 32–42.

[21] D.H. Hyers, *On the stability of the linear functional equation*, Proc. Natl. Acad. Sci. USA, 27 (1941), pp. 222–224.

[22] M. Kiwi, F. Magniez, and M. Santha, *Approximate testing with relative error*, in Proceedings of the 31st ACM Symposium on Theory of Computing, 1999, pp. 51–60.

[23] S.R. Kumar and D. Sivakumar, *On self-testing without the generator bottleneck*, in Proceedings of the 15th Annual Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1026, Springer-Verlag, Berlin, 1995, pp. 248–262.

[24] R. Lipton, *New directions in testing*, in Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography, 1991, pp. 191–202.

[25] G.G. Lorentz, *Approximation of Functions*, Holt, Rinehart and Winston, New York, 1966.

[26] C. Lund, *The Power of Interaction*, Ph.D. Thesis, University of Chicago, Chicago, IL, 1991.

[27] S. Mazur and W. Orlicz, *Grundlegende Eigenschaften der Polynomischen Operationen,* Studia Math., 5 (1934), pp. 50–68.

[28] R. Rubinfeld, *On the robustness of functional equations*, SIAM J. Comput., 28 (1999), pp. 1972–1997.

[29] R. Rubinfeld and M. Sudan, *Self-testing polynomial functions efficiently and over rational domains*, in Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algo-

rithms, Orlando, FL, 1992, pp. 23–43.
[30] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials and their applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[31] F. Skof, *Sull'approssimazione delle applicazioni localmente δ-additive*, Atti della Accademia delle Scienze di Torino, 117 (1983), pp. 377–389.
[32] M. Sudan, *private communication*, University of California, Berkeley, 1991.
[33] A.F. Timan, *Theory of Approximation of Functions of a Real Variable*, Pergamon Press, New York, 1963.
[34] E.C. Titchmarsh, *The Theory of Functions*, Oxford University Press, 1947.
[35] F. Vainstein, *Algebraic Methods in Hardware/Software Testing*, Ph.D. Thesis, Boston University, Boston, MA, 1993.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 577–600

THE POLYGON EXPLORATION PROBLEM*

FRANK HOFFMANN†, CHRISTIAN ICKING‡, ROLF KLEIN§, AND KLAUS KRIEGEL†

Abstract. We present an on-line strategy that enables a mobile robot with vision to explore an unknown simple polygon. We prove that the resulting tour is less than 26.5 times as long as the shortest watchman tour that could be computed off-line.

Our analysis is doubly founded on a novel geometric structure called the *angle hull*. Let D be a connected region inside a simple polygon, P. We define the *angle hull* of D, $\mathcal{AH}(D)$, to be the set of all points in P that can see two points of D at a right angle. We show that the perimeter of $\mathcal{AH}(D)$ cannot exceed in length the perimeter of D by more than a factor of 2. This upper bound is tight.

Key words. angle hull, competitive strategy, computational geometry, curve length, motion planning, navigation, on-line algorithm, optimum watchman tour, polygon, robot

AMS subject classifications. 68U05, 68U30

PII. S0097539799348670

1. Introduction.

In the last decade, the path planning problem of autonomous mobile systems has received a lot of attention in the communities of robotics, computational geometry, and on-line algorithms; see, e.g., Rao et al. [23], Blum, Raghavan, and Schieber [5], and the surveys by Mitchell [21] in Sack and Urrutia [25] and by Berman [4] in Fiat and Woeginger [13]. We are interested in strategies that are correct, in that the robot will accomplish its mission whenever this is possible, and in performance guarantees that allow us to relate the robot's cost to the cost of an optimal off-line solution or to other complexity measures of the scene.

In this work we are addressing a basic problem in this area. Suppose a mobile robot has to explore an unknown environment modeled by a simple polygon. The robot starts from a given point, s, on the polygon's boundary. It is equipped with a vision system that continuously provides the visibility of the robot's current position. When each point of the polygon has at least once been visible, the robot returns to s.[1]

In the on-line polygon exploration problem we ask for a *competitive* exploration strategy that guarantees that the robot's path will never exceed in length a constant *competitive factor* times the length of the optimum watchman tour through s, i. e., of the shortest tour inside the polygon that contains s and has the property that each point of the polygon is visible from some point of the tour. This approach to evaluating the performance of an on-line strategy goes back to Sleator and Tarjan [27]. A priori it is not clear whether a competitive exploration strategy exists.

*Received by the editors June 10, 1999; accepted for publication (in revised form) November 6, 2000; published electronically October 11, 2001. This work was supported by the Deutsche Forschungsgemeinschaft, grant Kl 655/8-3. This is the final version of conference papers that have appeared in the proceedings of SODA '97 and WAFR '98 (section 2; see [15, 16]) and SWAT '98 (section 3; see [17]).

http://www.siam.org/journals/sicomp/31-2/34867.html

†Freie Universität Berlin, Institut für Informatik, Takustrasse 9, D-14195 Berlin, Germany (hoffmann@inf.fu-berlin.de, kriegel@inf.fu-berlin.de).

‡FernUniversität Hagen, Praktische Informatik VI, Feithstrasse 142, D-58084 Hagen, Germany (icking@feu.de).

§Universität Bonn, Institut für Informatik I, Römerstrasse 164, D-53117 Bonn, Germany (rolf.klein@uni-bonn.de).

[1] In the absence of holes, the robot has seen each point inside the polygon as soon as it has seen each point on its boundary.

Even the *off-line* version of the polygon exploration problem is not easy. Here we are given a simple polygon and have to compute the optimum watchman tour through a specified boundary point, s. Initially this problem has been suspected to be NP-hard. Chin and Ntafos [9] were the first to provide a polynomial time solution. They have shown how to compute the optimum watchman tour in time $O(n^4)$, where n denotes the number of vertices of the polygon. Later, their result has been improved by Tan and Hirata [28]; see also the updates in [14, 29].

Carlsson, Jonsson, and Nilsson [7] have proven that the optimum watchman tour without a specified point s can be computed in time $O(n^3)$. Furthermore, Carlsson and Jonsson [6] proposed an $O(n^6)$ algorithm for computing the shortest path inside a simple polygon from which each point of the boundary is visible when start and end points are not specified. In these papers it is always assumed that the range of the robot's visibility is unbounded. Some authors have also studied the case of limited visibility, e.g., Arkin, Fekete, and Mitchell [3] and Ntafos [22].

As to the *on-line* version of the polygon exploration problem, Deng, Kameda, and Papadimitriou [11] were the first to claim that a competitive strategy does exist. In their seminal paper they discussed a factor of 2016 for a greedy off-line approach which has to be implemented as an on-line strategy. For the rectilinear case, they gave a complete and elegant proof in [12]; here the greedy strategy can be applied that performs surprisingly well.

The first proof for the more difficult case of nonrectilinear simple polygons has been given in our conference paper [15]. There we have provided an on-line exploration strategy and sketched a proof that the tour it generates in any polygon is not longer than 133 times the length of the optimum watchman tour. One of the main difficulties with this analysis was in establishing reasonably sharp length estimates for robot paths of complex structure and in relating them to the optimum watchman tour.

The present paper contains the first complete presentation and analysis of an exploration strategy for simple polygons. As compared to the conference version [15], this full paper has been greatly simplified and describes a new analysis that is built on an interesting geometric relation between the robot's path and the optimum watchman tour. This relation is expressed in terms of the *angle hull*, a novel geometric structure. With these improvements we are able to show that an unknown polygon can be explored, from a given boundary point, s, by a tour at most 26.5 times as long as the shortest watchman tour containing s.

Of course, there is still a considerable gap between this upper bound and the lower bound of $(1+\sqrt{2})/2$ given in [11]; however, experiments with our new strategy suggest that its actual performance is much better than the bound proven here; we conjecture a number far below 10.

The organization of this paper is as follows. Section 2 contains a hierarchical description of the strategy and of its analysis. In section 2.1 we first discuss how to explore a single corner, that is, a single reflex vertex one of whose adjacent edges has not yet been visible. By *reflex vertex* we denote a vertex of the polygon whose internal angle exceeds 180°; all other vertices are called *convex*. The robot explores these corners in a sophisticated order: Of all reflex vertices that touch the visible area from the right, the robot attempts to explore the one that is clockwise first on the polygon's boundary as seen from the starting point, s. However, the vertex hereby specified may change as the robot moves. From our angle hull result we obtain a bound on the length of the resulting path in terms of the length of the shortest path that leads to the final position.

Then, in section 2.2, we use this technique for efficiently exploring groups of right reflex vertices in clockwise order. The length of the resulting local tour is shown to be bounded by the perimeter of the relative convex hull of certain *base points* times a constant. In analogy to the standard definition of convex hulls, the *relative convex hull*, $\mathcal{RCH}(D)$, of a subset D of a polygon P is the smallest subset of P that contains D and, for any two points of D, the shortest path in P connecting them.

In section 2.3 we show how the robot recursively detects and explores an exhaustive system of groups of right and left reflex vertices. Groups of vertices that are on sufficiently different recursive levels give rise to base point sets whose $\mathcal{RCH}$s are mutually invisible. Therefore, the sum of their hulls' perimeters is less than the perimeter of the $\mathcal{RCH}$ of their union. In Lemma 2.6 we will show that each base point can see two points of the optimum watchman tour, W_{opt}, at an angle of 90°; therefore, all base points must be contained in the angle hull of W_{opt}. Consequently, the perimeter of the $\mathcal{RCH}$ of the base points must be less than the perimeter of the $\mathcal{RCH}$ of the angle hull of W_{opt}. The latter, in turn, can only be less than or equal to the perimeter of the $\mathcal{RCH}$ of W_{opt} itself, because the perimeter of the $\mathcal{RCH}$ of a connected object is no longer than the perimeter of the object itself. Now we can apply our angle hull result a second time in estimating the perimeter of the angle hull of W_{opt} against the length of W_{opt} itself. This way we have bounded the length of the whole exploration path walked by the robot by a multiple of the length of the optimum watchman path, W_{opt}.

Our analysis greatly benefits from this new geometric structure we propose to call the *angle hull*. Let D be a simple polygon contained in another simple polygon, P. Then the angle hull, $\mathcal{AH}(D)$, of D consists of all points in P that can see two points of D at an angle of 90°. The boundary of $\mathcal{AH}(D)$ can be described as the path of a diligent photographer who uses a 90° angle lens and wants to take a picture of D that shows as large a portion of D as possible but no walls of P. Before taking the picture, the photographer walks around D, in order to inspect all possible viewpoints.

In section 3 we prove that the photographer's path is at most twice as long as the perimeter of her model, D.

2. The strategy and its analysis. Let P be a simple polygon and let s be a point on its boundary. The *shortest path tree* of s consists of all shortest paths from s to the vertices of P. Its internal nodes are reflex vertices of P. Those vertices touching a shortest path from the right are called *right reflex vertices*; left reflex vertices are defined accordingly. If we follow the shortest path from s to reflex vertex v, one of its adjacent polygon edges remains invisible until v is actually reached. The extension into the polygon of this invisible edge is called a *cut* of P with respect to s.

Exploring a polygon P is equivalent to visiting all of its cuts with respect to the start point s.

Figure 2.1 shows an example of the *optimum watchman tour*, W_{opt}, containing a boundary point, s. Tan and Hirata [28] have provided an off-line algorithm for computing W_{opt} within time $O(n^2)$ for a polygon of n edges.

We say a vertex has been *discovered* after it has been visible at least once from the robot's current position. A reflex vertex is *unexplored* as long as its cut has not been reached, and *fully explored* thereafter.

In an unknown polygon, even exploring a single reflex vertex requires a little care. For example, one cannot afford to go straight to the vertex in order to get to its cut: The cut could be passing by the start point very closely, so that a much shorter path would be optimal.

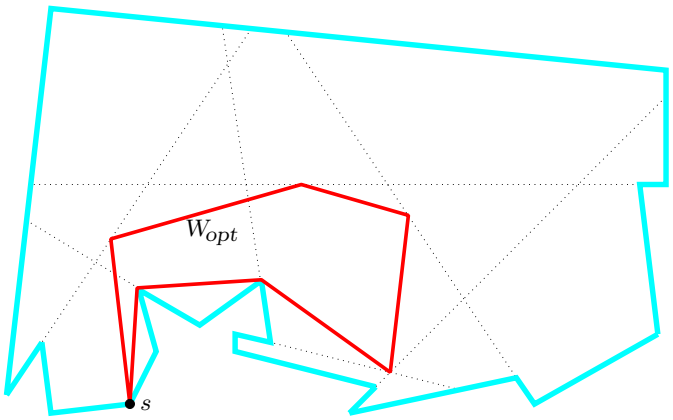


FIG. 2.1. *The optimum watchman tour visits all cuts of the polygon.*

We avoid this difficulty as follows. Whenever the robot wants to explore a right reflex vertex, r, visible from some local start point, p, it approaches r along the clockwise oriented circle *spanned* by p and by r, denoted by $\mathit{circ}\,(p, r)$, i.e., the smallest circle that contains p and r.

Consequently, when the robot reaches the cut of r at some point c, the ratio of the length of the circular arc from p to c over their euclidean distance is bounded by $\frac{\pi}{2} \approx 1.57$.

One might wonder if the subproblem of exploring a single vertex can be solved more efficiently by using curves other than circular arcs. This is, in fact, the case; Icking, Klein, and Ma [20] have shown that an optimum ratio of ≈ 1.212 is achieved by curves that result from solving certain differential equations. However, these curves are lacking a useful property possessed by circular arcs: The intersection point, c, of the circular arc with the cut is just the point on the cut closest to p, due to Thales's theorem.[2] This property turns out to be very helpful in our analysis.

In rectilinear polygons, the cut of each visible reflex vertex is known, and two cuts can cross only perpendicularly. This makes it possible to apply a simple greedy exploration strategy: The robot always walks to the cut of the next reflex vertex, in clockwise order, one of whose edges is invisible; see Deng, Kameda, and Papadimitriou [12].

For general polygons, this greedy approach is bound to fail, as Figure 2.2 illustrates. The example polygon shown there suggests exploring left and right reflex vertices separately. However, it is not really obvious how to do this in general, since, e.g., the existence of a left reflex vertex at the end of a long chain of right vertices is initially not known to the robot. Therefore, it seems necessary to partition left and right reflex vertices into compact groups that can be explored one by one.

2.1. Exploring a single vertex. The essential subtask of the robot's strategy is in exploring a single vertex. This is handled by the following procedure, *ExploreRightVertex*. We first list the complete pseudocode of the procedure, then we explain its steps and notation in detail.

[2]Thales's theorem: an angle inscribed in a semicircle is always a right angle.

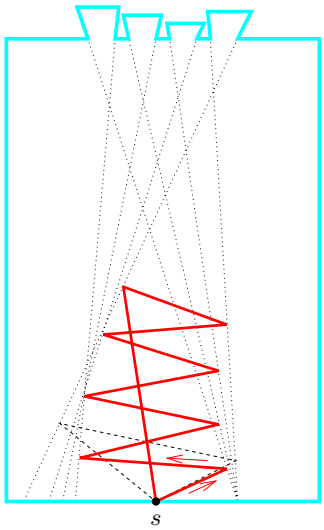


FIG. 2.2. *Visiting cuts in the order in which their vertices appear on the boundary does not lead to a competitive strategy.*

PROCEDURE *ExploreRightVertex* (**inout** *TargetList*, **inout** *ToDoList*);

BasePoint := *CP*;

Target := *First* (*TargetList*);

if *Target* not visible **then**
walk on shortest path from *BasePoint* to *Target*
until *Target* becomes visible;

Back := last vertex before *CP* on shortest path from *BasePoint* to *CP*;

walk clockwise along *circ* (*Back*, *Target*)
while maintaining *TargetList* and *ToDoList*

whenever *First* (*TargetList*) changes let *Target* := *First* (*TargetList*);
whenever *Back* becomes invisible update *Back*;

exceptions for walking along the circle:
if the boundary of P blocks the walk on the current circle **then**
walk clockwise along the boundary
until the circular walk is again possible;
if *Target* is becoming invisible **then**
walk towards *Target*
until the blocking vertex is reached;

until *Target* is fully explored;

end *ExploreRightVertex*;

We are using the abbreviation *CP* to denote the robot's current position. Procedure *ExploreRightVertex* works on two lists of vertices, *TargetList* and *ToDoList*. On entry, *TargetList* contains a list of right vertices, sorted in clockwise order along the boundary, that have already been discovered but not yet explored. When *ExploreRightVertex* is called for the very first time, *TargetList* contains exactly those right vertices that are visible from the start point, s, and have an invisible edge. *ToDo-*

List can be thought of as a long-term agenda that is passed to *ExploreRightVertex*; however, this procedure will only add to this list but not carry out one of the tasks.

In our pseudocode, *CP* (current position) is a global variable whose value can be changed only by **walk** statements. The robot's current position on calling *ExploreRightVertex* is called a *base point*.

The robot wants to explore the first vertex, *Target*, of *TargetList*. This vertex may have been discovered at an earlier stage, so that it may no longer be visible from the current position. In this case, the robot walks along the shortest path towards *Target* until it becomes visible again. Note that this shortest path is known to the robot; in fact a shortest path is always known between two points that have been seen (which would not be the case for polygons with holes).

Now the robot starts approaching *Target* along the circular arc spanned by the base point and by *Target*. On the way, a new right vertex, r, may be discovered. If one of its edges is invisible, r gets inserted into *TargetList*, provided that a certain criterion is met. Namely, the shortest path from the current *stage point*—a vertex defined one level up in the strategy—to r must not contain left turns. A right vertex that violates this criterion is ignored for now. A precise definition of a stage point is given in section 2.2.

It may happen that the vertex r newly discovered and inserted into *TargetList* comes before *Target* in clockwise order. In this case the robot ceases approaching its old target and starts exploring, from its current position, vertex r. This way, the vertex *Target* currently under exploration may repeatedly change.

It may also happen that the robot loses sight of the base point from which the current execution of procedure *ExploreRightVertex* has started. Namely, the robot's view of the base point may become obstructed by some left or right reflex vertex, b_1; examples will be shown in Figure 2.3. In this case, the exploration of the current *Target* no longer proceeds along the circle spanned by the base point and by *Target*; instead, it switches to the circle spanned by *Target* and by b_1. As the robot continues, its view of b_1 may become obstructed by some reflex vertex b_2, and so on. These reflex vertices, b_1, b_2,..., b_i define the shortest path from the base point to the robot's current position, and the robot explores *Target* along the circle spanned by *Target* and b_i. This last reflex vertex of the chain is named *Back* in the code. If the robot reaches the cut of *Target* at some point c, it arrives there at a right angle by the Thales property. Consequently, the shortest path in P from the base point to the cut goes through b_1, b_2,..., b_i and ends in c.

If the robot crosses the cut of a right vertex different from *Target* the former vertex is removed from *TargetList* because it has been explored on the way, this is done in the "while maintaining *TargetList*" step. Eventually, the target itself is deleted from the list when its cut has been reached.

When a right reflex vertex is explored, all of its children in the shortest path tree have already been discovered. Those right vertices having a left child are inserted into *ToDoList*, as candidates for future stages, together with references to their left children.

Finally, there are some exceptional events procedure *ExploreRightVertex* needs to take care of. If the robot's circular exploration path hits the boundary of the polygon, the robot follows the boundary until a circular path again becomes possible. If the robot's view of the target vertex is about to be blocked, the robot walks straight to the blocking vertex and continues from there on a circular path.

For ease of reference we summarize the rules by which *ExploreRightVertex* proceeds.

1. The current *target* vertex, i. e., the vertex whose cut we are intending to reach at the moment, is always the clockwise first among those right reflex vertices that have been discovered but not yet fully explored, i. e., the first element of *TargetList*. Only such right vertices can be in *TargetList* whose shortest paths from the stage point make only right turns.
2. To explore a right reflex vertex, r, we follow the clockwise oriented circle spanned by r and by the last vertex before CP on the shortest path from the base point to CP.
3. When the view to the current target vertex gets blocked (or when the boundary is hit) we walk straight towards the blocking vertex (or follow the boundary) until motion according to rule 2 becomes possible again.

Figure Figure 2.3 demonstrates how this strategy works. Initially, r_3 is the only right vertex visible; consequently, *TargetList* contains only r_3, and the robot's path begins with a circular arc spanned by s and by r_3. At point a, right vertex r_2 becomes visible. It is situated before r_3 on the boundary; therefore, the robot switches to exploring r_2, according to rule 1. Note that the circle spanned by s and by r_2 is passing through a, too, so that it is in fact possible to apply rule 2 at this point.

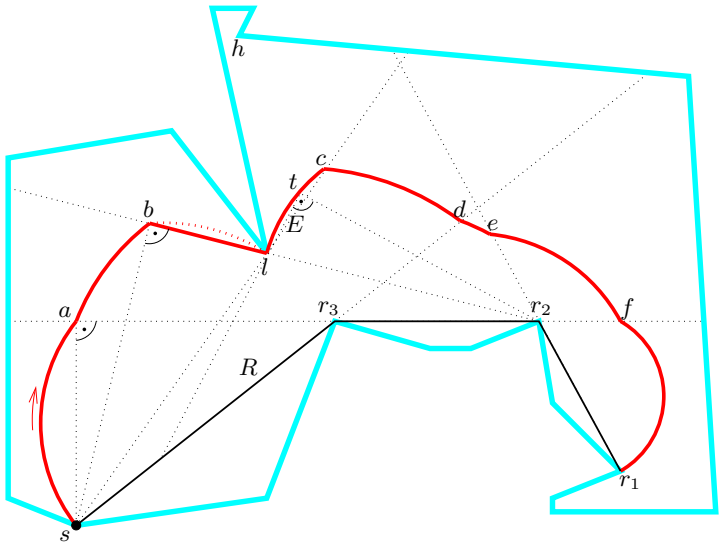


FIG. 2.3. *While executing ExploreRightVertex, the target vertex is initially r_3, then changes to r_2 and finally to r_1.*

At point b, vertex r_2 would become invisible if the robot were to follow the circular arc. But now rule 3 applies, causing the robot to walk straight to the left reflex vertex l. From there, a circular motion is again possible; but the shortest path from s to CP now contains vertex l. By rule 2, the robot continues its approach to r_2 along the arc spanned by l and by r_2.

Notice that at vertex l, also the right vertex h becomes visible, but it is ignored because its shortest path from s makes a left turn at l.

From c on, the shortest path to s is the line segment. Since the circle spanned by s and by r_2 is passing through c, the robot proceeds along that circle, applying rule 2.

At d, the shortest path to s changes again; now it contains vertex r_3. The robot walks along the circle spanned by r_3 and r_2, and gets to point e from which vertex r_1 becomes visible. From here, the robot explores r_1, following the circle spanned by r_3 and by r_1, the former changing to r_2 at f. Eventually, the robot arrives at r_1, thereby fully exploring r_1. Here procedure *ExploreRightVertex* terminates.

In order to provide an upper bound on the length of the resulting path we need to give the definition of the angle hull. A detailed discussion of this topic will follow in section 3.

Let D be a bounded, connected region in the plane. For convenience, we shall assume that D is a simple polygon, but our results can easily be generalized to curved objects by approximation. Now suppose that a photographer wants to take a picture of D that shows as large a portion of D as possible, but no white space. The photographer is using a fixed angle lens. For now, we assume that the angle equals 90°; later, at the end of section 3.2, we will see how to generalize to arbitrary angles.

Before taking the picture, the diligent photographer walks around D and inspects all possible viewpoints. We are interested in comparing the length of the photographer's path to the perimeter of the object, D.

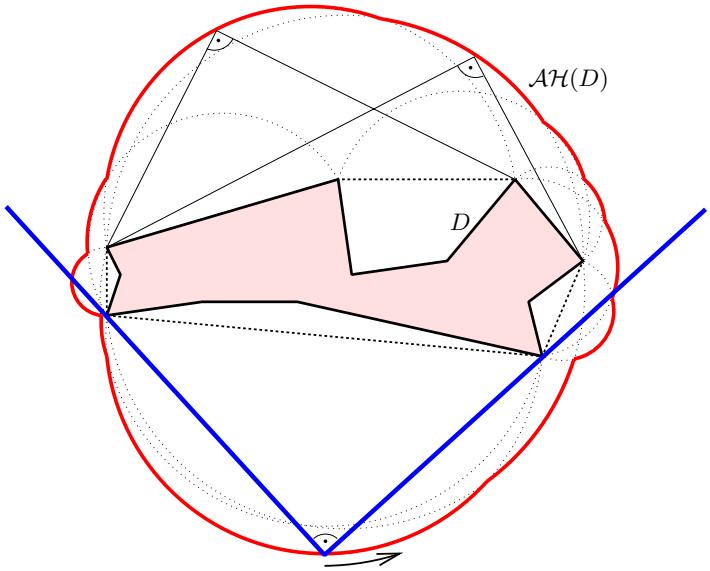


FIG. 2.4. *Drawing the angle hull $\mathcal{AH}(D)$ of a region D.*

In the simple outdoor setting there are no obstacles that can obstruct the photographer's view of D; this situation is depicted in Figure 2.4. At each point of the path, the two sides of the lens' angle touch the boundary of D from the outside, in general at a single vertex each.

While the right angle is touching two vertices, v and w, of D, its apex describes a circular arc spanned by v and w, as follows from Thales' theorem. All points enclosed by the photographer's path, and no other, can see two points of D at a 90° angle; we call this point set the *angle hull* of D and denote it by $\mathcal{AH}(D)$.

Only such vertices can be touched by the right angle that are situated on the convex hull of D. Consequently, the photographer's path depends on the convex hull, $\mathcal{CH}(D)$, of D, rather than on D itself, therefore we have $\mathcal{AH}(D) = \mathcal{AH}(\mathcal{CH}(D))$.

It is not hard to see that, without further obstacles, the perimeter of the angle hull is at most $\pi/2$ times the perimeter of D; the worst case occurs when D is a line segment or a rectangle; see [16].

However, for our application we need to analyze the indoor setting where D is contained in a simple polygon P whose edges give rise to visibility constraints. The photographer does not want any wall segments to appear in the picture; thus, the viewing angle can now be constrained in different ways: Either side may touch a convex vertex of D that is included in the angle, as before, or it may touch a reflex vertex of P that is excluded; see Figure 2.5.

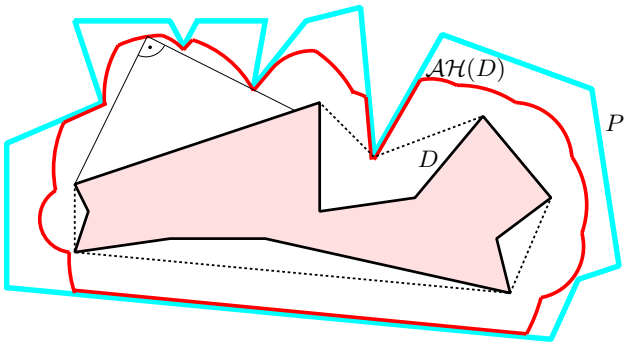


FIG. 2.5. *The angle hull $\mathcal{AH}(D)$ inside a polygon P.*

Any combination of these cases is possible.

As a consequence, the photographer's path contains circular arcs spanned by vertices of D and of P; in addition, it may contain segments of edges of P that prevent the photographer from stepping back far enough; see Figure 2.5.

Formally, we define the angle hull, $\mathcal{AH}(D)$, of D with respect to P to be the set of all points of P that can see two points of D at a right angle. Its boundary equals the photographer's path. In the indoor setting, the angle hull $\mathcal{AH}(D)$ depends only on the relative convex hull, $\mathcal{RCH}(D)$, of D; in other words $\mathcal{AH}(D) = \mathcal{AH}(\mathcal{RCH}(D))$.

In section 3 we show that the angle hull can have at most twice the perimeter of D.

Coming back to our exploration strategy, the crucial observation is that its path is essentially an angle hull (AH).

LEMMA 2.1. *Assume the robot starts at the base point and invokes procedure ExploreRightVertex. Suppose this procedure terminates with the robot reaching the cut of target vertex r_1 at point c. Then the shortest path, R, inside P from the base point to the cut also reaches the cut at c. Moreover the robot's path is part of the boundary of the angle hull $\mathcal{AH}(R)$ of R except for straight line segments leading to blocking vertices.*

Proof. In the discussion of procedure *ExploreRightVertex* above in this section, we have already shown that the robot's path and the shortest path to the cut arrive at the same point c.

Now let t be a point on the robot's path that is not contained on a straight line segment of the path. Assume that, at t, the robot is exploring right reflex vertex r_2, as in the example shown in Figure 2.3. Since r_2 and r_1 are in convex position relative to the base point, vertex r_2 lies on the shortest path, R, from the base point to r_1.

Now consider the shortest path, T, from the base point to t. As a consequence of rule 2 and by Thales' theorem, the last line segment, E, of T is perpendicular to the line through t and r_2. Since the backward prolongation of E is bound to hit R, we know that point t can see two points of R at a right angle. Thus, t belongs to the angle hull $\mathcal{AH}(R)$; it lies on the boundary because the angle's sides are both touching reflex vertices of P or endpoints of the path R. □

To estimate the length of the path from the base point to the cut we make use of the analysis of the angle hull from section 3.

LEMMA 2.2. *The robot's path from the base point to the cut of the target vertex explored by procedure ExploreRightVertex is not longer than twice the length of the shortest path.*

Proof. If the robot's path contains straight line segments leading to blocking vertices, like the segment from b to l in Figure 2.3, these segments are replaced by circular arcs in the angle hull $\mathcal{AH}(R)$. Thus, the robot's path to the cut of r_1 cannot be longer than the angle hull's perimeter. If it ends at the point r_1 itself, as in Figure 2.3, we can apply Theorem 3.5—which states that the arc length of the angle hull of a polygon D in P is less than twice as long as D's boundary—to the shortest path as D and obtain the desired upper bound.

If the robot reaches the cut of r_1 at some point different from r_1, we can arrive at the same conclusion using Corollary 3.6. □

It is important to note that procedure *ExploreRightVertex* ignores such vertices as h in Figure 2.3, whose shortest paths from the current stage point include left turns. Otherwise, it would not be clear how to apply Lemma 2.1.

There is a symmetric procedure *ExploreLeftVertex* which is identical to *ExploreRightVertex*, except that left/right and clockwise/counterclockwise are exchanged.

2.2. Exploring a group of vertices. Each exploration of a group of vertices starts from a *stage point*. The importance of stage points lies in the fact that they are visited by the optimum watchman tour, W_{opt}, too. The first stage point encountered is the robot's start point, s. All stage points are vertices of the shortest path tree of s; the shortest path from s to any vertex of a group leads through the group's stage point.

The exploration of a group of right vertices is performed by procedure *ExploreRightGroup*.

```
PROCEDURE ExploreRightGroup ( in TargetList, out ToDoList );
  StagePoint := CP;
  ToDoList := empty list;
  while TargetList is not empty do
    ExploreRightVertex (TargetList, ToDoList );
      (* CP is now on the cut, C, of the last target. *)
    walk to the point on C that is closest to StagePoint
      while maintaining TargetList and ToDoList;
  walk on the shortest path back to StagePoint;
end ExploreRightGroup;
```

The stage point of a right group is always a left vertex. Initially, *ToDoList* is empty, whereas *TargetList* contains a sorted list of unexplored right vertices whose shortest path from the base point makes only right turns. Among them are all unexplored right vertices visible from *StagePoint*.

Roughly, the group exploration proceeds by repeatedly calling procedure *ExploreRightVertex* introduced in section 2.1 until *TargetList* becomes empty. Afterwards, all right vertices initially present in *TargetList* have been explored, together with their *purely right descendants* in the shortest path tree of s. Here, vertex w is called a *purely right* descendant of vertex v in the shortest path tree of s if w is a right vertex and if the path from v to w makes only right turns. This set of vertices constitutes a *group*, by definition.

On returning from a call to *ExploreRightVertex* the robot has just explored the clockwise first vertex of *TargetList* and is now situated on this vertex's cut. Before it continues, the robot walks along this cut to the point closest to the stage point; this will be the base point in the next execution of *ExploreRightVertex*. The reason for this step will become clear in the proof of Lemma 2.5; essentially, it keeps the robot closer to the optimum watchman tour.

Once the last vertex of *TargetList* has been explored, the robot walks back to the stage point, thus completing the exploration of the group. Now *ToDoList* contains, of all right vertices explored, those who have left children, together with references to the latter.

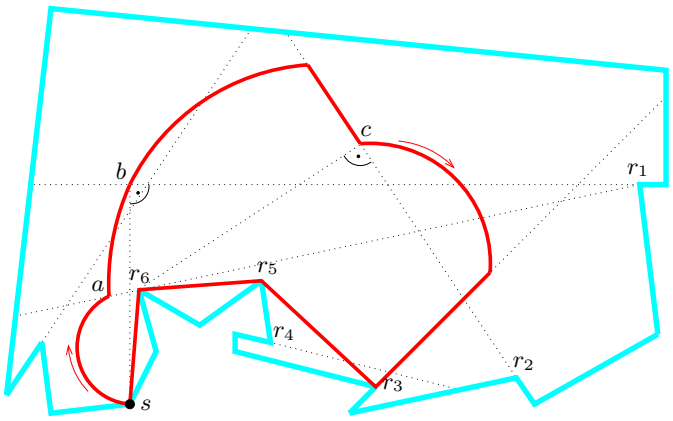


FIG. 2.6. *Exploring a group of right vertices.*

For an example, see Figure 2.6. Point s is the stage point and also the first base point, and *ExploreRightVertex* is called with *First* (*TargetList*) $= r_6$. While exploring r_6, point r_1 is discovered at point a and becomes *First* (*TargetList*). At $CP = b$ procedure *ExploreRightVertex* returns. Meanwhile, r_2 and r_5 have been added to *TargetList* while r_1 and r_6 have been removed. Point b is also the closest point to s on the current cut.

As we continue with exploring *First* (*TargetList*) $= r_2$, point r_5 gets explored on the way. Once the cut of r_2 is reached, we walk to c, the closest point to s on the cut. Similar for r_3; while walking along the cut to the point closest to s, which is r_3 itself, r_4 gets explored and no unexplored right vertices remain.

As before with *ExploreRightVertex*, for *ExploreRightGroup* we also have a symmetric counterpart, *ExploreLeftGroup*.

First we prove a useful structural result which says that the shortest paths to the base points fan out in the same order as the base points are generated.

LEMMA 2.3. *Suppose that procedure ExploreRightGroup generates the base points* $b_1, \ldots, b_m$ *in* m *consecutive calls of subroutine ExploreRightVertex. Then the*

shortest paths from the stage point to $b_1, \ldots, b_m$ are in clockwise order.

Proof. Let base point b_i be situated on the cut of right reflex vertex v_i. Since each call to *ExploreRightVertex* explores the clockwise first right vertex that is still unexplored, $v_1, \ldots, v_m$ appear in clockwise order on the boundary. The stage point must be situated below the cuts of v_i and v_{i+1} as these are unexplored right vertices. The same holds for the last point, p, the shortest paths from the stage point to b_i and b_{i+1} have in common. Moreover, b_i must be below the cut of v_{i+1} because the latter is still unexplored when the robot reaches b_i; see Figure 2.7. Since neither of the shortest paths nor the cut between v_i and b_i can be penetrated by the polygon's boundary, the claim follows. □

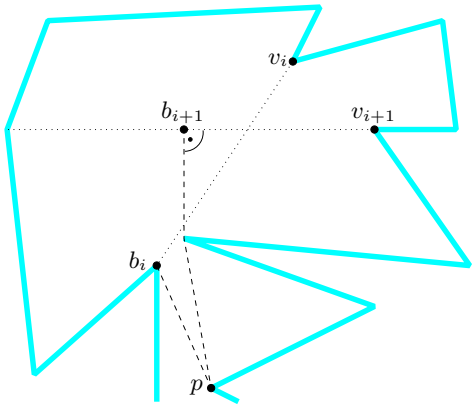


FIG. 2.7. *As seen from p, the shortest path to b_i runs to the left of the shortest path to b_{i+1}.*

Now we turn to analyzing the length of the path the robot spends on exploring a group of vertices.

LEMMA 2.4. *The robot's path between two consecutive base points is at most 3 times as long as the shortest path.*

Proof. Let us call the base points b_1 and b_2, and let c be the point where $cut(v_2)$ is reached. Then c is also the cut's closest point to b_1, by Lemma 2.1. By Lemma 2.2, the robot's path to c is not longer than twice the length of the shortest path from b_1 to c and therefore is also not longer than twice the length of the shortest path from b_1 to b_2; see Figure 2.8.

It remains to account for the walk along the cut from c to b_2. This line segment can be orthogonally projected onto the shortest path from b_1 to b_2 and, therefore, it must be shorter.

Observe that Figure 2.8 is in fact generic: As seen from s, the shortest path to b_1 runs to the left to the shortest path to b_2, by Lemma 2.3; base point b_1 must be located below the cut of v_2; the shortest paths from b_1 to c and from s to b_2 cannot cross because they are both shortest paths to this cut. □

The next steps consist of comparing the length of an *ExploreRightGroup* tour with the relative convex hull of the base points visited.

LEMMA 2.5. *The length of a path caused by a call to ExploreRightGroup does not exceed $3\sqrt{2}$ times the perimeter of the relative convex hull of the base points visited.*

Proof. Let sp be the stage point and $sp = b_0, \ldots, b_{m-1}, b_m = sp$ be the sequence of base points visited by *ExploreRightGroup*. Due to Lemma 2.3, $b_1, \ldots, b_{m-1}$ appear in clockwise order as leaves of the shortest path tree from sp to $b_1, \ldots, b_{m-1}$. So, even

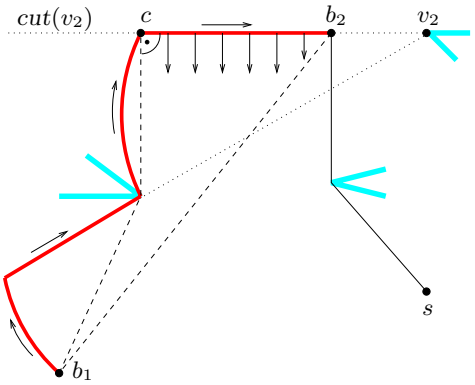


FIG. 2.8. *Line segment $c\,b_2$ must be shorter than the shortest path from b_1 to b_2.*

factor 3 of Lemma 2.4 would apply if all base points b_i were vertices of their relative convex hull, $\mathcal{RCH}$, in P. In the following we show how to reduce to this case.

Suppose that for $i \leq k-2$ the base points b_i and b_k are situated on the boundary of $\mathcal{RCH}$ and the points $b_{i+1}, \ldots, b_{k-1}$ in between are not. The shortest path from *sp* to the cut of each of them must have a right angle to the cut because of Lemma 2.1; see Figure 2.9.

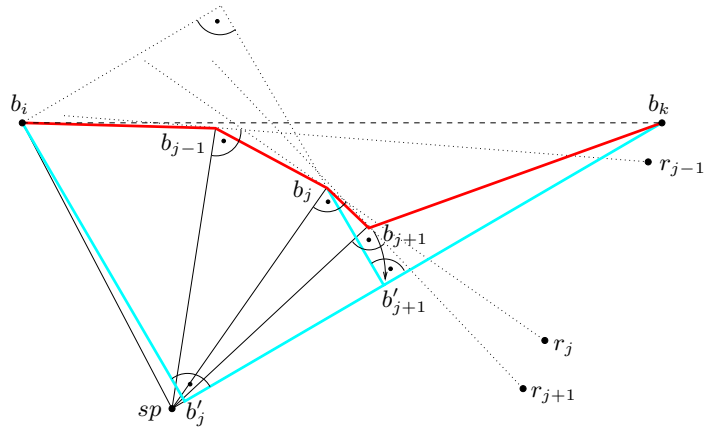


FIG. 2.9. *The cut of vertex r_j containing the base point b_j must not intersect the shortest path from sp to b_{j-1}.*

For $i < j < k$, the cut of b_j must pass above b_{j-1} because otherwise its cut would intersect every possible path from s to b_{j-1}, in particular the robot's path, contradicting the fact that vertex r_j is explored *after* r_{j-1}.

While maintaining these properties, we move the base points one after the other such that the path $b_i, \ldots, b_k$ becomes even longer. For all vertices v of this path, starting with b_{k-1} and going back to b_{i+1}, we do the following. If the path from b_i to b_k makes a left turn at v, like at b_{j+1} in Figure 2.9, then we move v to the point on the shortest path from *sp* to v's successor such that the left turn is a right angle; see b'_{j+1}. Note that every left turn must be an obtuse angle which again is due to the fact that the cut of b_j must pass above b_{j-1}. In case of a right turn we do nothing. Eventually, we end up with a path whose left turns are all right angles.

Now a maximal sequence of right turns, in the example the chain from b_i to b'_{j+1}, can be replaced by one left turn of 90° which is clearly longer than the chain, see b'_j and the rectangle with vertices b_i, b'_j, and b'_{j+1}. Finally, no right turn remains and the new path makes only one left turn of 90° for which the claim is obvious. □

In relating the robot's path to the optimum watchman tour, the following lemma is crucial.

LEMMA 2.6. *All base points are contained in the angle hull* $\mathcal{AH}(W_{opt})$.

Proof. A base point b is, by definition, the closest point to s of a cut. The optimum watchman tour W_{opt} connects the start point s to the cut. Let E be the last edge of the shortest path from s to the cut, i.e., to b.

In most cases, edge E is orthogonal to the cut. Then we have a right angle at b whose one side goes along the cut and touches W_{opt}, while the other side K extends edge E. Either the other endpoint of E equals s, so that K touches W_{opt} in s, or K separates s and the cut because P is simple, and W_{opt} must also be touched by K.

In the remaining case, when there is no right angle between edge E and the cut, point b must be one endpoint of the cut, i.e., it is the target vertex itself or the other endpoint. In both cases the inner angle between E and the cut is necessarily greater than 90°, otherwise there would be a shorter path to the cut. As before either E or its extension meets W_{opt}. □

2.3. Subdividing the polygon. Now we want to combine the exploration of several groups of vertices to finally explore the whole polygon P. This is done by making the *ExploreGroup*-procedures recursive.

```
PROCEDURE ExploreRightGroupRec ( in TargetList );
  ExploreRightGroup ( TargetList, ToDoList );    (*  ToDoList gets filled in. *)
  Clean up ToDoList:
    retain only those right vertices in ToDoList
    which are highest up in the shortest path tree;
  for all vertices v of ToDoList in clockwise order do
    walk on the shortest path to v;    (* connect stage points *)
    ExploreLeftGroupRec( {all known left descendants of v in counterclw. order} );
end ExploreRightGroupRec;
```

The task of *ExploreRightGroupRec* is to explore, from the current position *CP*, all vertices in the input parameter *TargetList* and everything behind.

ExploreRightGroupRec performs in three steps. The *TargetList* is handed over to *ExploreRightGroup*, so *CP* is the new stage point, and after the exploration we are back at this point. We are given a *ToDoList* of candidates for stage points in recursive explorations.

The next step is a necessary cleanup for the *ToDoList*, which contains all purely right descendants of the current stage point which have left children. Some of these right vertices are descendants of others in this list; they must be removed from the list. Only maximal (highest up) right vertices are retained; these will become stage points in further steps. To each of these future stage points we associate a list of all known left descendants that were referenced in *ExploreRightVertex* and *ExploreRightGroup*.

Finally, the remaining vertices in *ToDoList* are visited in clockwise order, at each vertex procedure *ExploreLeftGroupRec* is called to explore the list of all known left descendants (as *TargetList*) from there. *ExploreLeftGroupRec* is the symmetric counterpart of *ExploreRightGroupRec* with one particularity. In the **for** loop, the

vertices in *ToDoList* are also visited in clockwise order. The reason for this will become clear in the proof of Theorem 2.10.

To conclude the bottom-up presentation of our strategy, we show the main program. Its task is, of course, to explore a given polygon, P, starting at a boundary point, s. First, in a call to the nonrecursive *ExploreRightGroup*, the right vertices visible from s are explored. The next target list contains all left children of the right vertices just explored and the left vertices visible from s. All these, and everything behind, gets explored by a call to the recursive *ExploreLeftGroupRec* with this target list.

PROCEDURE *ExplorePolygon* (**in** P, **in** s);

ExploreRightGroup ({clockwise list of all right vertices visible from s}, *ToDoList*);

TargetList := {all left children of the vertices of *ToDoList*};

Add all left vertices visible from s into *TargetList* and sort counterclockwise;

ExploreLeftGroupRec (*TargetList*);

end *ExplorePolygon*;

Each call of *ExploreRightGroup* or *ExploreLeftGroup* generates a set of base points; the first base point of the set is the stage point. For estimating the length of the complete tour, we distribute all these sets into three categories.

The set of base points generated by the call of *ExploreRightGroup* in *ExplorePolygon* belongs to category 0. For the remaining sets, we use their level of recursion to determine their category: the set of base points generated by a call of *ExploreRightGroup* or *ExploreLeftGroup* at total recursion depth i belongs to category (i mod 3).

For example, the very first call of *ExploreLeftGroup* belongs to category 1, and the calls of *ExploreRightGroup* one level deeper belong to category 2. All calls of *ExploreLeftGroup* have an odd level, and all calls of *ExploreRightGroup* an even level.

A key observation is that two sets of base points of the same category will be mutually invisible; see Lemma 2.7 below. Thus the three categories will contribute a factor of 3 to the final analysis in Theorem 2.10 below.

One might wonder why the first nonrecursive call of *ExploreRightGroup* in procedure *ExplorePolygon* is necessary. Without this call the right vertices visible from s, and everything behind, would not be explored because procedure *ExploreLeftGroupRec* deals exclusively with left vertices visible from s and recursively with their offspring.

LEMMA 2.7. *The relative convex hulls of two sets of base points of the same category are mutually invisible, with a possible exception for their stage points.*

Proof. The recursion depths of two sets of base points, B_1 and B_2, of the same category differ by a multiple of 3, possibly 0, as explained above. Let $s_1 \neq s_2$ be the stage points of B_1, resp., B_2. We distinguish two cases depending on the shortest paths from s to s_1 and s_2.

If stage point s_1 is not on the shortest path from s to s_2 and vice versa then let s_0 be the vertex where the shortest paths from s to s_1 and s_2 separate. Without loss of generality we assume that s_2 is a left vertex and that the clockwise order on the boundary is s_0, s_1, s_2. The left picture in Figure 2.10 shows such a situation.

The base points of B_2 are on cuts of right vertices whose shortest paths from s_0 all pass through s_2. Therefore, the shortest path from s_0 to s_2 is invisible from any point of B_2, except s_2. But this shortest path separates B_2 from B_1, they are therefore mutually invisible, except for s_1 and s_2. This argument easily extends to the convex hulls as well.

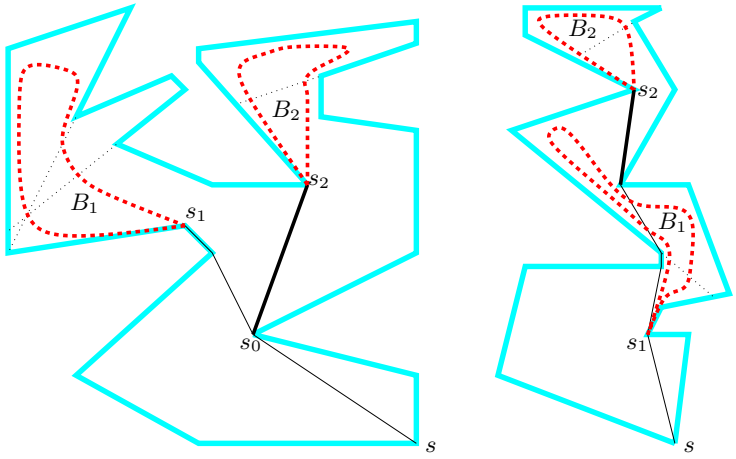


FIG. 2.10. *Base points in B_2 cannot see B_1, two cases.*

Otherwise assume that s_1 lies on the shortest path from s to s_2, see the right picture in Figure 2.10. Then the recursion depths of B_1 and B_2 differ by at least three. Similar to the previous case, no point of B_2 can see the shortest path from s_2 to its parent stage point, but this path definitely separates B_1 and B_2.

Note that a difference of three levels is really necessary. If B_2 is only two levels deeper than B_1 then the stage points s_1 and s_2 are of the same type and the parent stage point of s_2 can be a direct descendant of s_1, and therefore it can very well be contained in $\mathcal{RCH}(B_1)$. □

As a consequence, we conclude that the union of all base points of one category has no shorter perimeter than the perimeters of all of its sets of base points together. Let $\mathrm{per}(\mathcal{RCH}(A))$ denote the perimeter of the relative convex hull of set A.

LEMMA 2.8. *Let B_1 and B_2 be two sets of base points of the same category. Then we have $per(\mathcal{RCH}(B_1)) + per(\mathcal{RCH}(B_2)) \le per(\mathcal{RCH}(B_1 \cup B_2))$.*

As a consequence, we can estimate the path length caused by all calls of *ExploreRightGroup* or *ExploreLeftGroup* in the same category.

LEMMA 2.9. *The path length caused by all calls of ExploreRightGroup and ExploreLeftGroup in one category is less than $6\sqrt{2} \le 8.5$ times the length of W_{opt}.*

Proof. Let the category consist of sets B_i, $i = 1, \ldots,$ of base points. By Lemma 2.5, the length of the path created by one call of *ExploreRightGroup* or *ExploreLeftGroup* with set B_i is not greater than $3\sqrt{2}\,\mathrm{per}(\mathcal{RCH}(B_i))$.

The relative convex hulls $\mathcal{RCH}(B_i)$ are mutually invisible (Lemma 2.7); hence we conclude from Lemma 2.8 for the path length, L, caused by all calls of *ExploreRightGroup* or *ExploreLeftGroup* of this category,

$$L \le 3\sqrt{2} \sum_i \mathrm{per}(\mathcal{RCH}(B_i)) \le 3\sqrt{2}\,\mathrm{per}\left(\mathcal{RCH}\left(\bigcup_i B_i\right)\right).$$

All base points considered are contained in the angle hull of W_{opt}, as Lemma 2.6 has shown, hence the perimeter of their relative convex hull is shorter than the perimeter of $\mathcal{RCH}(\mathrm{AH}(W_{opt}))$.

The perimeter of $\mathcal{RCH}(\mathrm{AH}(W_{opt}))$ is not longer than the perimeter of the angle hull of W_{opt} itself. By Theorem 3.5 this is not greater than twice the length of W_{opt} and the claim follows. □

As the main result for our complete strategy, we obtain a factor of 26.5.

THEOREM 2.10. *For a polygon, P, and a start point s on the boundary of P, a procedure call ExplorePolygon(P, s) explores the polygon and returns to s. The total path length used is less than $(18\sqrt{2}+1) \leq 26.5$ times the length of the optimum watchman tour from s.*

Proof. Since we have three categories of base point sets, all *ExploreRightGroup* and *ExploreLeftGroup* calls together cause a path length of less than $3 \cdot 6\sqrt{2}|W_{opt}|$.

It remains to bound the path length caused by the walks during the **for** loops of *ExploreRightGroupRec* and *ExploreLeftGroupRec*. They connect only stage points by shortest paths, and all those stage points are visited in clockwise order along the boundary of P, independently of whether this is done in *ExploreRightGroupRec* or *ExploreLeftGroupRec*.

The optimum watchman path visits all stage points, and some of them even twice. In any case the sequence of stage points as they appear on the boundary of P is a subsequence of the boundary points of P that are visited by W_{opt} because W_{opt} can not properly cross itself. Therefore, we can be sure that all those walks together make up for an additional path length of at most $|W_{opt}|$. □

3. The angle hull. In on-line navigation algorithms for autonomous robots, analyzing the length of the robot's path is often a complicated issue. Sometimes, only the discovery of certain structural properties has led to a reasonably sharp analysis; see [1, 18, 19, 20] and Rote [24].

Here we provide a new result of this type. It is crucial in analyzing our on-line strategy of section 2, and it seems also to be interesting in its own right.

For convenience, we repeat the definition of the angle hull. Let D be a simply connected region contained in a simple polygon P, then the angle hull, $\mathcal{AH}(D)$, of D with respect to P consists of all points of P that can see two points of D at a right angle. Note that D itself is included in $\mathcal{AH}(D)$. The boundary of the angle hull was denoted the photographer's path of D in section 2.1; for an example see Figure 2.5.

We are now going to analyze the length of the photographer's path, i.e., the perimeter of the angle hull. In section 3.1 we show that, in the indoor setting, the angle hull may have twice the perimeter of D, in the limit. Then, in section 3.2, we prove that this is the worst that can happen.

3.1. The lower bound. We start with the proof that the angle hull of a set D contained in a polygon P can be twice as long as the perimeter of D. Our construction is rather simple, D is a line segment and P is a jagged halfcircle. Region D is called *relatively convex* iff $D = \mathcal{RCH}(D)$.

LEMMA 3.1. *Let $\varepsilon > 0$. There is a polygon, P, and a relatively convex region, D, inside P, for which the boundary of the angle hull $\mathcal{AH}(D)$ with respect to P is longer than $2 - \varepsilon$ times the boundary of D.*

Proof. As our region D, we take a horizontal line segment of length 1. Let p_0, $\ldots$, p_n be equidistant points on the halfcircle spanned by D, where p_0 and p_n are the endpoints of D; see Figure 3.1. From each point p_i we draw the right angle to the endpoints of D. Let P be the concatenation of the upper envelope of these angles and its reflection at D. Then we have $P = \mathcal{AH}(D)$ by construction. Let us analyze the upper envelope.

We will show that the length of the jagged line from p_0 to p_n is less than 2, but comes arbitrarily close to 2, as n increases. Let q_i be the intersection of the segments $p_0\,p_{i+1}$ and $p_i\,p_n$. If we rotate, for all i, the ascending segments $q_i\,p_{i+1}$ about p_0 onto D (see the dotted arcs in Figure 3.1), these segments cover disjoint pieces of D,

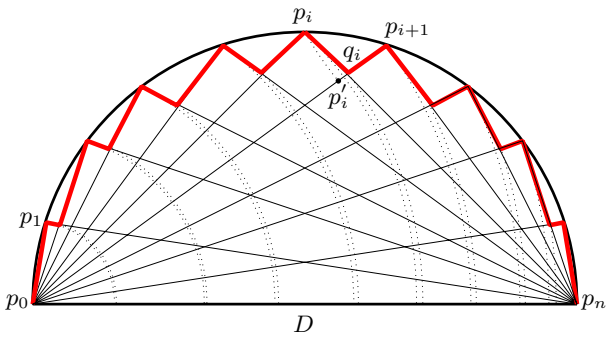


FIG. 3.1. *The boundary of the upper envelope of the right angles is less than* $2|D|$.

so the total length of all ascending segments is always less than 1. By symmetry, the same bound holds for the descending segments. It remains to show that the ascending length can come arbitrarily close to 1.

Consider the triangle $p_i\, q_i\, p'_i$, where p'_i is the orthogonal projection of p_i onto $p_0\, q_i$. Point p_0 is closer to p'_i than to p_i, so for the distances from p_0 to p_i and to q_i we have

$$|p_0\, q_i| - |p_0\, p_i| \le |p_0\, q_i| - |p_0\, p'_i| = |p'_i\, q_i| = |p_i\, q_i| \sin\frac{\pi}{2n}.$$

The total length of all ascending segments is therefore 1 minus the following rest:

$$\sum_i (|p_0\, q_i| - |p_0\, p_i|) \le \sin\frac{\pi}{2n} \sum_i |p_i\, q_i| \le \sin\frac{\pi}{2n}.$$

For $n \to \infty$, this tends to 0. The last inequality holds because $\sum_i |p_i\, q_i| \le 1$ is the length of all descending segments. □

The proof also works for nonequidistant points as long as the maximum distance between subsequent points tends to 0. We are obliged to Seidel [26] for this elegant proof of Lemma 3.1.

3.2. The upper bound. Interestingly, the same jagged lines as used in the proof of Lemma 3.1 are also very useful in the proof of the upper bound. For any circular arc C we can construct a jagged line by distributing auxiliary points along C and by taking the upper envelope of the right angles at these points whose sides pass through the two spanning vertices of C; see Figure 3.2. We denote with *jagged length*, $J(C)$, of C the limit of the lengths of these jagged lines as the maximum distance between subsequent points tends to 0. This limit is well defined, i.e., it does not depend on how the points are chosen. In the proof of Lemma 3.1 we have already seen how to determine this length by separately estimating the lengths of the ascending and descending segments. For the jagged length of a circular arc with diameter 1 from angle α to angle β (see Figure 3.2), we obtain analogously to the proof of Lemma 3.1

$$J(C) = \sin\beta - \sin\alpha - \cos\beta + \cos\alpha,$$

which can also be written as

$$J(C) = \int_\alpha^\beta (\cos\gamma + \sin\gamma)\, d\gamma\,.$$

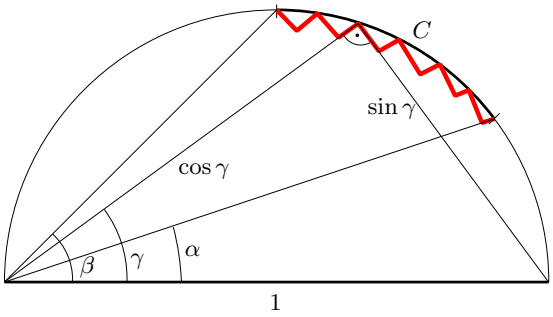


FIG. 3.2. *Analyzing the jagged length, $J(C)$, of a circular arc C.*

LEMMA 3.2. *The jagged length of an arc is always greater than the arc length itself.*

Proof. Consider a circle with diameter d and a circular arc a on its boundary. Two lines from an arbitrary point on the boundary through the endpoints of the arc always intersect in the same angle ϕ, by the generalized Thales's theorem. For the length of a, we have $|a| = \phi d$; see Figure 3.3.

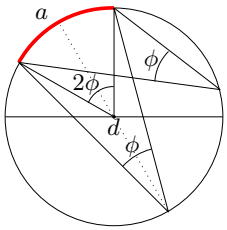


FIG. 3.3. $|a| = \phi d$.

So the arc length of the arc C in Figure 3.2 equals $\beta - \alpha$, and we have

$$J(C) = \int_\alpha^\beta (\cos\gamma + \sin\gamma)\, d\gamma \geq \int_\alpha^\beta 1\, d\gamma = \beta - \alpha\,;$$

the inequality follows from $\cos\gamma + \sin\gamma \geq 1$ for $\gamma \in [0, \frac{\pi}{2}]$. □

The integral form for the jagged length also has a geometric interpretation. Let us consider a right angle with slope γ contained in the halfcircle, as shown in Figure 3.2. The length of the two sides of the right angle equals $\cos\gamma + \sin\gamma$. If we define

$$C_\gamma := \begin{cases} \text{length of the right angle} & \text{if its apex is contained in } C, \\ 0 & \text{otherwise,} \end{cases}$$

we obtain the nice form

$$J(C) = \int_0^{\frac{\pi}{2}} C_\gamma\, d\gamma\,.$$

This form is used in the proof of the next lemma.

LEMMA 3.3. *Let D be a line segment, and let P be a surrounding polygon such that P and the angle hull $\mathcal{AH}(D)$ with respect to P touch only at vertices of P; see*

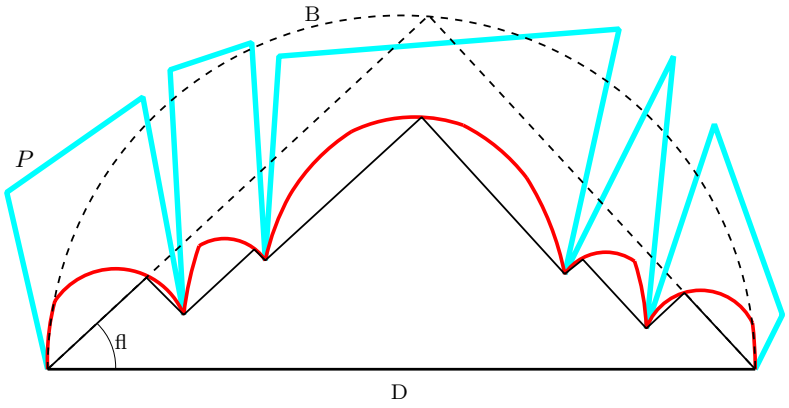


FIG. 3.4. *For a line segment D we have $J(\mathcal{AH}(D)) = J(B) = 2|D|$.*

Figure 3.4. *Then the arc length of $\mathcal{AH}(D)$ with respect to P from one endpoint of D to the other is less than $2|D|$.*

Proof. By Lemma 3.2, the arc length of $\mathcal{AH}(D)$ is certainly shorter than the jagged length of $\mathcal{AH}(D)$, i.e., the sum of the jagged lengths of all circular arcs of $\mathcal{AH}(D)$, and we obtain

$$\begin{aligned} \mathit{length}(\mathcal{AH}(D)) \le J(\mathcal{AH}(D)) &= \sum_{C\in\mathcal{AH}(D)} J(C) \\ &= \sum_{C\in\mathcal{AH}(D)} \int_0^{\frac{\pi}{2}} C_\gamma \, d\gamma = \int_0^{\frac{\pi}{2}} \left(\sum_{C\in\mathcal{AH}(D)} C_\gamma \right) d\gamma \,. \end{aligned}$$

But for any angle γ the sum over the lengths of the right angles of slope γ which are contained in the halfcircles of the different circular arcs of $\mathcal{AH}(D)$ is equal to the length, B_γ, of the big right angle in the halfcircle B spanned by the two endpoints of D, which means that

$$\int_0^{\frac{\pi}{2}} \left(\sum_{C\in\mathcal{AH}(D)} C_\gamma \right) d\gamma = \int_0^{\frac{\pi}{2}} B_\gamma \, d\gamma = J(B) = 2|D| \,. \qquad \square$$

Note that in the proof of Lemma 3.3 the halfcircle B does not depend on P, and $J(\mathcal{AH}(D)) = J(B)$ therefore means that the jagged lengths of the angle hulls of D for different surrounding polygons P are all identical! We may also say that we have bounded the length of the angle hull with respect to a surrounding polygon P by the jagged length of the angle hull without obstacles.

LEMMA 3.4. *The statement of Lemma* 3.3 *remains true if D is a convex chain instead of a line segment.*

Proof. We consider a convex chain, D, and a surrounding polygon, P, such that P and the angle hull $\mathcal{AH}(D)$ with respect to P touch only at vertices of P.

We make a construction similar to the proof of Lemma 3.3. For an angle γ we find the tangent to D with that slope. Starting with the touching vertex we go into direction γ until we hit an arc of the angle hull, then we turn by a right angle and go to the vertex of P (or D) that co-spans the current arc. Here we turn back to the original direction and continue accordingly to obtain a connected chain of right angles; see Figure 3.5.

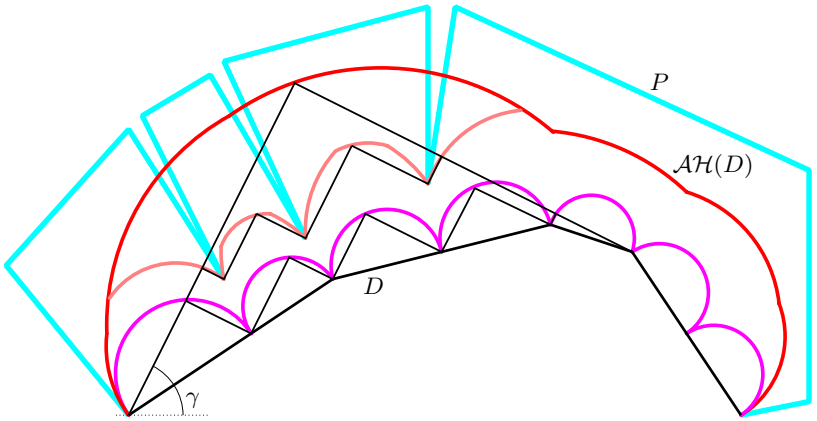


FIG. 3.5. *Three chains of right angles which are all of the same length.*

This chain has the same length as the two sides of the "unfolded" big right angle of slope γ which generates the angle hull without obstacles. As before, this shows that the jagged length of the angle hull does not depend on P.

Now it is not difficult to see how long it really is. Consider the set of halfcircles spanned by the segments of D. Analogously to the previous construction, we can construct the chain of right angles of slope γ below these halfcircles which again has the same length. But these right angles represent the jagged lengths of the isolated segments, and each of them equals twice the length of the segment, by Lemma 3.3. Therefore the total length of the angle hull of D is less than twice the length of D. □

To obtain our main result we need to consider an arbitrary surrounding polygon P that influences the angle hull not only with acute reflex vertices but also with its edges.

THEOREM 3.5. *Let P be a simple polygon containing a relatively convex polygon D. The arc length of the boundary of the angle hull, $\mathcal{AH}(D)$, with respect to P is less than* 2 *times the length of D's boundary. This bound is tight.*

The bound also holds if there are several obstacles instead of P that influence the angle hull and also if their boundaries consist of arbitrary curves.

Proof. Each convex chain of D can be treated separately because the angle hull must pass through the reflex vertices of D.

First, we consider the angle hull $\mathcal{AH}_1(D)$ with respect to only the vertices of P as obstacle points. Its arc length is less than $2|D|$, by Lemma 3.4.

Now also the edges come into play. The angle hull $\mathcal{AH}_2(D)$ with respect to the whole of P contains circular arcs and some pieces of P's edges, for an example see Figure 2.5. The circular arcs of $\mathcal{AH}_2(D)$ are also part of $\mathcal{AH}_1(D)$.

For every piece of an edge which contributes to $\mathcal{AH}_2(D)$, the piece's two endpoints are also on the boundary $\mathcal{AH}_1(D)$. Therefore, $\mathcal{AH}_2(D)$ can only be shorter than $\mathcal{AH}_1(D)$.

The bound is tight by Lemma 3.1.

The proof easily generalizes to the case of several obstacles around D that influence the angle hull instead of P. Indeed, we have never used the fact that the parts of P that are touching the angle hull are connected by edges of P. And if we have several obstacles, we can always connect them to a single one by edges which do not influence the angle hull.

The proof carries over to arbitrary curves by approximation of these curves with polygons. □

The following variation of Theorem 3.5 for an "incomplete angle hull" is used for analyzing the exploration strategy in section 2.

COROLLARY 3.6. *Consider a convex chain, D, from s to t and its angle hull from s to some point g. The jagged length of this part of the angle hull is bounded by twice the length of the shortest path around D from s to g.*

Proof. Let m be the last segment of D and let α be the angle between m and the last segment of the shortest path from s to g; see Figure 3.6.

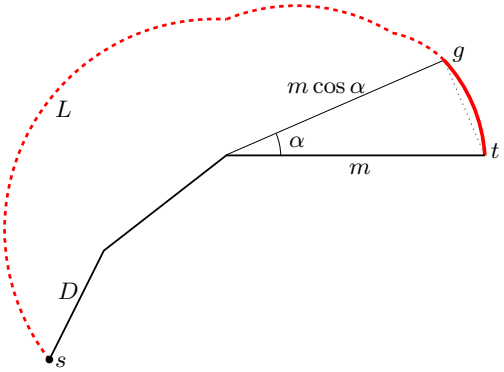


FIG. 3.6. *The cut of r_1 is reached at point g.*

The jagged length of the angle hull from g to t equals $m(1+\sin\alpha-\cos\alpha)$. The path, L, from the base point to g can therefore be estimated in the following way, using Theorem 3.5:

$$\begin{aligned} L &\le 2|D| - m(1+\sin\alpha-\cos\alpha) \\ &= 2(|D| - m + m\cos\alpha) + m(1-\cos\alpha-\sin\alpha) \\ &\le 2(|D| - m + m\cos\alpha). \end{aligned}$$

The last inequality holds because of $0 \le \alpha \le \frac{\pi}{2}$. But $|D| - m + m\cos\alpha$ is exactly the length of the shortest path from the base point to g. □

Remark. The result of Theorem 3.5 can be generalized to angles $\phi \neq 90°$. Namely, for the jagged length of a circular arc C spanned with fixed angle ϕ by a chord of length 1 from angle α to angle β we have

$$J_\phi(C) = (\sin\beta - \sin\alpha - \cos\beta + \cos\alpha)\frac{\cos\phi+1}{\sin^2\phi}$$

from which a tight factor of $2(\cos\phi+1)/\sin^2\phi$ follows; see [16] for this and other interesting extensions. Interestingly, the angle hull with respect to an arbitrary angle has independently been described, in a different context and without estimations about its length, by de Berg et al. [10]—this corresponds to our "outdoor setting"—and in the successor article by Cheong and van Oostrum [8] for the "indoor setting," i.e., with obstacles.

4. Conclusions. We have seen that a combination of suitable analysis techniques is necessary for proving an upper bound for the competitive factor of a rather simple strategy. Still, we believe that its actual performance, even in the worst case,

is considerably better than the proven bound. Establishing a lower bound for polygon exploration, higher than the trivial $(1+\sqrt{2})/2 \approx 1.207$, and closing the gap to the upper bound seem to be challenging problems.

There are many interesting variations and generalizations of the polygon exploration problem. For example, one could study different cost models for the robot's motion. Also, the case of polygons with holes deserves investigation. Here the off-line problem becomes NP-hard, by reduction from the traveling salesperson problem. Recently, Albers, Kursawe, and Schuierer [2] have shown that in a rectilinear environment no better competitive factor than $O(\sqrt{k})$ can be achieved for the on-line problem in the presence of k rectilinear holes, what was known before only for general polygons.

We have also introduced a new type of hull operator that suits us well in analyzing the on-line exploration strategy and that is interesting in its own right. Here we have analyzed the perimeter of the angle hull, $\mathcal{AH}(D)$, in terms of the perimeter of the region D. A number of interesting questions remain open: If we consider a subset of D, is the perimeter of its angle hull always shorter than the perimeter of $\mathcal{AH}(D)$? Does the iterated construction of the angle hull approximate a circle? How can angle hulls be generalized, and analyzed, in three dimensions?

Acknowledgment. We would like to thank the anonymous referees for their valuable comments.

REFERENCES

[1] O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe, and G. Rote, *Generalized self-approaching curves*, in Proceedings of the 9th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 1533, Springer-Verlag, Berlin, 1998, pp. 317–326.

[2] S. Albers, K. Kursawe, and S. Schuierer, *Exploring unknown environments with obstacles*, in Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, 1999, pp. 842–843.

[3] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell, *Approximation Algorithms for Lawn Mowing and Milling*, Tech. Report, Mathematisches Institut, Universität zu Köln, 1997.

[4] P. Berman, *On-line searching and navigation*, in On-line Algorithms: The State of the Art, A. Fiat and G. Woeginger, eds., Springer-Verlag, Berlin, 1998, pp. 232–241.

[5] A. Blum, P. Raghavan, and B. Schieber, *Navigating in unfamiliar geometric terrain*, SIAM J. Comput., 26 (1997), pp. 110–137.

[6] S. Carlsson and H. Jonsson, *Computing a shortest watchman path in a simple polygon in polynomial time*, in Proceedings of the 4th Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, 1995, pp. 122–134.

[7] S. Carlsson, H. Jonsson, and B. J. Nilsson, *Finding the shortest watchman route in a simple polygon*, in Proceedings of the 4th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 762, Springer-Verlag, Berlin, 1993, pp. 58–67.

[8] O. Cheong and R. van Oostrum, *Reaching a polygon with directional uncertainty*, Internat. J. Comput. Geom. Appl., (2001), to appear.

[9] W.-P. Chin and S. Ntafos, *Shortest watchman routes in simple polygons*, Discrete Comput. Geom., 6 (1991), pp. 9–31.

[10] M. de Berg, L. Guibas, D. Halperin, M. Overmars, O. Schwarzkopf, M. Sharir, and M. Teillaud, *Reaching a goal with directional uncertainty*, Theoret. Comput. Sci., 140 (1995), pp. 301–317.

[11] X. Deng, T. Kameda, and C. Papadimitriou, *How to learn an unknown environment*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, CA, 1991, pp. 298–303.

[12] X. Deng, T. Kameda, and C. Papadimitriou, *How to learn an unknown environment* I: *The rectilinear case*, J. ACM, 45 (1998), pp. 215–245.

[13] A. Fiat and G. Woeginger, eds., *On-line Algorithms: The State of the Art*, Lecture Notes in Comput. Sci. 1422, Springer-Verlag, Berlin, 1998.

[14] M. Hammar and B. J. Nilsson, *Concerning the time bounds of existing shortest watchman route algorithms*, in Proceedings of the 11th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. 1279, Springer-Verlag, Berlin, 1997, pp. 210–221.

[15] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel, *A competitive strategy for learning a polygon*, in Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, 1997, pp. 166–174.

[16] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel, *Moving an angle around a region*, in Proceedings of the 6th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 1432, Springer-Verlag, Berlin, 1998, pp. 71–82.

[17] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel, *The polygon exploration problem: A new strategy and a new analysis technique*, in Robotics: The Algorithmic Perspective, Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics, A.K. Peters, Wellesley, MA, 1998, pp. 211–222.

[18] C. Icking and R. Klein, *Searching for the kernel of a polygon: A competitive strategy*, in Proceedings of the 11th Annual ACM Symposium on Computational Geometry, ACM, New York, 1995, pp. 258–266.

[19] C. Icking, R. Klein, and E. Langetepe, *An optimal competitive strategy for walking in streets*, in Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 110–120.

[20] C. Icking, R. Klein, and L. Ma, *How to look around a corner*, in Proceedings of the 5th Canadian Conference on Computational Geometry, University of Waterloo, Waterloo, ON, Canada, 1993, pp. 443–448.

[21] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier, North-Holland, Amsterdam, 2000, pp. 633–701.

[22] S. Ntafos, *Watchman routes under limited visibility*, in Proceedings of the 2nd Canadian Conference on Computational Geometry, University of Ottawa, Ottawa, ON, Canada, 1990, pp. 89–92.

[23] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar, *Robot Navigation in Unknown Terrains: Introductory Survey of Non-heuristic Algorithms*, Tech. Report ORNL/TM-12410, Oak Ridge National Laboratory, Oak Ridge, TN, 1993.

[24] G. Rote, *Curves with increasing chords*, Math. Proc. Cambridge Philos. Soc., 115 (1994), pp. 1–12.

[25] J.-R. Sack and J. Urrutia, eds., *Handbook of Computational Geometry*, North-Holland, Amsterdam, 2000.

[26] R. Seidel, *private communication,* 1997.

[27] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Commun. ACM, 28 (1985), pp. 202–208.

[28] X. Tan and T. Hirata, *Constructing shortest watchman routes by divide-and-conquer*, in Proceedings of the 4th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 762, Springer-Verlag, Berlin, 1993, pp. 68–77.

[29] X. Tan, T. Hirata, and Y. Inagaki, *Corrigendum to "An incremental algorithm for constructing shortest watchman routes,"* Internat. J. Comput. Geom. Appl., 9 (1999), pp. 319–323.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 601–625

ON THE COMPUTATIONAL COMPLEXITY OF UPWARD AND RECTILINEAR PLANARITY TESTING*

ASHIM GARG† AND ROBERTO TAMASSIA‡

Abstract. A directed graph is upward planar if it can be drawn in the plane such that every edge is a monotonically increasing curve in the vertical direction and no two edges cross. An undirected graph is rectilinear planar if it can be drawn in the plane such that every edge is a horizontal or vertical segment and no two edges cross. Testing upward planarity and rectilinear planarity are fundamental problems in the effective visualization of various graph and network structures. For example, upward planarity is useful for the display of order diagrams and subroutine-call graphs, while rectilinear planarity is useful for the display of circuit schematics and entity-relationship diagrams.

We show that upward planarity testing and rectilinear planarity testing are NP-complete problems. We also show that it is NP-hard to approximate the minimum number of bends in a planar orthogonal drawing of an n-vertex graph with an $O(n^{1-\epsilon})$ error for any $\epsilon > 0$.

Key words. graph drawing, planar drawing, upward drawing, rectilinear drawing, orthogonal drawing, layout, ordered set, planar graph, algorithm, computational complexity, NP-complete problem, approximation algorithm

AMS subject classifications. 05C62, 06A06, 65D18, 68Q17

PII. S0097539794277123

1. Introduction.

Graph drawing addresses the problem of constructing geometric representations of abstract graphs and networks [6, 7]. It is an emerging area of research that combines flavors of topological graph theory and computational geometry. The automatic generation of drawings of graphs has important applications in key computer technologies such as software engineering, database design, visual interfaces, and computer-aided design.

Various graphic standards have been proposed for the representation of graphs in the plane. Usually, each vertex is represented by a point and each edge (u, v) is represented by a simple open Jordan curve joining the points associated with vertices u and v. A *straight-line* drawing maps each edge into a straight-line segment. A drawing is *planar* if no two edges cross. A graph (or digraph) is planar if it admits a planar drawing. A drawing of a digraph is *upward* if every edge is monotonically nondecreasing in the y-direction. A drawing of a digraph is *planar upward* if it is planar and upward. A digraph is *upward planar* if it admits a planar upward drawing. Figure 1.1(a) shows a planar straight-line upward drawing. An *orthogonal* drawing maps each edge into a chain of horizontal and vertical segments. A *rectilinear* drawing is an orthogonal straight-line drawing, i.e., a drawing where every edge is either a horizontal or a vertical segment. A graph is *rectilinear planar* if it admits a planar rectilinear drawing. Figure 1.1(b) shows a planar rectilinear drawing.

*Received by the editors November 16, 1994; accepted for publication (in revised form) July 12, 1999; published electronically October 11, 2001. This research was supported in part by the National Science Foundation under grants CCR-9423847, CCR-9732327, and IIS-9985136 (CAREER Award), and by the Army Research Office under grant DAAH04-96-1-0013. A preliminary version of this paper was presented at *Graph Drawing* '94, Princeton, NJ, 1994.

http://www.siam.org/journals/sicomp/31-2/27712.html

†Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (agarg@cse.buffalo.edu). The work of this author was performed in part while he was with the Department of Computer Science at Brown University.

‡Center for Geometric Computing, Department of Computer Science, Brown University, Providence, RI 02912-1910 (rt@cs.brown.edu).

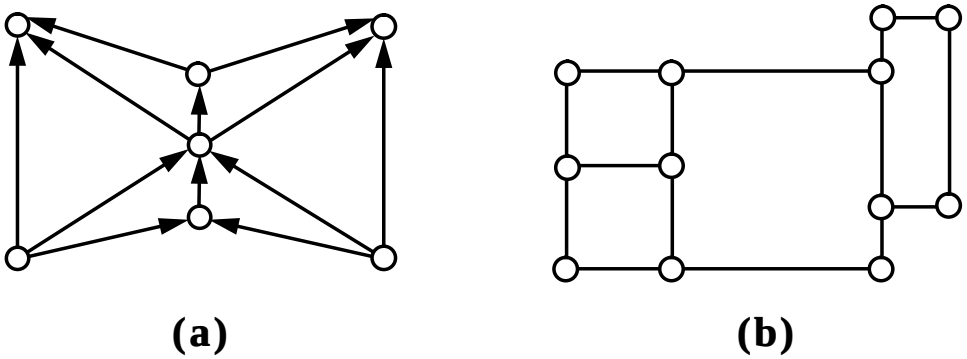


FIG. 1.1. *Examples of* (a) *a planar straight-line upward drawing of a digraph and* (b) *a planar rectilinear drawing of a graph.*

Testing upward planarity and rectilinear planarity are fundamental problems in the effective visualization of various graph and network structures. For example, upward planarity is useful for the display of order diagrams and subroutine-call graphs, while rectilinear planarity is useful for the display of circuit schematics and entity-relationship diagrams. In this paper, we show that the following two problems are NP-complete:

Upward planarity testing. Testing whether a digraph is upward planar.

Rectilinear planarity testing. Testing whether a graph is rectilinear planar.

These problems have challenged researchers in order theory, topological graph theory, computational geometry, and graph drawing for many years. Our intractability results motivate the following observations:

- Testing whether a graph admits a planar drawing or an upward drawing can be done in linear time. Combining the two properties makes the problem NP-hard.
- Every planar graph admits a planar straight-line drawing. Hence, we can say that planarity is equivalent to straight-line planarity, and both properties can be verified in linear time. We can view upward and rectilinear planarity as derived from straight-line planarity by adding further constraints, which apparently make the problem become much more difficult.

We also show that it is NP-hard to approximate the minimum number of bends in a planar orthogonal drawing of an n-vertex graph with an $O(n^{1-\epsilon})$ error for any $\epsilon > 0$.

Previous results on upward and rectilinear planarity testing are summarized below. In the rest of this section, we denote with n the number of vertices of the graph being considered.

Combinatorial results on upward planarity of covering digraphs of lattices were first given in [17, 24]. Further results on the interplay between upward planarity and ordered sets are surveyed by Rival [25, 26, 27]. Lempel, Even, and Cederbaum [18] relate the planarity of biconnected undirected graphs to the upward planarity of *st*-digraphs. A combinatorial characterization of upward planar digraphs is provided in [10, 16]; namely, a digraph is upward planar if and only if it is a spanning subgraph of a planar *st-digraph*. This characterization implies that upward planarity testing is in NP.

Di Battista, Liu, and Rival [9] show that every planar bipartite digraph is up-

ward planar. Papakostas [23] gives a polynomial-time algorithm for upward planarity testing of outerplanar digraphs. Bertolazzi and Di Battista [2] and Bertolazzi et al. [3] give a polynomial-time algorithm for testing upward planarity of triconnected digraphs and of digraphs with a fixed embedding. Concerning single-source digraphs, Thomassen [33] characterizes upward planarity in terms of forbidden circuits. Hutton and Lubiw [14] combine Thomassen's characterization with a decomposition scheme to test upward planarity of a single-source digraph in $O(n^2)$ time. Bertolazzi et al. [4] show that upward planarity testing of a single-source digraph can be done optimally in $O(n)$ time. They also give a parallel algorithm that runs in $O(\log n)$ time on a CRCW PRAM with $n \log\log n / \log n$ processors.

Di Battista and Tamassia [10] and Di Battista, Tamassia, and Tollis [11] give algorithms for constructing upward planar drawings of planar *st*-digraphs and investigate area bounds and symmetry display. Tamassia and Vitter [32] show that the above drawing algorithms can be efficiently parallelized. Upward planar drawings of series-parallel digraphs are studied in [1].

Regarding rectilinear planarity testing, Shiloach [28] and Valiant [34] show that any planar graph of degree at most 4 admits a planar orthogonal drawing. Vijayan and Wigderson [35] study structural properties of rectilinear planar drawings. From their results, the membership of rectilinear planarity testing in NP is easy to establish. Storer [29], Tamassia and Tollis [31], Liu et al. [20], Liu, Morgana, and Simeone [21, 22], Liu, Marchioro, and Petreschi [19], Even and Granot [12], Kant [15], and Biedl and Kant [5] give various techniques for constructing planar orthogonal drawings with $O(n)$ bends. Tamassia [30] gives an $O(n^2 \log n)$-time algorithm that constructs a planar orthogonal drawing with the minimum number of bends for an embedded planar graph. Di Battista, Liotta, and Vargiu [8] give polynomial-time algorithms for minimizing bends in planar orthogonal drawings of series-parallel and cubic graphs. The latter two results show that rectilinear planarity testing can be done in polynomial time for a fixed embedding or for special classes of graphs.

Our proof techniques are based on a two-phase reduction from the known NP-complete problem NOT-ALL-EQUAL-3-SAT. In the first phase, we reduce NOT-ALL-EQUAL-3-SAT to an auxiliary undirected flow problem. In the second phase, we reduce this undirected flow problem to the upward (or rectilinear) planarity testing of a special class of digraphs. The latter reduction is interesting on its own and provides new insights on the characterization by flow networks of the angles formed by the edges of upward planar drawings [2, 3] and orthogonal drawings [8, 30].

The rest of this paper is organized as follows. Preliminary definitions and results are provided in section 2. The reduction from NOT-ALL-EQUAL-3-SAT to the auxiliary flow problem is given in section 3. Sections 4 and 5 describe the reductions from the auxiliary flow problem to upward and rectilinear planarity testing, respectively. Conclusive remarks are given in section 6.

2. Preliminaries. We assume that the reader is familiar with the standard concepts and definitions on NP-completeness [13]. Our results use reductions from the following well-known NP-complete problem:

NOT-ALL-EQUAL-3-SAT. Given a set of clauses with three literals each, is there a truth assignment such that each clause has at least one *true* literal and one *false* literal?

An *embedding* of a planar graph is the collection of circular permutations of the edges incident upon each vertex in a planar drawing of the graph. An *embedded graph* is a planar graph equipped with an embedding. We do not distinguish between a

graph and its embedding if the embedding is unique and the meaning is clear from the context. A vertex has *degree d* if it has d edges incident upon it. A *degree-d* graph is one each of whose vertices has degree at most d.

We denote by $G - \{v\}$ the subgraph of graph G obtained by removing vertex v and its incident edges from G.

The *angles* of an embedded *undirected* graph are the pairs of consecutive edges incident on the same vertex. The *angles* of an embedded *directed* graph are the pairs of consecutive incoming and outgoing edges incident upon the same vertex. Such angles are mapped to geometric angles in a straight-line drawing of the graph.

A *labeled embedding* of an *undirected* graph G is an embedding of G in which each angle is assigned a label from the set $\{1, 2, 3, 4\}$. A *labeled embedding* of a *directed* graph G is an embedding of G in which each angle is assigned a label from the set $\{small, large\}$.

A *rectilinear embedding* of a graph G is a labeled embedding of G such that there exists a rectilinear drawing of G in which each angle labeled ℓ in the embedding measures $\ell\pi/2$ in the drawing. Each rectilinear embedding has a unique external face.

The following definitions are from [2, 3]. An *upward embedding* of a directed graph (digraph) $\vec{G}$ is a labeled embedding of $\vec{G}$ such that there exists a planar straight-line upward drawing of $\vec{G}$ where each angle labeled *small* has measure $< \pi$ and each angle labeled *large* has measure $> \pi$. Each upward embedding has a unique external face.

A *source* of a digraph $\vec{G}$ is a vertex with all outgoing edges, and a *sink* of $\vec{G}$ is a vertex with all incoming edges. A *switch* of $\vec{G}$ is a source or sink of $\vec{G}$. A source or sink of a face f of $\vec{G}$ is called a *local* source or sink of f. Note that a local switch of f may or may not be a switch of $\vec{G}$.

2.1. Upward and rectilinear embeddings. In this section, we give some lemmas on upward and rectilinear embeddings that will be used extensively in the proofs.

We first give some definitions. In an embedding of a digraph $\vec{G}$, a vertex is *bimodal* if its incident edges can be partitioned into two (possibly empty) sets of consecutive edges consisting of its incoming and outgoing edges, respectively. An embedding of $\vec{G}$ is *bimodal* if each vertex is bimodal.

Consider an assignment of labels from the set {*small*, *large*} to the angles of an embedding of $\vec{G}$. For a face f of the embedding, let $L(f)$ and $S(f)$ be the number of angles of f with label *large* and *small*, respectively. Face f is said to be *consistently assigned* if

$$L(f) - S(f) = \begin{cases} -2 & \text{if } f \text{ is an internal face,} \\ +2 & \text{if } f \text{ is the external face.} \end{cases}$$

An assignment of labels to the angles of an embedding of digraph $\vec{G}$ is a *consistent assignment* if

- all the angles at a nonsource or nonsink vertex of $\vec{G}$ are assigned label *small*,
- exactly one angle at a source or sink vertex of $\vec{G}$ is assigned label *large*, and
- each face is consistently assigned.

We paraphrase a result of [2, 3] in the following lemma.

LEMMA 2.1. *An embedding of a digraph $\vec{G}$ can be extended to an upward embedding if and only if it is bimodal and admits a consistent assignment of labels to its angles.*

Let G be an undirected graph of degree at most 4. Consider an assignment of labels from the set $\{1, 2, 3, 4\}$ to the angles of an embedding of G. For a face f of the embedding, let $N_i(f)$ be the number of angles of f with label i. Face f is said to be *consistently rectilinearly assigned* if

$$2 \cdot N_4(f) + N_3(f) - N_1(f) = \begin{cases} -4 & \text{if } f \text{ is an internal face,} \\ +4 & \text{if } f \text{ is the external face.} \end{cases}$$

An assignment of labels to the angles of an embedding of a graph G is a *consistent rectilinear assignment* if

- the sum of the labels of the angles around each vertex is 4, and
- each face is consistently rectilinearly assigned.

The following lemma is an immediate consequence of the results in [30, 35].

LEMMA 2.2. *An embedding of a graph G can be extended to a rectilinear embedding if and only if it admits a consistent rectilinear assignment of labels to its angles.*

2.2. Tendrils and wiggles. We now define several graphs that will be used as gadgets in our reductions.

Tendril T_k, for $k \geq 0$, is an acyclic digraph with two designated poles, a *source pole* denoted by s and a *sink pole* denoted by t, and is defined recursively as follows:

- Tendril T_0 consists of a single directed edge (s, t).
- Tendril T_1 is the 10-vertex graph shown in Figure 2.1(a).
- Tendril T_k is constructed from T_{k-1} by adding the graph H_k shown in Figure 2.1(b) to it by identifying edges (and their endpoints) e_{k-1} of T_{k-1} and f_k of graph H_k (Figure 2.1(c)).

It is easy to see that T_k has exactly $k + 1$ sources and $k + 1$ sinks. Figure 2.2(a) and Figure 2.2(b) show tendrils T_2 and T_3, respectively.

LEMMA 2.3. *Tendril T_k is upward planar and admits a unique upward embedding.*

Proof. We use induction on k. It is straightforward to verify that T_1 has a unique upward embedding by applying Lemma 2.1 to their $O(1)$ planar embeddings. Now suppose that T_{k-1} has a unique upward embedding. Again, it can be easily verified that H_k has a unique upward embedding by applying Lemma 2.1 to its $O(1)$ planar embeddings. Hence, for finding upward embeddings of T_k, we need only to consider the four planar embeddings of T_k obtained by flipping the upward embeddings of T_{k-1} and H_k around their common edge f_k. Applying Lemma 2.1 to the four faces containing both endpoints of f_k shows that only one of these four planar embeddings can be extended to an upward embedding of T_k. □

In the upward planar embedding of T_k, the external face consists of two paths, namely, the *outer* and *inner* paths, between s and t. The outer path has $2k$ large angles and no small angles, and the *inner path* has $2k$ small angles and no large angles. The outer and inner paths of T_2 are drawn with shaded and solid thick lines, respectively, in Figure 2.2(a). When T_k replaces an edge of an embedded planar digraph, the outer (inner) path becomes a subpath of a face f, and we say that the *contribution* of the outer (inner) path to f is $+2k$ $(-2k)$.

Figure 2.2(b) shows a *wiggle* W_k, which consists of a chain of $2k + 1$ edges whose orientations alternate along the chain. The two extreme vertices of W_k are called its *source* and *sink* poles, respectively, and are denoted by s and t, respectively. Later, in section 4, we will consider transformations where a directed edge (u, v) of an embedded digraph is replaced with W_k such that s is identified with u and t with v, and W_k

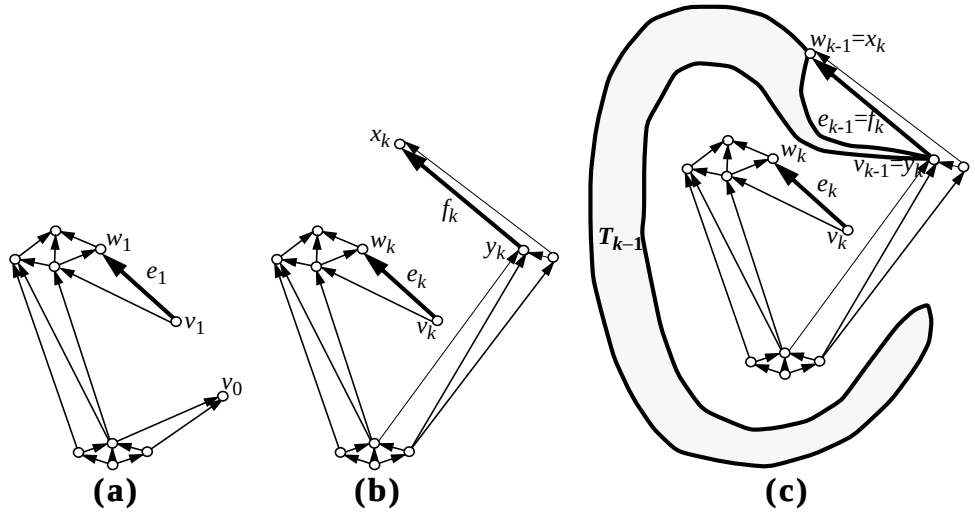


FIG. 2.1. (a) *Tendril* T_1*;* (b) *graph* H_k*;* (c) *constructing* T_k *from* T_{k-1}.

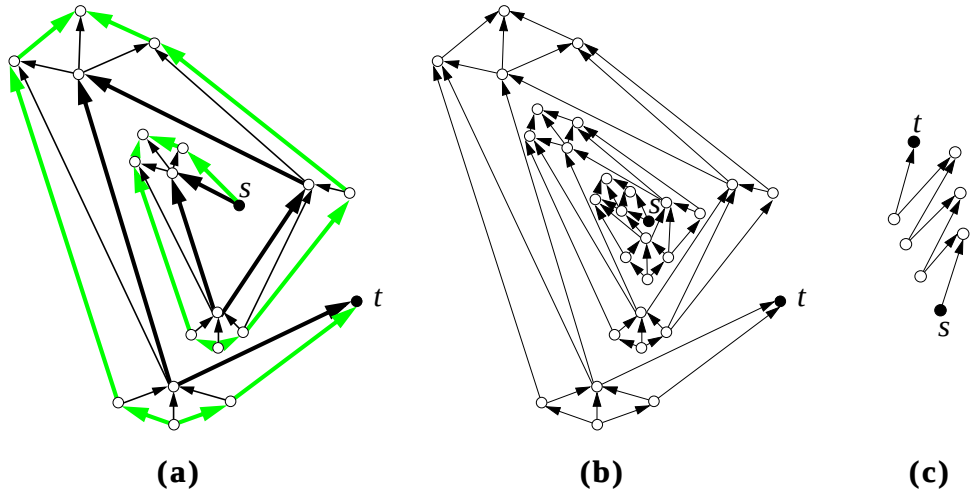


FIG. 2.2. *Examples of tendrils and wiggles:* (a) *tendril* T_2*;* (b) *tendril* T_3*;* (c) *wiggle* W_3*. The outer and inner paths of* T_2 *are drawn using shaded and solid thick lines, respectively.*

becomes a subpath of two faces f_1 and zz_2 in the new digraph. Given an upward embedding of W_k, we say that the *contribution* of W_k to face f_i, where $i = 1$ or 2, is equal to the number of large angles minus the number of small angles of W_k in f_i. Because each angle of W_k in f_i is either large or small, the contribution of W_k to f_i is an even number c, where $-2k \leq c \leq 2k$. Note that if W_k gives contribution c to face f_1 (f_2), it gives contribution $-c$ to face f_2 (f_1).

A *rectilinear tendril* T_k, for $k \geq 0$, is an undirected graph with two designated poles s and t, and is defined recursively as follows:

- T_0 consists of a single edge (s, t).
- T_1 is the 10-vertex graph with poles $s = s_1$ and $t = t_1$ shown in Figure 2.3(a).
- T_k is constructed from T_{k-1} by removing s_{k-1} and its incident edge and

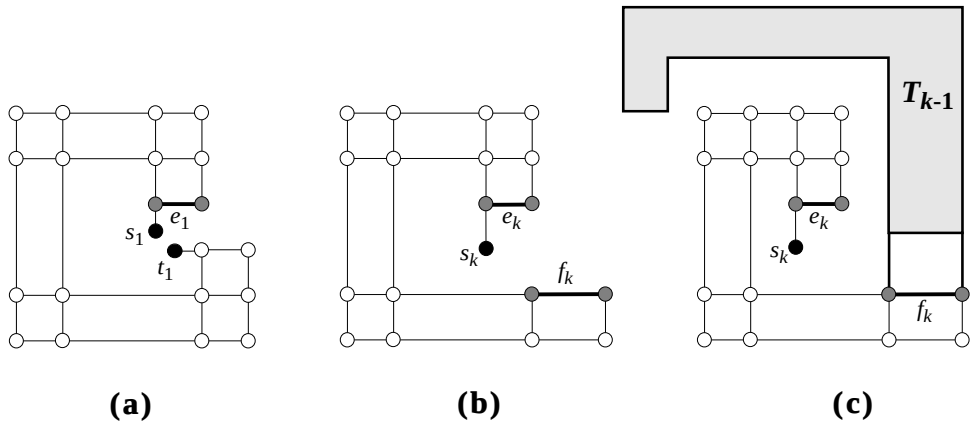


FIG. 2.3. (a) *Rectilinear tendril* T_1*;* (b) *graph* H_k*;* (c) *constructing* T_k *from* T_{k-1}.

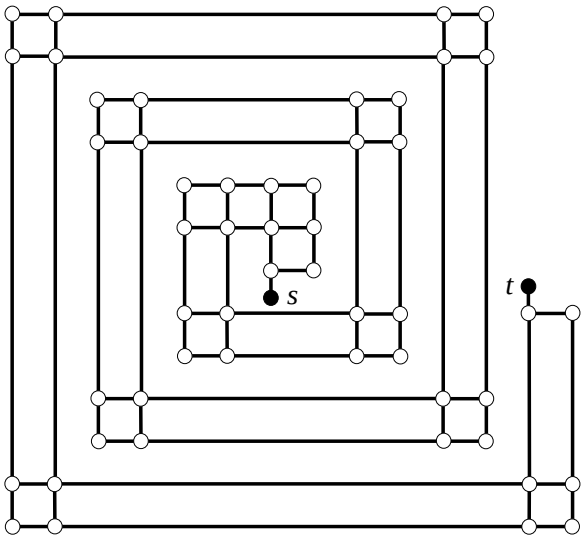


FIG. 2.4. *Rectilinear tendril* T_3.

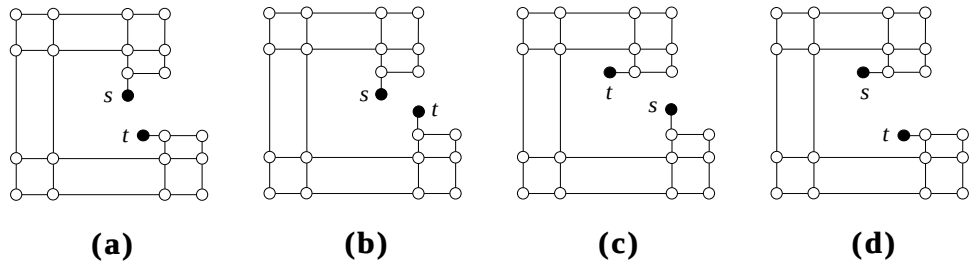


FIG. 2.5. *The four rectilinear embeddings of rectilinear tendril* T_1.

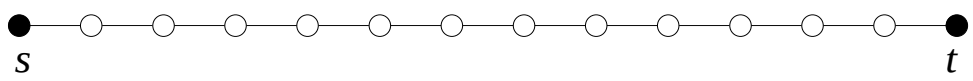


FIG. 2.6. *Rectilinear wiggle* W_3.

adding the graph H_k shown in Figure 2.3(b) to it by identifying its edge e_{k-1} with the edge f_k of H_k (Figure 2.3(c)). The poles of T_k are $s = s_k$ and $t = t_1$. Figure 2.4 shows rectilinear tendril T_3.

LEMMA 2.4. *Rectilinear tendril T_k is rectilinear planar and admits exactly four rectilinear embeddings.*

Proof. The proof has the same flavor as that of the proof of Lemma 2.3. We use induction on k and apply Lemma 2.2. □

Figure 2.5 shows the four rectilinear embeddings of T_1. An *external* path of T_k is a subpath from s to t of its external face in a rectilinear planar embedding of T_k. Notice that T_k has exactly two external paths.

A *rectilinear wiggle* W_k consists of a chain of $4k+1$ edges, and its two end vertices are called the *poles*. Figure 2.6 shows rectilinear wiggle W_3.

Later, in section 5, we will consider transformations where an edge (u, v) of an embedded graph is replaced with a rectilinear tendril T_k or wiggle W_k such that u and v are identified with the poles of T_k or W_k. Let f_1 and f_2 be the two faces of the new graph such that W_k or the external paths of T_k are subpaths of f_1 and f_2. The *contribution* of W_k or T_k to f_1 (f_2) is equal to the number of its angles labeled 3 minus the number of its angles labeled 1 that are in f_1 (f_2). Hence, the contribution of T_k to f_1 (f_2) is an integer c, where c is equal to one of $-(4k+2)$, $-(4k+1)$, $-4k$, $4k$, $4k+1$, and $4k+2$, and the contribution to f_2 (f_1) is $-c$. The contribution of W_k to f_1 (f_2) is an integer c, where $-4k \leq c \leq 4k$, and the contribution to f_2 (f_1) is $-c$.

3. An auxiliary flow problem. In this section, we define two auxiliary flow problems and show that they are equivalent to NOT-ALL-EQUAL-3-SAT under polynomial-time reductions.

A *switch-flow network* is an undirected flow network $\mathcal{N}$, where each edge is labeled with a range $[c' \cdots c'']$ of nonnegative integer values, called the *capacity range* of the edge. For simplicity, we denote the capacity range $[c \cdots c]$ with $[c]$. A *flow* for a switch-flow network is an orientation of and an assignment of integer "flow" values to the edges of the network. A *feasible flow* is a flow that satisfies the following two properties:

Range property. The flow assigned to an edge is an integer within the capacity range of the edge.

Conservation property. The total flow entering a vertex from the incoming edges is equal to the total flow exiting the vertex from the outgoing edges.

Starting from an instance $\mathcal{S}$ of NOT-ALL-EQUAL-3-SAT, we construct a switch-flow network $\mathcal{N}$ as follows (see Figures 3.1–3.2). Let the literals of $\mathcal{S}$ be denoted with $x_1, y_1, \ldots, x_n, y_n$, where $y_i = \overline{x_i}$, and let the clauses of $\mathcal{S}$ be denoted with $c_1, \ldots, c_m$. Let θ be a positive integer parameter. We denote with α_i and β_i (where $i = 1, \ldots, n$) the number of occurrences of literals x_i and y_i, respectively, in the clauses of $\mathcal{S}$. Note that $\sum_{i=1}^{n}(\alpha_i + \beta_i) = 3m$. Also, we define $\gamma_i = (2i-1)\theta$ and $\delta_i = 2i\theta$ ($i = 1, \ldots, n$). Network $\mathcal{N}$ has a *literal vertex* for each literal of $\mathcal{S}$ and a *clause vertex* for each clause of $\mathcal{S}$, plus a special dummy vertex z. There are three types of edges in $\mathcal{N}$ (see Figure 3.2):

Literal edges. Joining pairs of literals associated with the same boolean variable; the capacity range of literal edge (x_i, y_i) is $[\alpha_i\gamma_i + \beta_i\delta_i]$.

Clause edges. Joining each literal to each clause; the capacity range of clause edge (x_i, c_j) is $[\gamma_i]$ if $x_i \in c_j$ and $[0]$ otherwise. The capacity range of clause edge (y_i, c_j) is $[\delta_i]$ if $y_i \in c_j$ and $[0]$ otherwise.

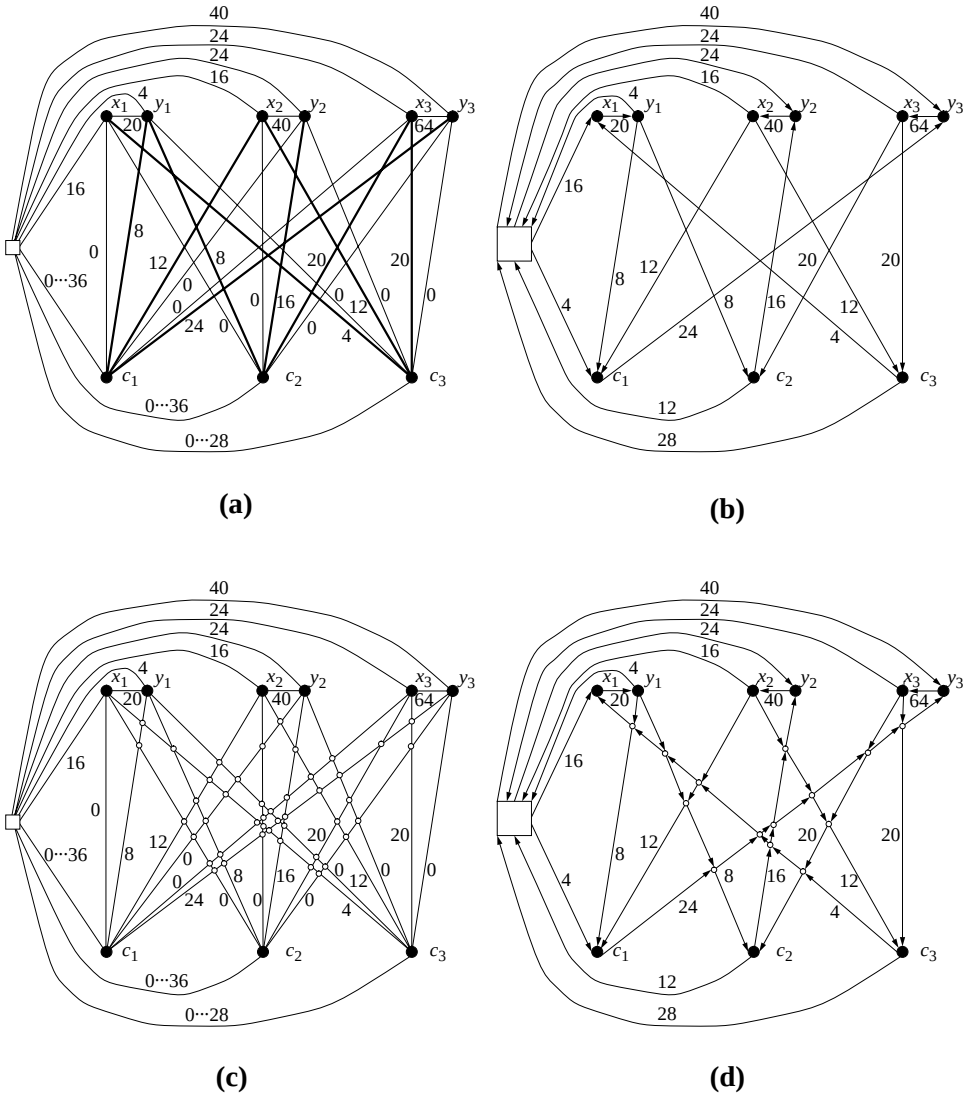


FIG. 3.1. (a) *Switch-flow network $\mathcal{N}$ with parameter $\theta = 4$ associated with the* NOT-ALL-EQUAL-3-SAT *instance $\mathcal{S}$ with clauses $c_1 = y_1x_2y_3$, $c_2 = y_1y_2x_3$, and $c_3 = x_1x_2x_3$. The clause edges with nonzero capacity range are shown with thick lines.* (b) *Feasible flow for $\mathcal{N}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown.* (c) *Planar switch-flow network $\mathcal{P}$ associated with $\mathcal{S}$.* (d) *Feasible flow for $\mathcal{P}$ corresponding to the satisfying truth assignment (y_1, x_2, x_3) for $\mathcal{S}$. Only the edges with nonzero flow are shown.*

Dummy edges. Joining each literal and each clause to the dummy vertex; the capacity ranges of dummy edges (z, x_i) and (z, y_i) are $[\beta_i\delta_i]$ and $[\alpha_i\gamma_i]$, respectively. The capacity range of dummy edge (z, c_j) is $[0 \cdots \eta_j - 2\theta]$, where η_j is the sum of the capacities of the clause edges incident on c_j.

The construction of network $\mathcal{N}$ from $\mathcal{S}$ is straightforward, and we have the following lemma.

LEMMA 3.1. *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses, the associated switch-flow network $\mathcal{N}$ has $O(n + m)$ vertices and $O(nm)$*

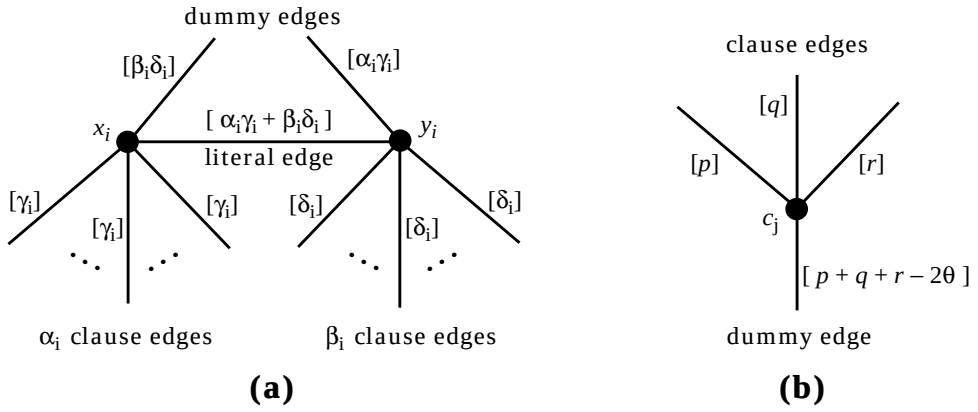


FIG. 3.2. *Schematic illustration of the edges incident on the literal and clause vertices of network N:* (a) *literal vertices x_i and y_i;* (b) *clause vertex c_j.*

edges and can be constructed in $O(nm)$ time.

A feasible flow in network $\mathcal{N}$ corresponds to a satisfying truth assignment for $\mathcal{S}$. Namely, we have that a literal is true whenever its incident literal edge is incoming in the feasible flow (see Figure 3.1(b)) and its incident clause edges with nonzero capacity range are outgoing. We formalize this correspondence in the following lemma.

LEMMA 3.2. *An instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *is satisfiable if and only if the associated switch-flow network $\mathcal{N}$ admits a feasible flow. Also, given a feasible flow for $\mathcal{N}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(nm)$, where n and m are the number of variables and clauses of $\mathcal{S}$, respectively.*

Proof. If. Given a feasible flow in $\mathcal{N}$, we construct a truth assignment A by setting a literal *true* if its incident literal edge is incoming in the flow. We now show that this is a satisfying assignment. Clearly, the two literals x_i and y_i associated with the same boolean variable consistently receive opposite truth assignments.

Because of the conservation property, all the incident clause edges with nonzero capacity range of a *true* literal are outgoing, and the amount of flow in each of them is equal to the capacity. Conversely, if a literal is *false*, then because of the conservation property, all its incident clause edges with nonzero capacity range are incoming, and the amount of flow in each of them is equal to the capacity. The three clause edges with nonzero capacity range incident on a clause vertex c_j cannot be all incoming or all outgoing because of the conservation property at vertex c_j and the choice of the capacity range for the dummy edge incident on c_j. Therefore, three literals in c_j can not be all *true* or all *false*. Hence, A is a satisfying truth assignment for $\mathcal{S}$. Also, A can be constructed in time linear in the number of edges of $\mathcal{N}$, which is $O(nm)$.

Only if. Let A be a satisfying truth assignment of $\mathcal{S}$. We construct a feasible flow f in $\mathcal{N}$ from A. In this flow, we have the following:

- The amount of flow through the literal, clause, and the dummy edges incident on literal vertices is equal to their capacities. The flow, therefore, satisfies the range property in these edges.
- If a literal is true, then its incident literal edge is incoming and its incident clause edges with nonzero capacity range and its dummy edge are outgoing. If a literal is false then its incident literal edge is outgoing and its incident clause edges with nonzero capacity and its dummy edge are incoming. In either case, the amount of flow coming into a literal l_i is $\alpha_i\gamma_i + \beta_i\delta_i$, and

the amount of flow leaving it is $\alpha_i\gamma_i + \beta_i\delta_i$. Therefore, the flow satisfies the conservation property at these vertices.

- If the net flow entering a clause vertex through the clause edges is ν_j, then the flow leaving it through its dummy edge is ν_j. Conversely, if the net flow leaving a clause through the clause edges is ν_j, then the flow entering it through its dummy edge is ν_j. Since A is a satisfying assignment, the three clause edges with nonzero capacity range incident on a clause vertex cannot be all incoming or all outgoing. The magnitude of flow in a clause edge is at least θ. Therefore, the net flow coming into or leaving a clause vertex c_j through its clause edges is at most $\eta_j - 2\theta$. Hence, the conservation property at c_j and the range property in its dummy edge are both satisfied by the flow.
- We show now that the conservation property holds at the dummy vertex by using the following argument. By successive mergers of two nondummy vertices of $\mathcal{N}$ and elimination of the resultant self loops and the flow through them, we can reduce $\mathcal{N}$ into a graph with multiple edges between two vertices: the dummy vertex and another vertex u for which the conservation property holds. The net flow between these two vertices is same as in f. Since the conservation property holds for u and the dummy vertex is neither a source nor a sink, the conservation property holds for the dummy vertex also. □

Now, starting from $\mathcal{S}$, we construct a planar switch-flow network $\mathcal{P}$ (see Figure 3.1). We first construct a *layered drawing* $\psi_{\mathcal{N}}$ of $\mathcal{N}$ as follows (see Figure 3.1(a)):

- Each literal and clause edge is drawn as a straight line. The dummy edges are drawn as continuous curves.
- The clause vertices are horizontally aligned and ordered $c_1, c_2, \ldots, c_m$ from left to right.
- The literal vertices are horizontally aligned above the clause vertices and ordered $x_1, x_2, y_1, y_2, \ldots, x_m, y_m$ from left to right.
- There are crossings only between the clause edges. However, no more than two clause edges cross at the same point.

We next replace the crossings of $\psi_{\mathcal{N}}$ with vertices called the *crossing* vertices, thus splitting the clause edges at the crossing vertices. We call *fragment edges*, or simply *fragments*, the edges originated by the splitting of the clause edges. Each fragment edge inherits the capacity range of the originating clause edge. We define the *facial degree* of a vertex as the total number of edges in its incident faces.

LEMMA 3.3. *Given network $\mathcal{N}$ representing an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with $n \geq 3$ variables and $m \geq 3$ clauses, the associated planar switch-flow network $\mathcal{P}$ is triconnected, has $O(n^2m^2)$ vertices and edges, and can be constructed in $O(n^2m^2)$ time. Also, in the unique embedding of $\mathcal{P}$, the facial degree of each vertex is at most $7nm$.*

Proof. Let $\psi_{\mathcal{N}}$ be the layered drawing of $\mathcal{N}$ that is used to construct $\mathcal{P}$ (see Figure 3.1(a)). There are $O(n^2m^2)$ crossings in $\psi_{\mathcal{N}}$. Therefore, $\mathcal{P}$ has $O(n^2m^2)$ crossing vertices and fragment edges. Consequently, $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges.

Now we show that $\mathcal{P}$ is triconnected. We denote by u_k the crossing vertex that corresponds to the crossing between the line joining c_k and y_n and the line joining c_{k+1} and x_1. We denote by v_k the crossing vertex that corresponds to the crossing between the line joining y_k and c_m and the line joining x_{k+1} and c_1. We call the vertices of type u_k and v_k the *bounding* crossing vertices of $\mathcal{P}$. No two bounding vertices are identical because they correspond to crossings between different pairs of

lines. Hence there are two vertex disjoint paths, $p = x_1 y_1 v_1 \dots x_k y_k v_k x_{k+1} \dots y_n$ and $q = c_1 u_1 c_2 \dots c_k u_k c_{k+1} \dots c_m$, in $\mathcal{P}$. Let a and b be two nondummy vertices of $\mathcal{P}$. From a and b, there are four vertex-disjoint paths p_a, q_a, p_b, and q_b such that p_a and q_a consist of the fragment edges of a clause edge incident upon a and connect a with a vertex of the paths p and q, respectively, and p_b and q_b consist of the fragment edges of a clause edge incident upon b and connect b with a vertex of the paths p and q, respectively. Notice that if a (b) is a literal vertex, then p_a (p_b) is empty; and if a (b) is a clause vertex, then q_a (q_b) is empty. Hence, between a and b there are two vertex-disjoint paths: one consisting of p_a, a subpath of p, and p_b, and the other consisting of q_a, a subpath of q, and q_b. Hence, graph $\mathcal{P} - \{z\}$ is biconnected. Now suppose, for contradiction, that $\mathcal{P}$ is not triconnected. Since the graph $\mathcal{P} - \{z\}$ is biconnected, z is not a member of a separating pair of $\mathcal{P}$. Let C_1 and C_2 be two connected components of $\mathcal{P}$ obtained after removing a separating pair. Suppose z is in C_1. Hence there are at most two vertex disjoint paths between z and the vertices in C_2. However, between z and a vertex w of $\mathcal{P} - \{z\}$, there are at least four vertex disjoint paths as follows:

- if w is a crossing vertex, then the four paths go through the endpoints of the two clause edges of $\mathcal{N}$ whose crossing is associated with v;
- if w is a clause vertex, then one path consists of the dummy edge incident upon w, and the other three paths go through three literal vertices; and
- if w is a literal vertex, then one path consists of the dummy edge incident upon w, and the other three paths go through three clause vertices.

Thus, we get a contradiction. Therefore $\mathcal{P}$ is triconnected.

Now we show that the facial degree of any vertex is at most $7nm$. All the faces incident on the dummy vertex z have at most four vertices: one is z, two of them are clause and/or literal vertices, and the fourth (if present) is a bounding crossing vertex. There are $n - 1 + m - 1$ bounding crossing vertices in $\mathcal{P}$. A simple counting argument, therefore, shows that the facial degree of z is equal to $3n + m + 2(n - 1 + m - 1) + 2 = 5n + 3m - 2$. Let f be a face that does not contain z. Face f contains at most two fragments of clause edges incident on the same clause or literal vertex. Hence, the number of edges in face f is at most $\min\{2 \cdot 2n, 2m\} = \min\{4n, 2m\}$. The degree of a nondummy vertex u is at most $\max\{2n + 1, m + 2, 4\}$, which is at most $\max\{2n + 1, m + 2\}$ for $n, m \geq 3$. Hence, the facial degree of u is at most $\max\{2n+1, m+2\} \cdot \min\{4n, 2m\} \leq (\max\{2n, m\}+2) \cdot \min\{4n, 2m\} = (\max\{2n, m\}+2) \cdot 2\min\{2n, m\} = 2(\max\{2n, m\} + 2)\min\{2n, m\} = 2\max\{2n, m\}\min\{2n, m\} + 4\min\{2n, m\} = 2(2nm) + 4\min\{2n, m\} = 4nm + \min\{8n, 4m\}$. Since $\min\{8n, 4m\}$ is at most $3nm$ for $n, m \geq 3$, we have that the facial degree of u is at most $4nm + 3nm = 7nm$. Since the facial degree of the dummy vertex is equal to $5n + 3m - 2$, we have that the facial degree of a vertex of $\mathcal{P}$ is at most $\max\{7nm, 5n + 3m - 2\}$, which is equal to $7nm$ for $n, m \geq 3$.

Finally, we show how to construct $\mathcal{P}$ from $\mathcal{N}$ in $O(n^2m^2)$ time. Suppose the literals are numbered from 1 to $2n$ so that literal $l_{2k-1} = x_k$ and $l_{2k} = y_k$. We inductively construct a layered drawing $\psi(k, m)$ of the subgraph of $\mathcal{N}$ induced by literals $l_1, \dots, l_k$ and clauses $c_1, \dots, c_m$ (see Figure 3.3). Drawing $\psi(2, m)$ is shown in Figure 3.3(a). Suppose we have already constructed drawing $\psi(k - 1, m)$ (see Figure 3.3(b)). We place literal l_k at the same height as l_{k-1} and sufficiently far to its right so that edge (l_k, c_1) intersects each clause edge of $\psi(k - 1, m)$ below its lowest crossing. Replacing the crossings of $\psi(2n, m)$ by crossing vertices gives us the planar network $\mathcal{P}$.

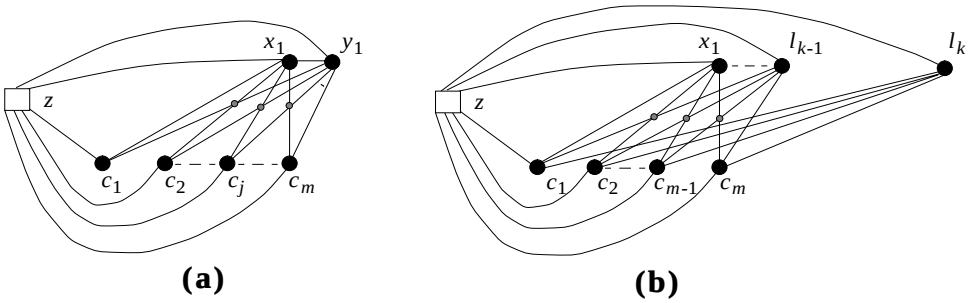


FIG. 3.3. *Proof of Lemma* 3.3: (a) *drawing* $\psi(2, m)$*;* (b) *drawing* $\psi(k, m)$*. The small circles show the lowest crossings of the clause edges in the drawing of* $\psi(k-1, m)$*.*

The construction of the network $\mathcal{P}$ does not require computing the exact coordinates of the vertices and crossings of $\psi(2n, m)$. In an actual implementation, $\mathcal{P}$ can be constructed directly by maintaining and updating at each inductive step a list of the lowest crossing vertices of the clause edges. The manipulation of this list takes constant time per crossing vertex and hence can be done in total $O(n^2m^2)$ time. The rest of the construction can also be carried out in $O(n^2m^2)$ time. □

LEMMA 3.4. *Network* $\mathcal{N}$ *admits a feasible flow if and only if network* $\mathcal{P}$ *admits a feasible flow, and a feasible flow for* $\mathcal{N}$ *can be computed from a feasible flow for* $\mathcal{P}$ *in* $O(n^2m^2)$ *time.*

Proof. Only if. Given a feasible flow in $\mathcal{N}$, we can get a feasible flow in $\mathcal{P}$ in which the flow through each fragment edge is the same as in the corresponding clause edge in $\mathcal{N}$, and the flow through the dummy and literal edges is same as in the corresponding edges in $\mathcal{N}$.

If. Suppose that $\mathcal{P}$ admits a feasible flow f. In f, at any crossing vertex, of the two fragment edges of a clause edge incident upon it, one is incoming and the other is outgoing. This is so because the fragment edges of different clause edges have different capacities and the conservation property is satisfied at the vertex. Consequently, all the fragment edges of a clause edge have the same flow. We can get a feasible flow in $\mathcal{N}$ in which the flow through a clause edge is same the as the flow in its fragment edges in f, and the flow through the dummy edges and the literal edges is the same as the flow in the corresponding edges in $\mathcal{P}$. □

By combining Lemmas 3.2, 3.3, and 3.4, we obtain the main result of this section.

THEOREM 3.5. *Given an instance* $\mathcal{S}$ *of* NOT-ALL-EQUAL-3-SAT *with* $n \geq 3$ *variables and* $m \geq 3$ *clauses, the associated planar switch-flow network* $\mathcal{P}$ *is triconnected, has* $O(n^2m^2)$ *vertices and edges, has facial degree at most* $7nm$*, and can be constructed in* $O(n^2m^2)$ *time. Instance* $\mathcal{S}$ *is satisfiable if and only if network* $\mathcal{P}$ *admits a feasible flow. Also, given a feasible flow for* $\mathcal{P}$*, a satisfying truth assignment for* $\mathcal{S}$ *can be computed in time* $O(n^2m^2)$*.*

4. Upward planarity testing. In this section, we show how to reduce the problem of computing a feasible flow in the planar switch-flow network associated with a NOT-ALL-EQUAL-3-SAT instance to the problem of testing the upward planarity of a suitable digraph.

Let $\mathcal{P}$ be the planar switch-flow network with parameter $\theta = 4$ associated with a NOT-ALL-EQUAL-3-SAT instance $\mathcal{S}$. Now we construct an orientation $\vec{\mathcal{P}}$ of $\mathcal{P}$ as follows (see Figure 4.1):

- Every literal edge (x_i, y_i) is oriented from x_i to y_i.

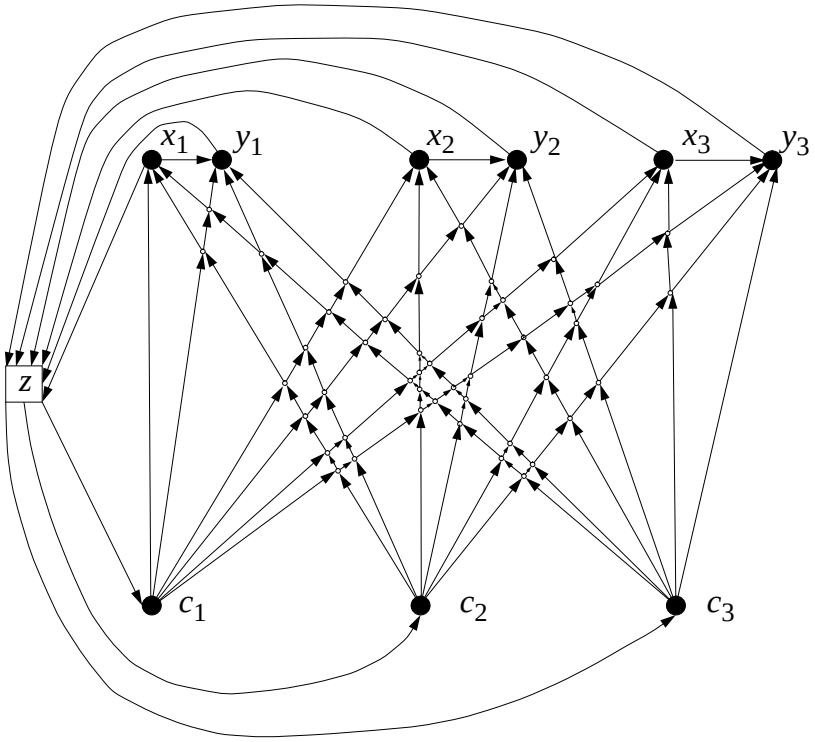


FIG. 4.1. *Orientation $\vec{\mathcal{P}}$ of the network $\mathcal{P}$ shown in Figure* 3.1(c).

- Every fragment edge is oriented "away from" the clause vertex and "towards" the literal vertex.
- Every dummy edge incident upon a literal vertex is oriented towards the dummy vertex, and every dummy edge incident upon a clause vertex is oriented towards the clause vertex.

LEMMA 4.1. *In digraph $\vec{\mathcal{P}}$, every vertex has at least one incoming and one outgoing edge, every directed cycle contains the dummy vertex, and there are exactly two faces that are directed cycles. Also, $\vec{\mathcal{P}}$ is bimodal and each face of $\vec{\mathcal{P}}$ consists of at most two directed paths.*

Proof. Let m be the number of clauses and n be the number of variables in $\mathcal{S}$. Let z be the dummy vertex of $\vec{\mathcal{P}}$.

Each crossing vertex has two incoming and two outgoing fragment edges. Each literal vertex y_i has $m+1$ incoming edges and one outgoing edge (to z). Each literal vertex x_i has m incoming edges and two outgoing edges (to z and y_i). Each clause vertex has $2n$ outgoing edges and one incoming edge (from z). The dummy vertex has $2n$ incoming edges and m outgoing edges. Therefore, each vertex has at least one incoming and one outgoing edge. It can easily be verified that each vertex of $\vec{\mathcal{P}}$ is bimodal, and hence $\vec{\mathcal{P}}$ is bimodal.

The digraph $\vec{\mathcal{P}} - \{z\}$ is upward planar with m sources, each being a clause vertex, and n sinks, each being a literal vertex y_i. Therefore, every directed cycle in $\vec{\mathcal{P}}$ contains z. There are exactly two faces in $\vec{\mathcal{P}}$ that that are directed cycles: one consisting of vertices x_1, z, and c_1, and the other consisting of vertices y_n, z, and c_m. It can be easily verified that the other faces of $\vec{\mathcal{P}}$ each consist of exactly two directed

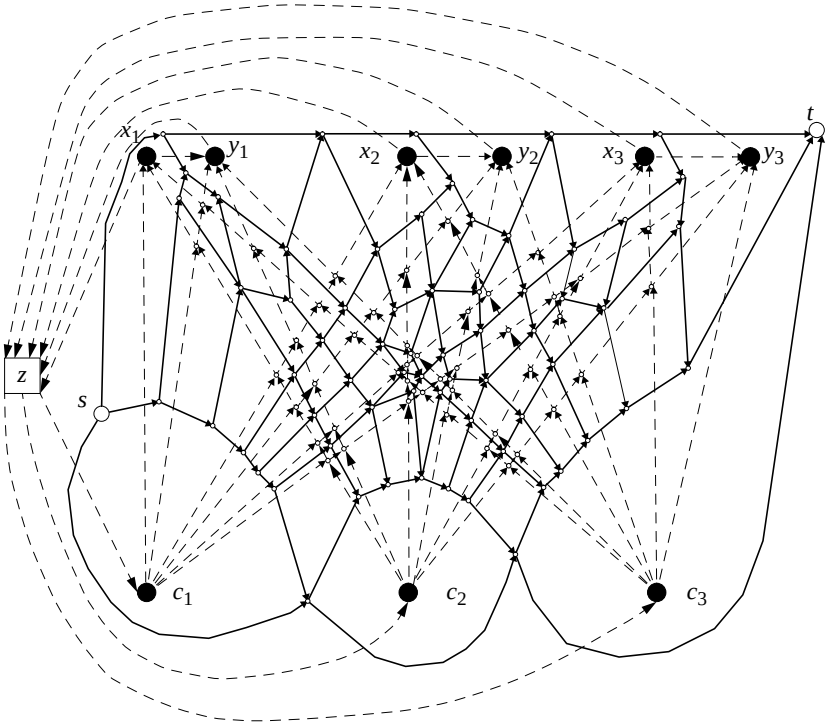


FIG. 4.2. *Dual digraph $\vec{\mathcal{D}}$ of the network $\vec{\mathcal{P}}$ shown in Figure* 4.1.

paths. □

Since $\mathcal{P}$ is triconnected (see Theorem 3.5), the planar embedding of $\mathcal{P}$ and the dual graph of $\mathcal{P}$ are unique. We construct the dual digraph $\vec{\mathcal{D}}$ of $\vec{\mathcal{P}}$ by taking the dual graph $\mathcal{D}$ of $\mathcal{P}$ and orienting every dual edge from the face on the left to the face on the right of the primal edge (see Figure 4.2).

LEMMA 4.2. *The dual digraph $\vec{\mathcal{D}}$ of $\vec{\mathcal{P}}$ is upward planar, triconnected, acyclic, and has exactly one source and one sink, denoted with s and t, both of which are on the same face. Also, each face of $\vec{\mathcal{D}}$ has exactly one source and one sink.*

Proof. Because $\vec{\mathcal{P}}$ is planar and triconnected (Theorem 3.5), so is its dual $\vec{\mathcal{D}}$. By Lemma 4.1, exactly two faces of $\vec{\mathcal{P}}$ are directed cycles. Hence, $\vec{\mathcal{D}}$ has exactly two switches, denoted by s and t (see Figure 4.2), respectively, corresponding to these two cycles. Switch s is a source vertex and corresponds to the face of $\vec{\mathcal{P}}$ that consists of the literal vertex x_1, dummy vertex z, and the clause vertex c_1. Switch t is a sink vertex and corresponds to the face of $\vec{\mathcal{P}}$ that consists of the vertices y_n, z, and c_m. Also notice that both s and t are on the face that is the dual of z. From Lemma 4.1, each face of $\vec{\mathcal{P}}$ consists of at most two directed paths, and hence each vertex of $\vec{\mathcal{D}}$ is bimodal. Therefore, $\vec{\mathcal{D}}$ is also bimodal. Again from Lemma 4.1, each vertex of $\vec{\mathcal{P}}$ is bimodal and none of them is a source or a sink. Hence, each face of $\vec{\mathcal{D}}$ has exactly one source and one sink. Since s and t are the only switches of $\vec{\mathcal{D}}$ and both of them are on the same face, it follows that there is a consistent assignment of labels to the angles of $\vec{\mathcal{D}}$ in which exactly two angles are labeled large, namely, the angles at s and t in their common face. Therefore, by Lemma 2.1, $\vec{\mathcal{D}}$ has an upward embedding and hence is upward planar. □

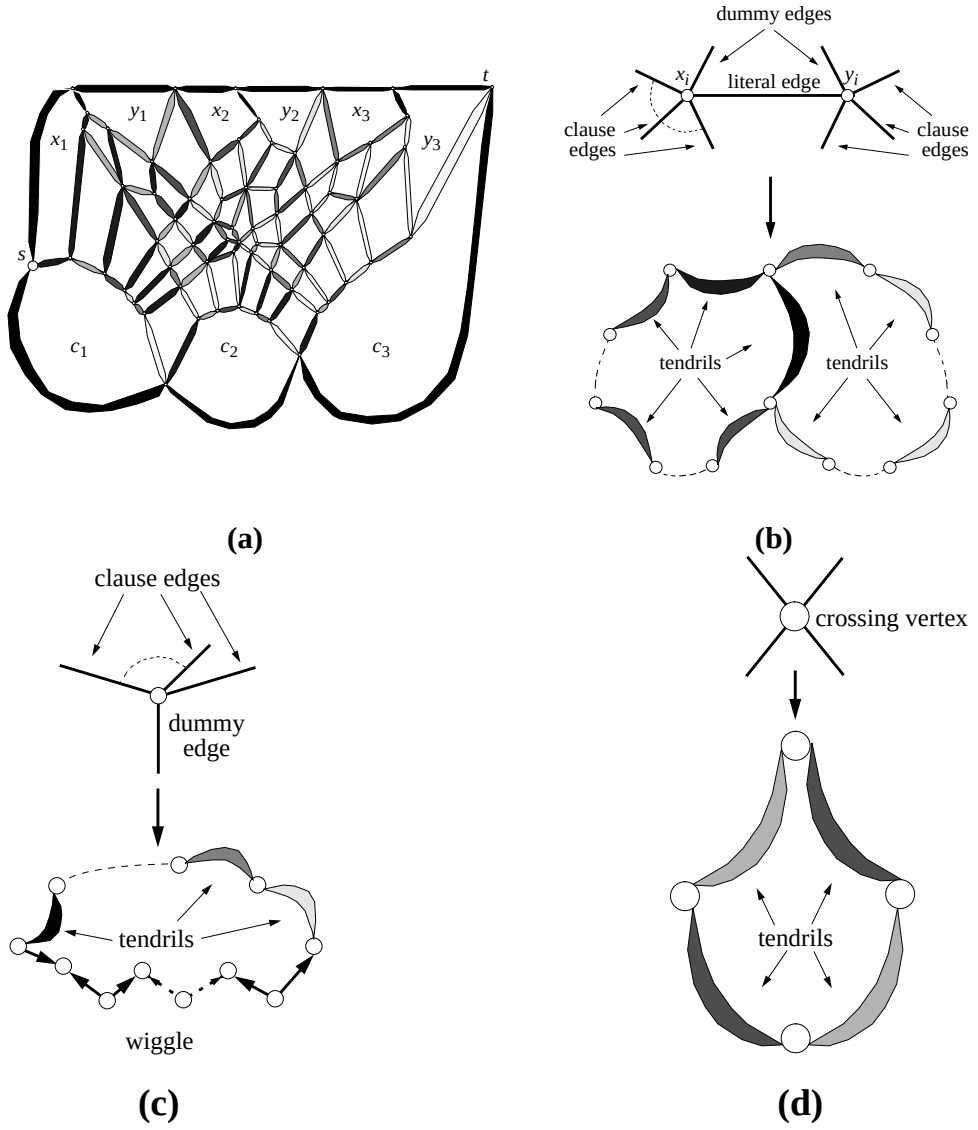


FIG. 4.3. (a) *Schematic illustration of digraph* $\vec{\mathcal{G}}$ *obtained from* $\vec{\mathcal{D}}$ *by replacing edges with tendrils and wiggles.* (b) *The two faces of* $\vec{\mathcal{G}}$ *associated with literal vertices* x_i *and* y_i *of* $\mathcal{P}$. (c) *The face of* $\vec{\mathcal{G}}$ *associated with a clause vertex of* $\mathcal{P}$. (d) *The face of* $\vec{\mathcal{G}}$ *associated with a crossing vertex of* $\mathcal{P}$.

Starting from digraph $\vec{\mathcal{D}}$, we construct a new digraph $\vec{\mathcal{G}}$ by replacing the edges of $\vec{\mathcal{D}}$ with subgraphs (tendrils or wiggles) as follows (see Figure 4.3):

- Every edge of $\vec{\mathcal{D}}$ that is the dual of a literal edge, fragment edge, or dummy edge incident on a literal vertex is replaced with tendril T_c, where $[c]$ is the capacity range of the dual edge. Notice that c is a multiple of parameter θ.
- Every edge of $\vec{\mathcal{D}}$ that is the dual of a dummy edge incident on a clause vertex is replaced with wiggle W_c, where $[0 \cdots c]$ is the capacity range of the dual edge.

A vertex of $\vec{\mathcal{G}}$ is a *primary* vertex if it is also a vertex of $\vec{\mathcal{D}}$, and is a *secondary* vertex otherwise. A face of an embedding of $\vec{\mathcal{G}}$ is a *secondary* face if it is bounded by the edges of the same tendril, and is a *primary* face otherwise. Figure 4.3(a) shows the primary faces of an embedding ψ of $\vec{\mathcal{G}}$ and the primary vertices of $\vec{\mathcal{G}}$; in this figure, the secondary faces of ψ and the secondary vertices of $\vec{\mathcal{G}}$ are hidden inside the shaded regions denoting the tendrils and the wiggles. We establish the following correspondence between the primary faces of an embedding ψ of $\vec{\mathcal{G}}$, the faces of $\vec{\mathcal{D}}$, and the vertices of $\vec{\mathcal{P}}$. Let f be a primary face of ψ; f corresponds to a face f' of $\vec{\mathcal{D}}$, namely, the one whose boundary edges were replaced by tendrils and wiggles to give f. Recall that $\vec{\mathcal{D}}$ is the dual digraph of $\vec{\mathcal{P}}$, and its faces correspond to the vertices of $\vec{\mathcal{P}}$. Therefore, f also corresponds to a vertex v of $\vec{\mathcal{P}}$, namely, the one that corresponds to f'. f is the *dummy face* of $\vec{\mathcal{G}}$ if v is the dummy vertex of $\vec{\mathcal{P}}$, and is a *nondummy* face otherwise. A *primary* angle a of f is its angle at a primary vertex u; a corresponds to an angle of f', namely, the one at u. A primary vertex of f is a *primary source* of f if it is also a source of f, and is a *primary sink* of f if it is also a sink of f.

There is no directed path from the sink pole to the source pole of a tendril or a wiggle. Therefore, $\vec{\mathcal{G}}$ is also acyclic. From Lemma 4.2 and the construction of digraph $\vec{\mathcal{G}}$, the following lemma is immediate.

LEMMA 4.3. *Digraph $\vec{\mathcal{G}}$ is planar and acyclic. All its embeddings can be obtained by choosing one of the two possible flips for each tendril and have the same set of secondary faces. Also, for every face f in an embedding of $\vec{\mathcal{G}}$, the following holds:*

- *if f is a secondary face, then it has exactly one source and exactly one sink;*
- *or else (f is a primary face), it has exactly one primary source and exactly one primary sink.*

We need the following technical lemma to prove Lemma 4.5.

LEMMA 4.4. *Let f be a primary face of an upward embedding ψ of $\vec{\mathcal{G}}$. Let $\tau(f)$ and $\omega(f)$ be the total contribution to f of its tendrils and wiggles, respectively. Then,*

$$|\tau(f) + \omega(f)| \leq \theta.$$

Proof. From Lemma 4.3, f has exactly one primary source s and exactly one primary sink t. In other words, f has exactly two primary angles. Let $\nu(f)$ be equal to the number of *large* primary angles minus the number of *small* primary angles of f. Clearly, $|\nu(f)| \leq 2$. Let us denote with $L(f)$ and $S(f)$, respectively, the number of *large* and *small* angles of f. Because ψ is an upward embedding, from Lemma 2.1 it follows that $|L(f) - S(f)| = 2$. An angle of f is either a primary angle of f or an angle of one of its tendrils and wiggles. Therefore, $L(f) - S(f) = \nu(f) + \tau(f) + \omega(f)$. Therefore, $\tau(f) + \omega(f) = L(f) - S(f) - \nu(f)$. Hence, $|\tau(f) + \omega(f)| = |L(f) - S(f) - \nu(f)| \leq |L(f) - S(f)| + |\nu(f)|$. Since $|\nu(f)| \leq 2$ and $|L(f) - S(f)| = 2$, we have that $|\tau(f) + \omega(f)| \leq 4$. Recall from the beginning of this section that $\theta = 4$. Hence, $|\tau(f) + \omega(f)| \leq \theta$. □

We are now ready to present Lemma 4.5.

LEMMA 4.5. *Digraph $\vec{\mathcal{G}}$ is upward planar if and only if its tendrils can be flipped and labels can be assigned to the angles of its wiggles such that for every primary face the total contribution to it of its tendrils and wiggles is zero.*

Proof. If. From Lemma 4.2, $\vec{\mathcal{D}}$ has an upward embedding $\psi_{\vec{\mathcal{D}}}$. Let g' be the external face of $\psi_{\vec{\mathcal{D}}}$. Let ψ be a labeled embedding of $\vec{\mathcal{G}}$ such that, for every face f of ψ,

- if f is a secondary face, then the angles of f are labeled *small*;
- or else (f is a primary face)

– the total contribution to it of its tendrils and wiggles is zero, and
– its primary angles have the same label as the corresponding angles of $\psi_{\vec{\mathcal{D}}}$.

Notice that such a ψ exists because the tendrils of $\vec{\mathcal{G}}$ can be flipped and labels can be assigned to the angles of the wiggles of $\vec{\mathcal{G}}$ such that for every primary face the total contribution to it of its tendrils and wiggles is zero.

Let g be the primary face of ψ that corresponds to g'. We now show that ψ is an upward embedding with g as its external face. From Lemma 2.1, this is equivalent to showing that each face of ψ is consistently assigned with g as the external face (see the definition of consistently assigned faces in section 2.1).

Let f be a face of ψ. If f is a secondary face, then, from Lemma 4.3, f has exactly one source and exactly one sink. Because both of them are labeled *small* and f is an internal face, f is consistently assigned.

If f is a primary face, then it corresponds to a face f' of $\psi_{\vec{\mathcal{D}}}$. Let d be an integer equal to the number of large angles minus the number of small angles of f. Define d' likewise for f'. Because the total contribution to f of its tendril and wiggles is zero, and the angles at its primary vertices have the same label as the corresponding angles of f', it follows that $d = d'$. Thus, since f' is consistently assigned, f is consistently assigned too.

Only if. Suppose $\vec{\mathcal{G}}$ has an upward embedding ψ. Let f be a face of ψ. Since ψ is an upward embedding, f is consistently assigned, and therefore, from Lemma 2.1, the difference of its large and small angles is 2. However, because f, in general, also has primary vertices, each one of which contributes a *large* or *small* angle to it, it may be possible that the total contribution to it of its tendrils and wiggles is not zero. However, using Lemma 4.5, we show now that $\vec{\mathcal{G}}$ admits a labeled embedding ψ' such that[1]

- ψ' has the same faces as ψ,
- the primary angles of ψ' are the same as those of ψ,
- the contribution of a tendril to a primary face f of ψ' is the same as its contribution to f in ψ,
- the contribution of a wiggle to a primary face f of ψ' *may be different* from its contribution to f in ψ, and
- for every primary face f of ψ', the total contribution to f of its tendrils and wiggles is zero.

(Thus, ψ' and ψ are the same except for a possible difference in the assignment of labels to the angles of their wiggles.)

Let f be a primary face of ψ (and therefore also of ψ'). Let $\tau(f)$ and $\omega(f)$ be the total contributions to f of its tendrils and wiggles, respectively, in ψ. Let $\tau'(f)$ and $\omega'(f)$ be the total contributions to f of its tendrils and wiggles, respectively, in ψ'. Our goal is to show that $\tau'(f) + \omega'(f) = 0$.

First of all, we notice that because ψ and ψ' have the same faces, $\tau(f) = \tau'(f)$.

We now describe the assignment of labels to the angles of the wiggles of ψ' and show that $\tau'(f) + \omega'(f) = 0$ with this assignment. We have the following three cases:

Case 1. f is a nondummy face and it corresponds to a literal vertex or a crossing vertex of $\vec{\mathcal{P}}$. Clearly, f has no wiggles. Therefore, $\omega'(f) = \omega(f) = 0$. Since $\omega(f) = 0$,

[1] By appropriately reassigning labels to the angles of those primary vertices that are endpoints of the wiggles, from ψ we can obtain an upward embedding in which for every primary face f the total contribution to f of its tendrils and wiggles is zero. However, for our purposes, it is sufficient to show the existence of a labeled embedding ψ', as described here.

from Lemma 4.4 it follows that $|\tau(f)| \leq \theta$. From the construction of graph $\vec{\mathcal{G}}$ and the fact that a tendril T_k gives a contribution equal to either $2k$ or $-2k$ to a face (see section 2.2), it follows that $|\tau(f)|$ is a multiple of 2θ. Therefore, $|\tau(f)| \leq \theta$ implies that $\tau(f)$ is equal to 0. Since $\tau'(f) = \tau(f)$, we have that $\tau'(f)$ is also equal to 0. Because $\omega'(f) = 0$, it follows that $\tau'(f) + \omega'(f) = 0$.

Case 2. f is a nondummy face and it corresponds to a clause vertex c_j of $\vec{\mathcal{P}}$. Clearly, f has exactly one wiggle w. In ψ', we assign labels to the angles of w such that $\omega'(f) = -\tau(f)$. As noted earlier, $\tau'(f) = \tau(f)$. Hence, $\tau'(f)+\omega'(f) = \tau(f)-\tau(f) = 0$ with this assignment of labels to w. Therefore, if we can show that it is possible to assign labels to the angles of w such that $\omega'(f) = -\tau(f)$, we are done.

Recall from the construction of network $\mathcal{N}$ from section 3 that the capacity range of the dummy edge incident on c_j is $[0 \cdots \eta_j - 2\theta]$, where η_j is the sum of the capacities of the clause edges incident on c_j. Also it follows from the construction of $\vec{\mathcal{G}}$ that w is a copy of the wiggle $w_{\eta_j - 2\theta}$. Because the magnitude of the contribution of a wiggle w_k to a face is at most $2k$ (see section 2.2), we have that $|\omega(f)| \leq 2(\eta_j - 2\theta)$. Therefore, if we are able to show that $|\tau(f)|$ is at most $2(\eta_j - 2\theta)$, then because $\tau(f)$ is an even number, and w can give any even valued contribution in the range $-2(\eta_j - 2\theta)$ to $2(\eta_j - 2\theta)$, we will be able to show that it is possible to assign labels to the angles of w such that the contribution of w to f is equal to $-\tau(f)$.

We now prove that $|\tau(f)|$ is at most $2(\eta_j - 2\theta)$. From the construction of $\vec{\mathcal{G}}$ and the fact that a tendril T_k gives a contribution equal to either $2k$ or $-2k$ to a face (see section 2.2), it follows that the maximum value of $|\tau(f)|$ is $2\eta_j$, and $|\tau(f)|$ is a multiple of 2θ. Therefore, either $|\tau(f)| = 2\eta_j$, or $|\tau(f)| = 2\eta_j - 2\theta$, or $|\tau(f)| \leq 2\eta_j - 4\theta$. However, because $|\omega(f)| \leq 2(\eta_j - 2\theta)$, from Lemma 4.4 it follows that $|\tau(f)| \leq 2(\eta_j - 2\theta) + \theta = 2\eta_j - 3\theta$. Hence, $|\tau(f)|$ cannot be equal to $2\eta_j$ or $2\eta_j - 2\theta$. Consequently, $|\tau(f)| \leq 2\eta_j - 4\theta = 2(\eta_j - 2\theta)$.

Case 3. f is the dummy face of ψ'. Let T be a tendril of $\vec{\mathcal{G}}$. If T contributes k to a primary face of ψ', it also contributes $-k$ to another primary face of ψ'. Therefore, the total contribution of the tendrils of ψ', when summed over all its primary faces, is 0. Similarly, the total contribution of the wiggles of ψ', when summed over all its primary faces, is 0. We have already shown by considering Cases 1 and 2 that if h is a nondummy face, then $\tau'(h) + \omega'(h) = 0$. Therefore, it follows that $\tau'(f) + \omega'(f) = 0$. □

THEOREM 4.6. *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses and the associated planar switch-flow network $\mathcal{P}$, digraph $\vec{\mathcal{G}}$ associated with $\mathcal{S}$ and $\mathcal{P}$ has $O(n^3m^2)$ vertices and edges and can be constructed in $O(n^3m^2)$ time. Instance $\mathcal{S}$ is satisfiable and network $\mathcal{P}$ admits a feasible flow if and only if digraph $\vec{\mathcal{G}}$ is upward planar. Also, given an upward planar embedding for $\vec{\mathcal{G}}$, a feasible flow for $\mathcal{P}$ and a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^3m^2)$.*

Proof. Since $\theta = 4 = O(1)$, from the construction of $\vec{\mathcal{G}}$ we have that the number of vertices and edges in a tendril or a wiggle is $O(n)$. Since $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges (see Theorem 3.5), $\vec{\mathcal{G}}$ has $O(n^3m^2)$ vertices and edges. $\mathcal{P}$ can be constructed from $\mathcal{S}$ in $O(n^2m^2)$ time (see Theorem 3.5), and $\vec{\mathcal{G}}$ can be constructed from $\mathcal{P}$ in $O(n^3m^2)$ time. Thus, we can construct $\vec{\mathcal{G}}$ from $\mathcal{S}$ in $O(n^3m^2)$ time.

We now show that instance $\mathcal{S}$ is satisfiable and network $\mathcal{P}$ admits a feasible flow if and only if digraph $\vec{\mathcal{G}}$ is upward planar.

We establish the following correspondences between digraph $\vec{\mathcal{G}}$ and network $\mathcal{P}$ (see Figure 4.3):

- The faces of $\vec{\mathcal{G}}$ correspond to the vertices of $\mathcal{P}$.
- The tendrils and wiggles of $\vec{\mathcal{G}}$ correspond to the the edges of $\mathcal{P}$.
- Flipping a tendril T_k of $\vec{\mathcal{G}}$ corresponds to orienting an edge e of $\mathcal{P}$, where e is the dual of the edge replaced by T_k in constructing $\vec{\mathcal{G}}$ from $\vec{\mathcal{D}}$. Edge e is oriented towards an endpoint v (which is a vertex of $\mathcal{P}$) if and only if v corresponds to f and T_k contributes $2k$ to f.
- The contribution of a tendril or wiggle U of $\vec{\mathcal{G}}$ corresponds to the flow in an edge e of $\mathcal{P}$. Here e is the dual of the edge which is replaced by U in constructing $\vec{\mathcal{G}}$ from $\vec{\mathcal{D}}$. The contribution of U to a face f is equal to the amount of the flow coming through e into the endpoint (of e) that corresponds to f.
- The balance of the contributions of the tendrils and wiggles to the faces of $\vec{\mathcal{G}}$ corresponds to the conservation of flow at the corresponding vertices of $\mathcal{P}$; i.e., the total contribution of the tendrils and wiggles to a face is zero if and only if there is a conservation of flow at its corresponding vertex in $\mathcal{P}$.

From Theorem 3.5, Lemma 4.5, and the correspondence established above between a feasible flow in $\mathcal{P}$ and the upward planarity of $\vec{\mathcal{G}}$, it follows that $\mathcal{S}$ is satisfiable and $\mathcal{P}$ admits a feasible flow if and only if $\vec{\mathcal{G}}$ is upward planar. It also follows that given an upward planar embedding for $\vec{\mathcal{G}}$, a feasible flow for $\mathcal{P}$ and a satisfying truth assignment for $\mathcal{S}$ can be computed in $O(n^3m^2)$ time. □

From Theorem 4.6, we conclude the following corollary.

COROLLARY 4.7. *Upward planarity testing is NP-complete.*

5. Rectilinear planarity testing. In this section, we show that rectilinear planarity testing is NP-complete by reducing the problem of computing a feasible flow in the planar switch-flow network associated with an instance of NOT-ALL-EQUAL-3-SAT to the problem of testing the rectilinear planarity of a suitable graph $\mathcal{G}$. The construction of $\mathcal{G}$ is similar to the construction of $\vec{\mathcal{G}}$ in section 4 and is carried out in several stages, where at each stage an intermediate graph is produced.

Let $\mathcal{S}$ be an instance of NOT-ALL-EQUAL-3-SAT with n variables and m clauses; let $\mathcal{P}$ be the associated planar switch-flow network of $\mathcal{S}$ with parameter $\theta = 4 + 37nm$ (see section 3).

Let $\mathcal{D}$ be the dual graph of $\mathcal{P}$. Starting from $\mathcal{D}$, we construct a degree-3 planar graph $\mathcal{F}$ using the following two-step process:

1. First, replace each vertex of $\mathcal{D}$ by a binary tree with d leaves. Let $\mathcal{E}$ be the resultant graph.
2. Next, replace each edge e of $\mathcal{E}$ with a chain c_e consisting of five edges. The middle edge of c_e is called the *representative* of e in $\mathcal{F}$.

Since $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges, it follows that $\mathcal{D}$ and $\mathcal{F}$ also have $O(n^2m^2)$ vertices and edges each.

LEMMA 5.1. *Graph $\mathcal{F}$ has a unique planar embedding and admits a rectilinear embedding, which can be constructed in linear time.*

Proof. Since $\mathcal{D}$ has a unique planar embedding, it follows that $\mathcal{F}$ also has a unique planar embedding.

It is known that every degree-4 planar graph admits an orthogonal drawing with at most four bends per edge, which can be constructed in linear time (see, e.g., [31]). Hence, $\mathcal{E}$ also admits a planar orthogonal drawing R with at most four bends per edge. Because each edge e of $\mathcal{E}$ is replaced by a chain c_e in $\mathcal{F}$, from R we can construct a rectilinear embedding of $\mathcal{F}$ by replacing each bend of e by an intermediate vertex of c_e. □

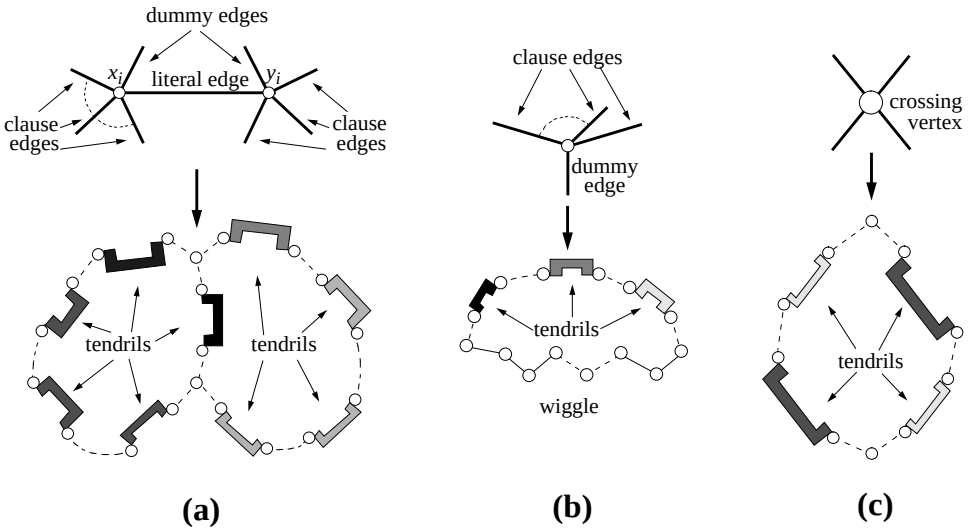


FIG. 5.1. *Schematic illustration of graph $\mathcal{G}$ obtained from $\mathcal{F}$ by replacing edges with rectilinear tendrils and wiggles:* (a) *the two faces of $\mathcal{G}$ associated with literal vertices x_i and y_i of $\mathcal{P}$;* (b) *the face of $\mathcal{G}$ associated with a clause vertex of $\mathcal{P}$;* (c) *the face of $\mathcal{G}$ associated with a crossing vertex of $\mathcal{P}$.*

We construct $\mathcal{G}$ from $\mathcal{F}$ by replacing the edges of $\mathcal{F}$ with subgraphs (rectilinear tendrils or wiggles) as follows (see Figure 5.1). Let e be an edge of $\mathcal{D}$ and *rep(e)* be the representative of e in $\mathcal{G}$.

- If e is the dual of a literal edge, fragment edge, or dummy edge incident on a literal vertex, then *rep(e)* is replaced with a rectilinear tendril T_c, where $[c]$ is the capacity range of the dual edge of e. Note that c is a multiple of parameter θ.
- If e is the dual of a dummy edge e' incident on a clause vertex, then *rep(e)* is replaced with a rectilinear wiggle W_c, where $[0 \cdots c]$ is the capacity range of the dual edge of e.

The vertices of $\mathcal{G}$ that are also vertices of $\mathcal{F}$ are called its *primary* vertices. We define the *primary*, *secondary*, *dummy*, and *nondummy* faces of $\mathcal{G}$ similar to their definition for $\vec{\mathcal{G}}$ in section 4 and also establish similar correspondences between the primary faces of an embedding ψ of $\mathcal{G}$, the faces of $\mathcal{F}$, and the vertices of $\mathcal{P}$.

By Lemma 5.1 and the construction of graph $\mathcal{G}$, all the embeddings of $\mathcal{G}$ are obtained by choosing one of the two possible flips for each rectilinear tendril.

Let ψ be a rectilinear embedding of $\mathcal{G}$. Recall that a rectilinear tendril T_k contributes one of $4k$, $4k+1$, $4k+2$, $-4k$, $-(4k+1)$, and $-(4k+2)$ to a face of $\mathcal{G}$. In ψ, the *significant* contribution of a rectilinear tendril T_k to a face is $4k$ if its contribution is one of $4k$, $4k+1$, and $4k+2$, and is $-4k$ otherwise. In ψ, the *significant* contribution of a rectilinear wiggle to a face is equal to its contribution to the face. Hence, the difference in the contribution and significant contribution of a tendril (wiggle) to a face of ψ is at most 2 (0). Also, the total significant contribution of the tendrils to a face of ψ is a multiple of 4θ. The *contribution* of a primary vertex to a face f of ψ is 1 if its angle in f is labeled 3, is -1 if its angle in f is labeled 1, and is 0 otherwise.

LEMMA 5.2. *If $n \geq 3$ and $m \geq 3$, then in a rectilinear embedding of $\mathcal{G}$ the magnitude of the total contribution of primary vertices to a nondummy face is at*

most $35nm$.

Proof. We show that, in a rectilinear embedding of $\mathcal{G}$, a nondummy face has at most $35nm$ primary vertices, and hence the magnitude of the total contribution of primary vertices to it is at most $35nm$.

Let f be a face of $\mathcal{D}$ with k vertices. Let f be the dual of vertex u of $\mathcal{P}$. Expanding a vertex of f with degree d by a binary tree adds at most $d-1$ vertices to f. Hence, expanding each vertex of f by a binary tree adds at most $r-k$ vertices to f, where r is the facial degree of u. Therefore, after step 1, f has at most $k+r-k=r$ vertices. Hence, after replacing each edge of f by a chain of five edges in step 2, f has at most $5r$ vertices. From Theorem 3.5, r is at most $7nm$ for $n, m \geq 3$. Hence, f has at most $35nm$ vertices in $\mathcal{G}$. □

LEMMA 5.3. *Let f be a primary face of a rectilinear embedding ψ of $\mathcal{G}$. Let $\tau(f)$ and $\omega(f)$ be the total significant contributions to f of its tendrils and wiggles, respectively. Then*

$$|\tau(f) + \omega(f)| \leq \theta.$$

Proof. Let $\nu(f)$ be equal to the number of primary angles labeled 3 minus the number of primary angles labeled 1 of f. Let $\tau'(f)$ and $\omega'(f)$ be the total contribution to f of its tendrils and wiggles, respectively. Let us denote with $N_3(f)$ and $N_1(f)$ the number of angles of f labeled 3 and 1, respectively. An angle of f is either a primary angle of f or an angle of one of its tendrils and wiggles. Therefore, $N_3(f) - N_1(f) = \nu(f) + \tau'(f) + \omega'(f)$. Hence, $\tau'(f) + \omega'(f) = N_3(f) - N_1(f) - \nu(f)$. Therefore, $|\tau'(f) + \omega'(f)| = |N_3(f) - N_1(f) - \nu(f)| \leq |N_3(f) - N_1(f)| + |\nu(f)|$. Because ψ is a rectilinear embedding, the sum of the labels of the angles around each vertex of ψ is equal to 4. Hence, because each angle of $\mathcal{G}$ has label at least 1, and each vertex of $\mathcal{G}$ has degree at least 2, no angle of $\mathcal{G}$ has label 4. Therefore, $N_4(f) = 0$. Hence, from Lemma 2.2, it follows that $|N_3(f) - N_1(f)| = 4$. Therefore, $|\tau'(f) + \omega'(f)| \leq 4 + |\nu(f)|$. Since the difference in the contribution and significant contribution of a tendril to a face of ψ is at most 2, and f has at most nm tendrils, we have that $|\tau(f)| \leq |\tau'(f)| + 2nm$. Since $\omega(f) = \omega'(f)$, it follows that $|\tau(f) + \omega(f)| \leq |\tau'(f) + \omega'(f)| + 2nm \leq 4 + |\nu(f)| + 2nm$. Since, from Lemma 5.2, $|\nu(f)| \leq 35nm$, we have that $|\tau(f) + \omega(f)| \leq 4 + 37nm$. Recall from the beginning of this section that $\theta = 4 + 37nm$. Hence, $|\tau(f) + \omega(f)| \leq \theta$. □

We are now ready to present Lemma 5.4.

LEMMA 5.4. *Graph $\mathcal{G}$ is rectilinear planar if and only if its tendrils can be flipped and labels can be assigned to the angles of its wiggles such that for every primary face the total significant contribution to it of its tendrils and wiggles is zero.*

Proof. If. From Lemma 5.1, $\mathcal{F}$ has a rectilinear embedding $\psi_{\mathcal{F}}$. Let g' be the external face of $\psi_{\mathcal{F}}$. Let ψ be a labeled embedding of $\mathcal{F}$ such that, for every face f of ψ,

- if f is a secondary face, then the angles of f are labeled 1,
- or else (f is a primary face)
  - the contribution of a tendril or a wiggle to f is equal to its significant contribution, and the total contribution to f of its tendrils and wiggles is zero, and
  - its primary angles have the same label as the corresponding angles of $\psi_{\mathcal{F}}$.

The rest of the proof uses the same arguments as those in the *if* part of the proof for Lemma 4.5 with the contribution of a tendril or wiggle replaced by the significant contribution of a rectilinear tendril or wiggle.

Only if. The proof uses Lemma 5.3 and the same arguments as those by the *only if* part of the proof for Lemma 4.5 with the contribution of a tendril or wiggle replaced by the significant contribution of a rectilinear tendril or wiggle. □

THEOREM 5.5. *Given an instance $\mathcal{S}$ of* NOT-ALL-EQUAL-3-SAT *with n variables and m clauses, graph $\mathcal{G}$ associated with $\mathcal{S}$ has $O(n^4m^3)$ vertices and edges and can be constructed in $O(n^4m^3)$ time. Instance $\mathcal{S}$ is satisfiable if and only if graph $\mathcal{G}$ is rectilinear planar. Also, given a rectilinear planar embedding for $\mathcal{G}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^4m^3)$.*

Proof. Since $\theta = 4+37nm = O(nm)$, from the construction of $\mathcal{G}$ we have that the number of vertices and edges in a tendril or a wiggle is $O(n^2m)$. Since $\mathcal{P}$ has $O(n^2m^2)$ vertices and edges (see Theorem 3.5), $\mathcal{G}$ has $O(n^4m^3)$ vertices and edges. $\mathcal{P}$ can be constructed from $\mathcal{S}$ in $O(n^2m^2)$ time (see Theorem 3.5), and $\mathcal{G}$ can be constructed from $\mathcal{P}$ in $O(n^4m^3)$ time. Hence, we can construct $\mathcal{G}$ from $\mathcal{S}$ in $O(n^4m^3)$ time.

We now show that instance $\mathcal{S}$ is satisfiable and network $\mathcal{P}$ admits a feasible flow if and only if graph $\mathcal{G}$ is rectilinear planar.

We establish the following correspondences between graph $\mathcal{G}$ and network $\mathcal{P}$ (see Figure 5.1):

- The faces of $\mathcal{G}$ correspond to the vertices of $\mathcal{P}$.
- The rectilinear tendrils and wiggles of $\mathcal{G}$ correspond to the the edges of $\mathcal{P}$.
- Flipping a rectilinear tendril T_k of $\mathcal{G}$ corresponds to orienting an edge e of $\mathcal{P}$. Edge e is oriented towards an endpoint v (which is a vertex of $\mathcal{P}$) if and only if v corresponds to f and the significant contribution of T_k to f is $4k$.
- The significant contribution of a tendril or wiggle U of $\mathcal{G}$ corresponds to the flow in an edge e of $\mathcal{P}$. Edge e is the dual of the edge whose representative is replaced by U in constructing $\mathcal{G}$ from $\mathcal{F}$. The significant contribution of U to a face f is equal to the amount of the flow coming through e into the end point (of e) that corresponds to f.
- The balance of the significant contributions of the tendrils and wiggles to the faces of $\mathcal{G}$ corresponds to the conservation of flow at the corresponding vertices of $\mathcal{P}$, i.e., the total significant contribution of the tendrils and wiggles to a face f is zero if and only if there is a conservation of flow at its corresponding vertex in $\mathcal{P}$.

From Theorem 3.5, Lemma 5.4, and the correspondence established above between a feasible flow in $\mathcal{P}$ and the rectilinear planarity of $\mathcal{G}$, it follows that instance $\mathcal{S}$ is satisfiable and $\mathcal{P}$ admits a feasible flow if and only if graph $\mathcal{G}$ is rectilinear planar. It also follows that given a rectilinear planar embedding for $\mathcal{G}$, a satisfying truth assignment for $\mathcal{S}$ can be computed in time $O(n^4m^3)$. □

From Theorem 5.5 we conclude the following corollaries.

COROLLARY 5.6. *Rectilinear planarity testing is NP-complete.*

COROLLARY 5.7. *Computing a planar orthogonal drawing with the minimum number of bends is NP-hard.*

We can strengthen Corollary 5.7 as follows.

COROLLARY 5.8. *Let G be an n-vertex planar graph whose minimum number of bends in any planar orthogonal drawing is b^*. Computing a planar orthogonal drawing of G with $O(b^* + n^{1-\epsilon})$ bends is NP-hard for $\epsilon > 0$.*

Proof. Suppose there is a polynomial-time algorithm A that computes a planar orthogonal drawing of G with at most $c(b^* + n^{1-\epsilon})$ bends, where c is some constant. We can then use algorithm A to test in polynomial time whether graph G is rectilinear planar as follows. Construct a graph G' consisting of $K = \lceil (cn^{1-\epsilon})^{1/\epsilon} \rceil + 1$ copies of

G and give G' as an input to algorithm A. Clearly, G is rectilinear planar if and only if G' has a planar orthogonal drawing with fewer than K bends. Since G' has Kn vertices and $K > c(Kn)^{1-\epsilon}$, algorithm A computes a drawing of G' with fewer than K bends if and only if G is rectilinear planar. □

6. Conclusions. Finding efficient algorithms for upward and rectilinear planarity testing had been an open problem for many years. In this paper we have shown that a polynomial-time algorithm for either of these problems is unlikely to exist by proving that both problems are NP-complete. NP-completeness of rectilinear planarity testing also implies that the bend-minimization problem for planar orthogonal drawings is NP-hard.

Acknowledgments. We would like to thank Giuseppe Di Battista for useful discussions and Ivan Rival for comments on a preliminary version of this paper.

REFERENCES

[1] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis, *How to draw a series-parallel digraph*, Internat. J. Comput. Geom. Appl., 4 (1994), pp. 385–402.
[2] P. Bertolazzi and G. Di Battista, *On upward drawing testing of triconnected digraphs*, in *Proceedings of the 7th Annual Symposium on Computational Geometry*, North Conway, NH, 1991, pp. 272–280.
[3] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino, *Upward drawings of triconnected digraphs*, Algorithmica, 6 (1994), pp. 476–497.
[4] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia, *Optimal upward planarity testing of single-source digraphs*, SIAM J. Comput., 27 (1998), pp. 132–169.
[5] T. Biedl and G. Kant, *A better heuristic for orthogonal graph drawings*, Comput. Geom., 9 (1998), pp. 159–180.
[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: An annotated bibliography*, Comput. Geom., 4 (1994), pp. 235–282.
[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing*, Prentice Hall, Upper Saddle River, NJ, 1999.
[8] G. Di Battista, G. Liotta, and F. Vargiu, *Spirality and optimal orthogonal drawings*, SIAM J. Comput., 27 (1998), pp. 1764–1811.
[9] G. Di Battista, W. P. Liu, and I. Rival, *Bipartite graphs upward drawings and planarity*, Inform. Process. Lett., 36 (1990), pp. 317–322.
[10] G. Di Battista and R. Tamassia, *Algorithms for plane representations of acyclic digraphs*, Theoret. Comput. Sci., 61 (1988), pp. 175–198.
[11] G. Di Battista, R. Tamassia, and I. G. Tollis, *Area requirement and symmetry display of planar upward drawings*, Discrete Comput. Geom., 7 (1992), pp. 381–401.
[12] S. Even and G. Granot, *Rectilinear Planar Drawings with Few Bends in Each Edge*, Technical report 797, Computer Science Department, Technion, Haifa, Israel, 1994.
[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
[14] M. D. Hutton and A. Lubiw, *Upward planar drawing of single-source acyclic digraphs*, SIAM J. Comput., 25 (1996), pp. 291–311.
[15] G. Kant, *Drawing planar graphs using the canonical ordering*, Algorithmica, 16 (1996), pp. 4–32.
[16] D. Kelly, *Fundamentals of planar ordered sets*, Discrete Math., 63 (1987), pp. 197–216.
[17] D. Kelly and I. Rival, *Planar lattices*, Canad. J. Math., 27 (1975), pp. 636–665.
[18] A. Lempel, S. Even, and I. Cederbaum, *An algorithm for planarity testing of graphs*, in Theory of Graphs, Gordon and Breach, New York, 1967, pp. 215–232.
[19] Y. Liu, P. Marchioro, and R. Petreschi, *At most single-bend embeddings of cubic graphs*, Appl. Math. J. Chinese Univ. Ser. B, 9 (1994), pp. 127–142.
[20] Y. Liu, P. Marchioro, R. Petreschi, and B. Simeone, *Theoretical Results on at Most 1-Bend Embeddability of Graphs*, Technical report, Dipartimento di Statistica, Università di Roma "La Sapienza," Rome, Italy, 1990.
[21] Y. Liu, A. Morgana, and B. Simeone, *General theoretical results on rectilinear embeddability of graphs*, Acta Math. Appl. Sinica, 7 (1991), pp. 187–192.

[22] Y. Liu, A. Morgana, and B. Simeone, *A Linear Algorithm for* 3*-Bend Embeddings of Planar Graphs in the Grid*, manuscript, 1993.
[23] A. Papakostas, *Upward planarity testing of outerplanar dags*, in Graph Drawing, R. Tamassia and I. G. Tollis, eds., Lecture Notes in Comput. Sci. 894, Springer-Verlag, Berlin, 1995, pp. 298–306.
[24] C. Platt, *Planar lattices and planar graphs*, J. Combin. Theory Ser. B, 21 (1976), pp. 30–39.
[25] I. Rival, *The diagram*, in Graphs and Order, I. Rival, ed., Reidel, Dordrecht, the Netherlands, 1985, pp. 103–133.
[26] I. Rival, *Graphical data structures for ordered sets*, in Algorithms and Order, I. Rival, ed., Kluwer Academic Publishers, Dordrecht, the Netherlands, 1989, pp. 3–31.
[27] I. Rival, *Reading, drawing, and order*, in Algebras and Orders, I. G. Rosenberg and G. Sabidussi, eds., Kluwer Academic Publishers, Dordrecht, the Netherlands, 1993, pp. 359–404.
[28] Y. Shiloach, *Arrangements of Planar Graphs on the Planar Lattice*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1976.
[29] J. A. Storer, *On minimal node-cost planar embeddings*, Networks, 14 (1984), pp. 181–212.
[30] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Comput., 16 (1987), pp. 421–444.
[31] R. Tamassia and I. G. Tollis, *Planar grid embedding in linear time*, IEEE Trans. Circuits Systems, 36 (1989), pp. 1230–1234.
[32] R. Tamassia and J. S. Vitter, *Parallel transitive closure and point location in planar structures*, SIAM J. Comput., 20 (1991), pp. 708–725.
[33] C. Thomassen, *Planar acyclic oriented graphs*, Order, 5 (1989), pp. 349–361.
[34] L. Valiant, *Universality considerations in VLSI circuits*, IEEE Trans. Comput., 30 (1981), pp. 135–140.
[35] G. Vijayan and A. Wigderson, *Rectilinear graphs and their embeddings*, SIAM J. Comput., 14 (1985), pp. 355–372.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 626–641

NEW ALGORITHMIC ASPECTS OF THE LOCAL LEMMA WITH APPLICATIONS TO ROUTING AND PARTITIONING*

TOM LEIGHTON†, CHI-JEN LU‡, SATISH RAO§, AND ARAVIND SRINIVASAN¶

Abstract. The Lovász local lemma (LLL) is a powerful tool that is increasingly playing a valuable role in computer science. The original lemma was nonconstructive; a breakthrough of Beck and its generalizations (due to Alon and Molloy and Reed) have led to constructive versions. However, these methods do not capture some classes of applications of the LLL. We make progress on this by providing algorithmic approaches to two families of applications of the LLL. The first provides constructive versions of certain applications of an extension of the LLL (modeling, e.g., hypergraph-partitioning and low-congestion routing problems); the second provides new algorithmic results on constructing disjoint paths in graphs. Our results can also be seen as constructive upper bounds on the integrality gap of certain packing problems. One common theme of our work is a "gradual rounding" approach.

Key words. disjoint paths, randomized rounding, integer programming

AMS subject classification. 90-XX

PII. S0097539700379760

1. Introduction.

The Lovász local lemma (LLL) is a powerful tool that can be used to show the existence of various types of (discrete) structures [7]. Let e denote the base of natural logarithms and $E_1, E_2, \ldots, E_m$ be events with $\max_i \Pr[E_i] \leq p$. If each E_i is mutually independent of all but at most D of the other events E_j and if $ep(D+1) \leq 1$, then the key claim of the LLL (symmetric case) is that $\Pr[\bigwedge_{i=1}^{m} \overline{E_i}] \geq (1-ep)^m > 0$. See, e.g., [3, 19] for many applications of this. The term $(1-ep)^m$, though positive, is often "tiny," so the result does not directly imply an efficient algorithm to produce a structure that avoids all the E_i. Breakthroughs in this direction have been made by Beck [4] and generalized by Alon [1] and Molloy and Reed [18]. However, as mentioned in [4, 1, 18], certain classes of applications of the LLL are not covered by these results. One such class arises because the approaches of [4, 1, 18] require $pD^3 = \mathrm{O}(1)$; they require something close to "$pD^9 = \mathrm{O}(1)$" for some situations. More importantly, as sketched in section 4.1,

*Received by the editors March 6, 2000; accepted for publication (in revised form) October 18, 2000; published electronically October 11, 2001. Conference versions of this work appeared in the following two papers, both in the *ACM-SIAM Symposium on Discrete Algorithms*, 1999: (i) "New algorithmic aspects of the local lemma with applications to routing and partitioning" by the first, third, and fourth authors, and (ii) "A deterministic approximation algorithm for a minmax integer programming problem" by the second author.

http://www.siam.org/journals/sicomp/31-2/37976.html

†Mathematics Department and the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 (ftl@math.mit.edu). This author was supported in part by Army Grant DAAH04-95-1-0607 and DARPA grant N00014-95-1-1246.

‡Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C. (cjlu@iis.sinica.edu.tw).

§University of California, Berkeley, CA 94705 (satishr@cs.berkeley.edu). Part of this work was done while this author was at the NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.

¶Bell Laboratories, Lucent Technologies, 600–700 Mountain Avenue, Murray Hill, NJ 07974-0636 (srin@research.bell-labs.com). Parts of this work were done while this author was at the School of Computing, National University of Singapore, Singapore 119260, supported in part by National University of Singapore Academic Research Fund grants RP960620 and RP970607, and while this author was visiting the NEC Research Institute, 4 Independence Way, Princeton, NJ 08540.

they do not guarantee polynomial-time algorithms in some settings where the underlying random variables have "large" ranges, even if $pD^c = \mathrm{O}(1)$ for an arbitrarily large constant c. Another such class involves some extensions of the LLL where $D \gg p^{-1}$. Here, we provide constructive approaches for certain instances of both of these classes; we now start with a framework that motivates many of the problems considered.

Let $[t] \doteq \{1, 2, \ldots, t\}$. The NP-hard *low-congestion routing* problem has attracted much attention, e.g., from the VLSI layout and network routing viewpoints [21, 11]. Given a graph $G = (V, E)$ and a collection $\mathcal{T} = \{(s_i, t_i) : i \in [k]\}$ of pairs of vertices in G, how do we construct an (s_i, t_i) path for each i so that the *congestion* (maximum number of paths using any given edge) C is minimized? The multicommodity flow relaxation of this problem is to send one unit of flow from s_i to t_i for each i; the resultant minimum *fractional congestion* (maximum amount of flow using any edge) y^* is a lower bound on C. Since this relaxation is, e.g., a linear program (LP), the corresponding optimal multicommodity flow can be found efficiently. Next suppose we do a flow decomposition: for each (s_i, t_i), we find a set of ℓ_i paths $P_{i,j}$ for $j \in [\ell_i]$, with positive flow values that sum to 1. We can assume without loss of generality (w.l.o.g.) that $\sum_{i \in [k]} \ell_i \leq m$, as each time we can select a path and assign it a flow that saturates at least one edge on it (see, e.g., [21]). Finally, we want to choose exactly one path for each pair to minimize the congestion C. The first two steps can be done efficiently, so the final step is our focus.

How large can C be as a function of y^* and of some other parameters of G? (Even the special case $y^* = \mathrm{O}(1)$ is of much interest; $C \leq \mathrm{O}(\log |V| / \log \log |V|)$ here [21]. This special case is related to several conjectures and results [23, 15, 17, 14].) Our first family of results, described in section 1.1, yields new constructive results for generalizations of this question. Our second family of results, discussed in section 1.2, is related to the question, How large can $y^* < 1$ be if we *require* that $C = 1$, i.e., if we need an edge-disjoint routing?

1.1. Routing, partitioning, and minmax integer programs. Our first family of results concerns *minmax integer programs* (MIPs) as defined in the following; these were named *minimax integer programs* in [25].

Definition 1.1. *An MIP has variables Y and $\{x_{i,j} : i \in [k], j \in [\ell_i]\}$, for some integers $\{\ell_i\}$. Let $N = \sum_{i \in [k]} \ell_i$ and let x denote the N-dimensional vector of the variables $x_{i,j}$ (arranged in any fixed order). An MIP seeks to minimize a real Y subject to:* (i) $\forall i \in [k]$ $\sum_{j \in [\ell_i]} x_{i,j} = 1$*;* (ii) *a system of linear inequalities $Ax \leq \vec{Y}$, where $A \in [0,1]^{m \times N}$ and $\vec{Y}$ is the m-dimensional vector with the variable Y in each component, and* (iii) $\forall i, j,\ x_{i,j} \in \{0, 1\}$.

For convenience, we will call (i) the equality constraints, (ii) the congestion constraints, and (iii) the integrality constraints.

Note that the final step of the low-congestion routing problem can be captured by this form of MIP: the binary variable $x_{i,j}$, for $i \in [k]$ and $j \in [\ell_i]$, indicates whether or not the path $P_{i,j}$ is chosen, and $A_{r,(i,j)}$, for $r \in [m]$, $i \in [k]$, and $j \in [\ell_i]$, indicates whether the edge r is in the path $P_{i,j}$. The MIP problem also arises from a canonical NP-hard hypergraph-partitioning problem [9], which can be applied to support some divide and conquer approaches [2, 12, 16]. We are given a set-system $H = (V, F)$, where $V = [k]$ and $F = \{S_1, \ldots, S_M\} \subseteq 2^V$. Given a positive integer ℓ, the problem is to partition V into ℓ parts, so that *each* S_j is "split well": we want a $\chi : V \to [\ell]$ that minimizes $Y = \max_{j \in [M], q \in [\ell]} |\{i \in S_j : \chi(i) = q\}|$. This is easily written as an MIP with $m = M\ell$.

In MIPs, two parameters, d and κ, are of particular importance to us. Let d denote the maximum number of nonzero entries in any column of the matrix A. In the context of the low-congestion routing problem, d is the length of the longest path among the paths $P_{i,j}$. Let $\kappa \geq 1$ be such that the minimum nonzero entry of A is κ^{-1}. As was for the low-congestion routing problem, an LP relaxation for an MIP is to remove the integrality constraints (iii). The resulting LP optimum y^* is a lower bound on Y. As MIPs are NP-hard, we seek approximation algorithms for them: we will present an efficient way of rounding an optimal solution to the LP. The current-best upper-bounds on Y (as a function of y^* and other parameters) are due to [25], improving on earlier work of [21]. In particular, Y has been shown to be "not much above" y^* if d is "small," i.e., if an MIP instance is *column-sparse*. However, these techniques are nonconstructive and are based on an extension of the LLL. Here, we make progress on the algorithmic aspect, i.e., we develop approximation algorithms for MIPs, by focusing on MIP instances where κ is not "too large": indeed, note that we have $\kappa = 1$ for MIPs with $\{0,1\}$ matrices A (e.g., for low-congestion routing and hypergraph-partitioning).

The previous-best result for general MIPs was nonconstructive and showed that $Y \leq y^*(1 + \mathrm{O}(L'(y^*, 1/(2d)))) + \mathrm{O}(1)$ [25], where L' is a function to be defined in section 2. Here, we present a deterministic polynomial-time algorithm that will produce an integral solution of value at most $y^*(1+\mathrm{O}(L'(y^*, 1/(d\kappa))))+\mathrm{O}(1)$. $L'(\mu, p)$ depends logarithmically on p^{-1}; thus, for families of MIPs where, e.g., κ grows at most as fast as poly(d), we match the previous-best results *constructively*, to within a constant factor. Also, for any family of MIPs where $y^* = \omega(\log(d+\kappa))$ as $d\kappa \to \infty$, our algorithm will produce an integral solution of value $y^*(1 + o(1))$ (e.g., in the hypergraph-partitioning problem, we can asymptotically constructively match the LP optimum if $\ell = o((\max_i |S_i|)/\log d)$, where d is the degree of H). For general MIPs, our improvement over the previous-best *constructive* result [21] is in, essentially, replacing m by $d\kappa$ in the integral solution produced: we get good improvements for MIP instances where $d\kappa \ll m$.

See [6] for recent related results on constructive results for families of MIPs.

1.2. Constructive sufficient conditions for disjoint paths. Suppose we require congestion $C = 1$ in the routing problem described before. Let $G = (V, E)$ with $|V| = n$ and $|E| = m$. Randomized rounding approaches show that if $y^* \leq a_0/\sqrt{m}$, then $C = 1$ is achievable constructively; on the other hand, there are instances where $y^* = \Theta(1/\sqrt{n})$ and $C \geq 2$ [11]. Thus, the gap is large in general. An interesting alternative parametrization is based on d, the length of a longest path in a fractional optimal solution; the existentially optimal result that $C = 1$ if $y^* \leq a_1/d$ has been shown via the LLL [14]. (a_0, a_1, a_2, etc. denote positive constants.) It would be of much interest to make this constructive. As mentioned above, the approaches of [4, 1, 18] do not seem to extend here. We get around this by an approach that generates a large number of paths according to a carefully chosen distribution. Though many of these paths could intersect, we show that if $y^* \leq a_2/(d \ln k)$, then we can locate an edge-disjoint collection within these paths with high probability (whp).

Thus, we present new algorithmic approaches to some classes of packing problems, where the previous-best results were nonconstructive, based on versions of the LLL.

2. Preliminaries. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent random variables taking values from $[0,1]$ with $\mathrm{E}[X_i] = p_i$. Let $X \doteq \sum_{i\in[n]} X_i$ and $\mu \doteq$

$\mathrm{E}[X] = \sum_{i\in[k]} p_i$. Let S_b be the bth elementary symmetric polynomial, i.e.,

$$S_b(p_1,\ldots,p_n) \doteq \sum_{I\subseteq[k],|I|=b} \prod_{i\in I} p_i.$$

We summarize the large deviation bounds of Chernoff [5], Hoeffding [10], and Schmidt, Siegel, and Srinivasan [22] in the following lemmas.

LEMMA 2.1. *For $0 \le \delta \le 1$, $\Pr[|X-\mu| \ge \mu\delta] \le 2e^{-\mu\delta^2/3}$.*

LEMMA 2.2. *For $\delta \ge 0$, $\Pr[X \ge \mu(1+\delta)] \le S_{\lceil\mu\delta\rceil}(p_1,\ldots,p_n)/\binom{\mu(1+\delta)}{\lceil\mu\delta\rceil} \le H(\mu,\delta)$, where $H(\mu,\delta) \doteq (e^\delta/(1+\delta)^{(1+\delta)})^\mu$.*

For $p \in (0,1)$, let $L(\mu,p)$ be the smallest $\delta \ge 0$ such that $H(\mu,\delta) \le p$. It is not hard to see that

$$L(\mu,p) = \begin{cases} \Theta\left(\sqrt{\frac{\log p^{-1}}{\mu}}\right) & \text{if } \mu \ge \frac{1}{2}\log p^{-1}; \\ \Theta\left(\frac{\log p^{-1}}{\mu \log((\log p^{-1})/\mu)}\right) & \text{otherwise.} \end{cases}$$

In [25], a related bound $L'(\mu,p)$ was used, which was defined to be the smallest $\delta \ge 0$ such that $\lceil\mu\delta\rceil H(\mu,\delta) \le p$. So, for this δ, we have

$$\Pr[X \ge \mu(1+L'(\mu,p))] \le p/\lceil\mu\delta\rceil.$$

It can also be seen that

$$L'(\mu,p) = \begin{cases} \Theta\left(\sqrt{\frac{\log(\mu+p^{-1})}{\mu}}\right) & \text{if } \mu \ge \frac{1}{2}\log p^{-1}; \\ \Theta\left(\frac{\log p^{-1}}{\mu \log((\log p^{-1})/\mu)}\right) & \text{otherwise.} \end{cases}$$

3. Constructive approximations for minmax integer programs. We now present results for MIPs. Consider a generic MIP and its relaxation, as described in section 1. Let $\{x^*_{i,j} : i \in [k], j \in [\ell_i]\}$ denote the given optimal solution to the LP relaxation. Starting with this, the algorithm will construct a sequence of feasible solutions to the LP relaxation; variables $x_{i,j}$ that get rounded to 0 or 1 will never have their values realtered. There are two relevant parameters:

- ℓ: the maximum, over all $i \in [k]$, of the number of variables $x_{i,j}$ to be rounded[1];
- t: the maximum number of variables to be rounded in a row of congestion constraints.

We will show two ways to do the rounding. The first one has a slightly inferior performance and gives the following theorem.

THEOREM 3.1. *There exists a deterministic polynomial-time algorithm for finding a solution of value at most $y^*(1+O(L(y^*,1/(dt\ell))))$ for any given MIP.*

The dependency on $t\ell$ in Theorem 3.1 is undesirable, as $t\ell$ sometimes could be large. Next, we show how to do the rounding in several iterations, so that t and ℓ decrease in each iteration and finally reach $\mathrm{poly}(\kappa,d,y^*)$, with only a slight increase in y^*. Then we can apply Theorem 3.1 to get the following theorem. Note that $L(y^*,1/\mathrm{poly}(\kappa,d,y^*)) = O(L'(y^*,1/(d\kappa)))$.

THEOREM 3.2. *There exists a deterministic polynomial-time algorithm for finding a solution of value $y^*(1+O(L'(y^*,1/(d\kappa))))+O(1)$ for any given MIP.*

Next, we prove the above two theorems.

[1] We can assume w.l.o.g. that the LP optimal solution is a *basic* feasible solution, so $\ell \le m$ and $t \le m$.

3.1. Proof of Theorem 3.1. Independently for each $i \in [k]$, randomly round exactly one $x^*_{i,j}$ to 1, with $x^*_{i,j}$ chosen with probability $x^*_{i,j}$, and let $z_{i,j}$ be the random variable with the rounded value. Let z be the N-dimensional vector of the $z_{i,j}$'s. We say that a row $r \in [m]$ of the congestion constraints is *affected* by another row $r' \in [m]$ if the random variables $(Az)_r$ and $(Az)_{r'}$ are dependent, which occurs exactly when

$$\exists i \in [k],\ \exists j, j' \in [\ell_i] \text{ s.t. } A_{r,(i,j)}, A_{r',(i,j')} \neq 0, \text{ and } x^*_{i,j}, x^*_{i,j'} \notin \{0,1\}.$$

So each row can be affected by at most $D \doteq dt\ell$ other rows. For $r \in [m]$ and $i \in [k]$, define

$$Z_{r,i} \doteq \sum_{j \in [\ell_i]} A_{r,(i,j)} z_{i,j}.$$

Clearly, for each $r \in [m]$, $Z_{r,1}, \ldots, Z_{r,k}$ is a sequence of independent random variables taking values from $[0,1]$, with $\mathrm{E}[\sum_{i\in[k]} Z_{r,i}] = \mathrm{E}[(Az)_r] \leq y^*$, so

$$\Pr\left[\sum_{i\in[k]} Z_{r,i} > y^*\left(1 + L\left(y^*, \frac{1}{4D}\right)\right)\right] \leq \frac{1}{4D}.$$

By the LLL, there exists a rounding such that for every $r \in [m]$, $(Az)_r \leq y^*(1 + L(y^*, \frac{1}{4D}))$. We will follow the broad approach of [4, 1] to find a rounding such that for every $r \in [m]$,

$$(Az)_r \leq y^*\left(1 + O\left(L\left(y^*, \frac{1}{D}\right)\right)\right).$$

A direct application of the ideas of [4, 1] leads to $(Az)_r \leq cy^*(1 + L(y^*, \frac{1}{D}))$ for some constant $c > 1$, which is not sufficient for use in Theorem 3.2. A twist is needed to achieve our bound.

We first show a randomized algorithm and then derandomize it.

3.1.1. A randomized algorithm. For a row $r \in [m]$ and for $i \in [k], j \in [\ell_i]$ with $x^*_{i,j} \notin \{0,1\}$, we say that $z_{i,j}$ is a variable of r if $A_{r,(i,j)} > 0$. Consider the graph $G(V,E)$, where $V = [m]$ represents the m rows, and two nodes are adjacent iff one can be affected by the other. Note that each node has at most D neighbors. Let $G^{(a,b)}(V,E')$ be the graph with two nodes adjacent iff their distance is exactly a or b in G. Call a set of nodes in G an (a,b)-tree if they form a connected component in $G^{(a,b)}$. Call a node $r \in [m]$ *bad* if

$$(Az)_r = \sum_{i\in[k]} Z_{r,i} > y^*\left(1 + L\left(y^*, \frac{1}{6D^4}\right)\right),$$

which happens with probability at most $\frac{1}{6D^4}$. For a (1, 2)-tree T and a node $r \in [m]$, let $I_{r,T}$ denote the set of indices $i \in [k]$ such that $Z_{r,i}$ can be affected by rerounding variables in the rows represented by $T \setminus \{r\}$; i.e., $I_{r,T}$ is the set

$$\{i \in [k] : \exists r' \in T \setminus \{r\},\ \exists j, j' \in [\ell_i] \text{ s.t. } A_{r,(i,j)}, A_{r',(i,j')} \neq 0, \text{ and } x^*_{i,j}, x^*_{i,j'} \notin \{0,1\}\}.$$

Let $\overline{I}_{r,T} \doteq [k] \setminus I_{r,T}$. Call a node r *bad for* T if

$$\sum_{i\in\overline{I}_{r,T}} Z_{r,i} > \mathrm{E}\left[\sum_{i\in\overline{I}_{r,T}} Z_{r,i}\right] + y^* L\left(y^*, \frac{1}{6D^4}\right); \tag{3.1}$$

for any fixed r, r is bad for T with probability at most $\frac{1}{6D^4}$, by the definition of L. Call a (1, 2)-tree T *bad*, if every node $r \in T$ is either bad or bad for T.

The intuition is that our randomized rounding is unlikely to lead to large bad (1, 2)-trees and that variables in different bad (1, 2)-trees can be re-rounded separately. Our algorithm consists of at most three phases. In the first phase, we find a rounding such that all bad (1, 2)-trees have size at most $2D \log m / \log D$. Then we try to re-round variables in each bad (1, 2)-tree separately. If m is small, the rerounding can be done in the second phase. Otherwise we need another phase.

Phase 1. First we need a lemma.

LEMMA 3.3. *The probability that our randomized rounding leads to a bad* $(1,2)$*-tree of size at least* $2D \log m / \log D$ *is at most* $1/m$.

Proof. Let $u = 2 \log m / \log D$. As a (1, 2)-tree of size Du must contain a (2, 3)-tree of size u and two nodes of a (2, 3)-tree do not affect each other, a (1, 2)-tree of size Du is bad with probability at most $(\frac{1}{6D^4} + \frac{1}{6D^4})^u = (\frac{1}{3D^4})^u$. The number of (1, 2)-trees of size Du is at most $\frac{m}{(D^2-1)Du+1}\binom{D^3u}{u} < m(\frac{3D^3u}{u})^u = m(3D^3)^u$; see the appendix. Thus the probability that we get a bad (1, 2)-tree of size Du is at most

$$m(3D^3)^u \left(\frac{1}{3D^4}\right)^u \le m\left(\frac{1}{D}\right)^u \le \frac{1}{m}. \qquad \square$$

This gives us a randomized algorithm for finding a rounding with no large bad (1, 2)-trees. In section 3.1.2, we will show how to find such a rounding deterministically, by using the method of conditional probabilities [20].

Phase 2. Suppose we have found a rounding with no bad (1, 2)-tree of size Du. Consider a maximal bad (1, 2)-tree T. We will reround all variables in T using the same randomized approach as in Phase 1. Now the neighbors of T become *dangerous* as they are affected by T and could turn bad after the rerounding. Let $N(T)$ denote the set of T's neighbors. Note that nodes in T cannot be affected by other bad (1, 2)-trees and nodes in $N(T)$ are not dangerous for other bad (1, 2)-trees. So we can deal with each bad (1, 2)-tree separately.

For $r \in N(T)$, since T is maximal and $r \notin T$, we see from (3.1) that

$$\sum_{i \in \bar{I}_{r,T}} Z_{r,i} \le \mathrm{E}\left[\sum_{i \in \bar{I}_{r,T}} Z_{r,i}\right] + y^* L\left(y^*, \frac{1}{6D^4}\right). \tag{3.2}$$

After the rerounding, we call a node $r \in T \cup N(T)$ bad if

$$(Az)_r > y^*\left(1 + 2L\left(y^*, \frac{1}{6D^4}\right)\right).$$

For $r \in T$, this obviously happens with probability at most $\frac{1}{6D^4}$. For $r \in N(T)$, as $\sum_{i \in \bar{I}_{r,T}} Z_{r,i}$ is not affected, we get from (3.2) that r is bad only if

$$\sum_{i \in I_{r,T}} Z_{r,i} > \mathrm{E}\left[\sum_{i \in I_{r,T}} Z_{r,i}\right] + y^* L\left(y^*, \frac{1}{6D^4}\right),$$

which happens with probability at most $\frac{1}{6D^4}$.

$T \cup N(T)$ has at most $Du + D^2u = D(D+1)u$ nodes. Suppose $\sqrt{\log m / \log\log m} \le D$. Then $u = 2 \log m / \log D \le 5 \log m / \log\log m \le 5D^2$. So the probability of having

a bad node is at most

$$D(D+1)u\frac{1}{6D^4} < 5(D+1)D^3\frac{1}{6D^4} < 1.$$

We can find a good rerounding in deterministic polynomial time via the method of conditional probabilities. The proof is omitted here, as the idea is similar to that in section 3.1.2.

Otherwise, we have $D < \sqrt{\log m/\log\log m}$. Using a procedure similar to that in Phase 1, we can find a rerounding such that all bad (1, 2)-trees have size at most

$$O(D\log(D^2\log m)/\log D) = O(\sqrt{\log m\log\log m}/\log D).$$

Then we enter Phase 3.

Phase 3. The LLL shows that a good rerounding exists. Since (i) each bad (1, 2)-tree has at most $\beta = O(\sqrt{\log m\log\log m}/\log D)$ nodes, and (ii) $t \le D \le \sqrt{\log m/\log\log m}$, each bad (1, 2)-tree has at most $t\beta = O(\log m/\log D)$ variables. So, we can use an exhaustive search in deterministic poly(m) time to find a good rerounding such that for every $r \in [m]$,

$$(Az)_r \le y^*\left(1 + 3L\left(y^*, \frac{1}{6D^4}\right)\right).$$

3.1.2. Derandomization of Phase 1. Let R denote the set of all (1, 2)-trees of size $Du = 2D\log m/\log D$. From Lemma 3.3, a randomized rounding is unlikely to result in some tree in R being bad. How do we find a good rounding deterministically? The idea is to use the standard *method of conditional probabilities* with a pessimistic estimator [20]. For $i \in [k]$, let x_i^* denote the vector $(x^*_{i,1},\ldots,x^*_{i,\ell_i})$, and let z_i denote $(z_{i,1},\ldots,z_{i,\ell_i})$. We want to round x_i^* to z_i one at a time, subject to the equality constraints of the MIP. The value of z_i is chosen to minimize the probability of some tree in R being bad if we randomly round the remaining $x^*_{i+1},\ldots,x^*_k$, conditional on the previously chosen $z_1,\ldots,z_{i-1}$. The hope is that the final rounding is a good one because the final conditional probability, which is either 0 or 1, is at most the original unconditional one, which is less than 1. However, it is not easy to compute the exact conditional probability at each step. So we use a pessimistic estimator instead.

For $T \in R$, let T_{23} be an arbitrary maximal (2, 3)-tree in T. Suppose that we have already fixed $z_1,\ldots,z_i$. Then

$$\begin{aligned} P_i(z_1,\ldots,z_i) &\doteq \Pr_{z_{i+1},\ldots,z_k}[\exists T \in R, T \text{ is bad} \mid z_1,\ldots,z_i] \\ &\le \sum_{T\in R}\prod_{r\in T_{23}}\Pr_{z_{i+1},\ldots,z_k}[r \text{ is bad or bad for } T \mid z_1,\ldots,z_i]. \end{aligned}$$

Let $b \doteq \lceil y^*L(y^*, \frac{1}{6D^4})\rceil$ and $c \doteq y^*(1 + L(y^*, \frac{1}{6D^4}))$. Let $w_{r,T} \doteq \mathrm{E}[\sum_{j\in\overline{I}_{r,T}} Z_{r,j}] + y^*L(y^*, \frac{1}{6D^4})$, and let $S_b^r(I)$ denote S_b on input $(\mathrm{E}[Z_{r,v}] : v \in I)$, for $I \subseteq [k]$. Note that

$$\mathrm{E}[Z_{r,v}] = \begin{cases} \displaystyle\sum_{j\in[\ell_v]} A_{r,(v,j)}z_{v,j} & \text{if } z_v \text{ has been fixed,} \\ \displaystyle\sum_{j\in[\ell_v]} A_{r,(v,j)}x^*_{v,j} & \text{otherwise.} \end{cases}$$

Consider the following pessimistic estimator:

$$A_i(z_1,\dots,z_i) \doteq \sum_{T\in R}\prod_{r\in T_{23}}\left(\frac{S_b^r([k])}{\binom{c}{b}}+\frac{S_b^r(\overline{I}_{r,T})}{\binom{w_{r,T}}{b}}\right).$$

From Lemma 2.2, we have $P_i(z_1,\dots,z_i)\le A_i(z_1,\dots,z_i)$. Now,

$$\begin{aligned}A_i(z_1,\dots,z_i) &= \sum_{T\in R}\prod_{r\in T_{23}}\mathrm{E}_{z_{i+1}}\left[\frac{S_b^r([k])}{\binom{c}{b}}+\frac{S_b^r(\overline{I}_{r,T})}{\binom{w_{r,T}}{b}}\right]\\ &= \sum_{T\in R}\mathrm{E}_{z_{i+1}}\left[\prod_{r\in T_{23}}\left(\frac{S_b^r([k])}{\binom{c}{b}}+\frac{S_b^r(\overline{I}_{r,T})}{\binom{w_{r,T}}{b}}\right)\right]\\ &= \mathrm{E}_{z_{i+1}}[A_{i+1}(z_1,\dots,z_{i+1})],\end{aligned}$$

where the second equality is because two distinct nodes of T_{23} share no variable. We choose z_{i+1} to minimize $A_{i+1}(z_1,\dots,z_{i+1})$. Then we have

$$\frac{1}{m}\ge A_0\ge A_1(z_1)\ge\cdots\ge A_k(z_1,\dots,z_k)\ge P_k(z_1,\dots,z_k).$$

$P_k(z_1,\dots,z_k)$ is either 1 or 0 depending on whether or not the rounding results in a bad (2, 3)-tree of size u. As $P_k(z_1,\dots,z_k)<1$, we have found a rounding such that there is no bad (1, 2)-tree of size Du.

It remains to show that for any i and $z_1,\dots,z_i$, $A_i(z_1,\dots,z_i)$ can be computed efficiently. There are poly(m) number of (1, 2)-trees of size Du in R, and they can be enumerated in polynomial time (see appendix). Next, we show how to compute S_b efficiently. For any $y_1,\dots,y_k$, note that

$$S_b(y_1,\dots,y_k)=y_k\cdot S_{b-1}(y_1,\dots,y_{k-1})+S_b(y_1,\dots,y_{k-1}).$$

Thus, using a dynamic programming approach that is similar to the standard method of computing Pascal's triangle, we can compute S_b in deterministic polynomial time.

So A_i can be computed in deterministic polynomial time.

3.2. Proof of Theorem 3.2. Let $x=x^*$, $y=y^*$, and $D=dt\ell$. Note that we can just apply Theorem 3.1 if the following holds:

$$D\le\kappa^8d^8 \text{ or } D\le y^8 \text{ or } D\le c_0 \text{ or } D\le y^{-8/5}. \tag{3.3}$$

Here, c_0 is some absolute constant to be chosen after equation (3.6). So suppose (3.3) is not true; we will try to reduce D in several iterations to finally reach this condition. The idea is similar to that in [25], and we make the key observation that the approach of [25] can reduce not only ℓ but also t. So we can use the standard version of the LLL, instead of the generalization of the LLL developed in [25]. We consider two cases depending on the range of y.

Case 1. $y\ge 1$. We will scale up each $x_{i,j}$, do a rounding, and then normalize each $x_{i,j}$ back to satisfy the equality constraint. This is done as follows.

Recall the notion of randomized rounding of a real [21]: given a real $\alpha\ge 0$, we round it to $\lceil\alpha\rceil$ with probability $\alpha-\lfloor\alpha\rfloor$ and round it to $\lfloor\alpha\rfloor$ with probability $1-(\alpha-\lfloor\alpha\rfloor)$. Let $x'_{i,j}=x_{i,j}y^2\log^3 D$. We will randomly round each $x'_{i,j}$ *independently.* (Notice the difference from the rounding in Theorem 3.1.) Let $z_{i,j}$ be the random

variable taking the value $\lceil x'_{i,j} \rceil$ with probability $x'_{i,j} - \lfloor x'_{i,j} \rfloor$ and taking the value $\lfloor x'_{i,j} \rfloor$ with probability $\lceil x'_{i,j} \rceil - x'_{i,j}$. Note that $\mathrm{E}[z_{i,j}] = x'_{i,j}$; hence, for each $r \in [m]$, we have $\mathrm{E}[(Az)_r] \le y^3 \log^3 D$. Let R_r denote the bad event that

$$(Az)_r > y^3 \log^3 D \left(1 + L\left(y^3 \log^3 D, \frac{1}{6D^4}\right)\right),$$

with $\Pr[R_r] \le \frac{1}{6D^4}$. We will call events such as R_r as "type R" events. For each $i \in [k]$, we have $\mathrm{E}[\sum_{j \in [\ell_i]} z_{i,j}] = y^2 \log^3 D$; let S_i denote the bad event that

$$\left| \sum_{j \in [\ell_i]} z_{i,j} - y^2 \log^3 D \right| > 4y \log^2 D,$$

with $\Pr[S_i] \le \frac{1}{6D^4}$ from Lemma 2.1. Call these "type S" events.

Each type R event depends on at most dt type R events and at most t type S events. Each type S event depends on at most $d\ell$ type R events and on no other type S events. There are $m' = m + k$ bad events, each happening with probability at most $\frac{1}{6D^4}$, and each depending on at most D other bad events. From the LLL, there exists a rounding such that no bad event happens. An algorithm similar to that in the previous section can be used to find a good rounding in deterministic polynomial time, as sketched below. In Phase 1, we find a rounding such that all bad (1, 2)-trees have size at most $2D \log m' / \log D$. In Phase 2, either we can find a good rerounding and we are done, or we can find a rerounding such that all bad (1, 2)-trees are very small. Then we can use an exhaustive search to find a good rerounding in Phase 3. After this, we have that $\forall r \in [m]$,

$$(Az)_r \le y^3 \log^3 D(1 + O(1/(y^{1.5} \log D)))$$

and $\forall i \in [k]$,

$$\left| \sum_{j \in [\ell_i]} z_{i,j} - y^2 \log^3 D \right| \le O(y \log^2 D).$$

Then we do the normalization

$$x_{i,j} = \frac{z_{i,j}}{\sum_{v \in [\ell_i]} z_{i,v}}$$

to satisfy the equality constraints. Now for every $r \in [m]$,

$$(Ax)_r \le \frac{y^3 \log^3 D(1 + O(1/(y^{1.5} \log D)))}{y^2 \log^3 D(1 - O(1/(y \log D)))} = y\left(1 + O\left(\frac{1}{y \log D}\right)\right), \tag{3.4}$$

so the values $(Ax)_r$ increase just a little. Notice that each $z_{i,j}$ is a nonnegative integer. Thus, for each $i \in [k]$, the number of nonzero $z_{i,j}$ (or $x_{i,j}$), for $j \in [\ell_i]$, is at most

$$\sum_{j \in [\ell_i]} z_{i,j} = O(y^2 \log^3 D) = O(D^{1/4} \log^3 D), \tag{3.5}$$

since (3.3) is not true. (The constants hidden in the "big-Oh" notation in (3.4) and in (3.5), as well as in the remaining big-Oh upper bounds in this subsection, are all

absolute constants that are *independent* of c_0.) Next, recall that every nonzero entry of A is at least κ^{-1}. So, for each $r \in [m]$, the number of nonzero $A_{r,(i,j)}x_{i,j}$ is at most

$$\kappa(Az)_r = O(\kappa y^3 \log^3 D) = O(\kappa D^{3/8} \log^3 D).$$

The parameters ℓ and t drop to $O(D^{1/4} \log^3 D)$ and $O(\kappa D^{3/8} \log^3 D)$, respectively. Thus there is an absolute constant c_0', independent of c_0, such that the parameter D comes down to

$$\begin{aligned} c_0' d\kappa D^{5/8} \log^6 D &\le c_0' \kappa^{-1} D^{1/8} \kappa D^{5/8} \log^6 D \\ &= c_0' D^{3/4} \log^6 D, \end{aligned} \tag{3.6}$$

which is at most $D^{7/8}$ if we choose c_0 large enough such that $\forall D > c_0$, the bound $c_0' \log^6 D < D^{1/8}$ holds. Then repeating the above process $O(\log \log D)$ times will ensure that D becomes small enough to satisfy condition (3.3).

Finally, how large can y become? Suppose $y_0, y_1, \ldots, y_s$ are the successive values assumed by y, and $D_0, D_1, \ldots, D_s$ are the successive values assumed by D in the above process. We see from (3.6) that if c_0 is large enough, then $\log D_{i+1} \le (7/8) \log D_i$; hence, $\log D_i \ge (8/7)^{i-s} \log D_s$. Thus we see from (3.4) that

$$y_{i+1} \le y_i + O(1/\log D_i) \le y_i + O((7/8)^{s-i} (\log D_s)^{-1});$$

a telescoping sum shows that $y_s \le y_0 + O(1)$. Now that (3.3) holds and y has increased by at most a constant, we can apply Theorem 3.1.

Case 2. $y < 1$. The idea is the same here. The scaling up is now $x'_{i,j} = x_{i,j} y^{-1} \log^3 D$, for each $i \in [k]$ and $j \in [\ell_i]$, and the rounding and normalization are similar.

4. Constructive sufficient conditions for disjoint paths. We now move on to the problem of lower bounds on y^* in our low-congestion routing problem that can guarantee algorithmically that the integral congestion C is 1. Our main result here is Theorem 4.1. In typical applications of it, the parameter p is likely to be $o(1)$ or bounded away from 1. So, the term "$\lceil \ln p^{-1} \rceil$" can be taken to be "$\ln p^{-1}$" for a first reading.

As will be seen below, we will employ certain randomized rounding and "scaling up–randomized rounding–scaling down" ideas similar to those of section 3.2. Also, one of our original applications of Theorem 4.1 was for edge-disjoint paths in *expander graphs*; however, this problem has been completely recently resolved by Frieze [8]. In section 4.1, we discuss why approaches such as those of [4, 1, 18] do not appear to yield Theorem 4.1.

THEOREM 4.1. *There is a constant $c_1 > 0$ such that the following holds. Suppose we are given any (directed or undirected) graph G, any k (multi-)sets of paths in it, $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, and any parameter $p \in (0,1)$. Let the paths in $\mathcal{P}_i$ be denoted $P_{i,j}$; let d be the maximum, over all i, j, of the length of $P_{i,j}$. Let each $P_{i,j}$ carry some nonnegative amount of flow such that* (i) *the total flow in each $\mathcal{P}_i$ is 1, and* (ii) *for each edge in G, the total flow using it is at most $c_1/(d\lceil \ln p^{-1} \rceil)$. Then, there is a randomized polynomial algorithm to pick one path each from at least $k(1-p)$ of the $\mathcal{P}_i$, such that all paths chosen are edge-disjoint whp. (If we want one path from each of the $\mathcal{P}_i$, we can set, e.g., $p = 1/(2k)$.)*

Specializing to the case where all paths in any given $\mathcal{P}_i$ connect the same pair of vertices, we get the following.

COROLLARY 4.2. *There is a constant $c_1 > 0$ such that the following holds. Suppose we are given any (directed or undirected) graph G, any collection $\mathcal{T} = \{(s_i, t_i) : i \in [k]\}$ of vertex-pairs in G, and any parameter $p \in (0,1)$. Suppose there is a unit flow from s_i to t_i for each i, such that the fractional congestion is at most $c_1/(d\lceil \ln p^{-1} \rceil)$; d denotes the length of a longest flow-path. Then, there is a randomized polynomial algorithm to pick an (s_i, t_i)-path for at least $k(1-p)$ indices i, such that all paths chosen are edge-disjoint whp. (If we want one (s_i, t_i)-path for each i, we can set $p = 1/(2k)$.)*

Remark. Our algorithm will be easily seen to run in polynomial time and will involve several random processes. For each process for which success or failure can be checked efficiently, we will just show that the process succeeds with constant probability. This suffices since, letting N denote the "input size" for the problem, we can repeat each such process, say $O(N)$ times, to drive the failure probability down to $2^{-\Theta(N)}$. Thus, our algorithm will succeed whp.

We now present the algorithm and proof for Theorem 4.1.

Proof. We will choose the constant $c_1 \in (0,1)$ sufficiently small. Let $z \doteq d\lceil \ln p^{-1} \rceil / c_1$; thus, the *fractional congestion* (maximum flow using any given edge) is at most $1/z$. There are three simple steps in the algorithm:

Step 1. This is a "scaling up–randomized rounding–scaling down" step, similar to the one employed in Cases 1 and 3 of section 3.2. Let n be the number of vertices in G. Recall the notion of randomized rounding of a real α [21] from Case 1 of section 3.2. For a suitably large absolute constant $c_2 > 2$ (say 20), we scale all the flow values by $c_2 z \ln n$, and then conduct randomized rounding independently on each flow value (i.e., for each path $P_{i,j}$). By a standard analysis via Lemmas 2.1 and 2.2, we have whp that

(P1) the total flow f_i in each $\mathcal{P}_i$ is such that $c_2 z(\ln n)/2 \le f_i \le 2c_2 z \ln n$, and

(P2) the total flow on any edge is at most $2c_2 \ln n$.

(This can also be derandomized easily by the method of conditional probabilities.) Dividing the flow value on each $P_{i,j}$ by f_i, we restore the property that the total flow value in each $\mathcal{P}_i$ is 1. Furthermore, the fractional congestion now is at most

$$2c_2 \ln n (\min_i f_i)^{-1} \le 2c_2 \ln n (c_2 z (\ln n)/2)^{-1} = 4/z. \tag{4.1}$$

Let $x \doteq \max_i f_i$; (P1) shows that

$$c_2 z(\ln n)/2 \le x \le 2c_2 z \ln n. \tag{4.2}$$

Since the randomized rounding leads to integral flow values, we see, for any i, that the number of flow-paths $P_{i,j}$ carrying nonzero flow now is at most f_i; hence, the maximum number of flow-paths $P_{i,j}$ carrying nonzero flow now, for any i, is at most x.

Step 2. Suppose, after the scaling-rounding-rescaling process of Step 1, that flow path $P_{i,j}$ carries a flow of value $g_{i,j} \ge 0$. For each (i,j), we proceed as follows. Let $g'_{i,j}$ be the largest multiple of $1/(2x)$ that is no larger than $g_{i,j}$; replace $P_{i,j}$ by $2xg'_{i,j}$ copies of itself, each of the copies carrying a flow of precisely $1/(2x)$. We refer to the new paths as $P'_{i,\ell}$. Fix i. How many flow paths do we have in any $\mathcal{P}_i$ now? Since each flow value is now exactly $1/(2x)$ and the total flow in $\mathcal{P}_i$ is now at most 1, this number can be at most $2x$. On the other hand, recall that $\sum_j g_{i,j} = 1$, and, as seen above, that there are at most x (nonzero-flow) paths $P_{i,j}$. Since $g'_{i,j} \ge g_{i,j} - 1/(2x)$, we see by summing that $\sum_j g'_{i,j} \ge 1 - x/(2x) = 1/2$. Thus, since each of the new

flow-paths carries a flow of exactly $1/(2x)$, there must be at least x many $P'_{i,j}$. So, for each i, the set A_i of flow-paths $P'_{i,j}$ is such that

$$x \le |A_i| \le 2x. \tag{4.3}$$

In the rest of the algorithm description, "flow paths" shall refer to these newly constructed flow paths $P'_{i,j}$.

Let y denote the maximum number of flow-paths $P'_{i,j}$ that use any given edge. Now, all paths $P'_{r,s}$ carry a flow of exactly $1/(2x)$. Thus, by (4.1),

$$y \le (4/z)/(1/(2x)) = 8x/z \le 8(2c_2 z \ln n)/z = 16c_2 \ln n, \tag{4.4}$$

the second inequality following from (4.2).

Step 3. Let $t \doteq \lceil \ln p^{-1} \rceil$. Independently for each $i \in [k]$, we do the following. For each i, we randomly select t flow-paths (without replacement) from among the collection of flow-paths $P'_{i,j}$. (The bounds $c_2 > 2$, (4.3), and (4.2) show that $|A_i| \ge z \ln n$. Now, $t = c_1 z/d \le z \ln n$ since $c_1 < 1$. So our random-selection process is well defined.) Call all the paths selected here *tentatively chosen*. We will say that two paths $P'_{a,b}$ and $P'_{r,s}$ are *neighbors* iff $a \ne r$ and $P'_{a,b}$ and $P'_{r,s}$ share a common edge.

Call $P'_{i,j}$ *bad* iff it and some neighbor of it are both tentatively chosen; call i dangerous iff all its t tentatively chosen paths $P'_{i,j}$ are bad. We will next prove Theorem 4.3; let us see why this will help us complete the proof of Theorem 4.1. As usual, the probability of 2/3 in Theorem 4.3 can be boosted by repeating the random process sufficiently many times. Thus, we will be able to efficiently (i) select some $T \subseteq [k]$ of cardinality at least $k(1-p)$, and (ii) pick one tentatively chosen $P'_{i,j}$ for each $i \in T$, such that all the paths picked are edge-disjoint. □

THEOREM 4.3. *If $c_1 > 0$ is sufficiently small, then with probability at least* 2/3*, the number of dangerous i is at most kp.*

Proof. Since each path passes through at most d edges and since each edge has at most y flow-paths using it (by definition of y), we see that each flow-path has at most dy neighbors. Fix i. We now bound the probability that i is dangerous. Suppose i is dangerous, and that it has a collection S of t bad paths $P'_{i,j}$. For each path in S, choose one among its bad neighbors arbitrarily. Suppose there are q such distinct bad neighbors $p_1, p_2, \ldots, p_q$ of the elements of S, where $1 \le q \le t$. We can represent this by an unordered q-tuple $\langle [S_1, p_1], [S_2, p_2], \ldots, [S_q, p_q] \rangle$, where (i) the sets S_j form a partition of S, and (ii) p_j is a bad neighbor of all elements of S_j. Consider any such fixed tuple $T = \langle [S_1, p_1], [S_2, p_2], \ldots, [S_q, p_q] \rangle$. Suppose, for some distinct indices $u_1, u_2, \ldots, u_r$, that the $t+q$ paths in T are ℓ_1 paths from A_{u_1}, ℓ_2 paths from A_{u_2}, …, ℓ_r paths from A_{u_r}. (The positive integers r and $\ell_1, \ell_2, \ldots, \ell_r$ are such that $\ell_j \le t$ and $\sum_j \ell_j = t+q$.) The probability of all of T's $(t+q)$ paths getting tentatively chosen is

$$\left(\prod_{j=1}^{r} \left[\binom{|A_{u_j}| - \ell_j}{t - \ell_j} \Big/ \binom{|A_{u_j}|}{t} \right] \right) \le \prod_{j=1}^{r} (t/|A_{u_j}|)^{\ell_j} \le \prod_{j=1}^{r} (t/x)^{\ell_j} = (t/x)^{t+q}; \tag{4.5}$$

the second inequality is a consequence of (4.3).

Fix q. We next upper-bound the number of possible unordered q-tuples $\langle [S_1, p_1], [S_2, p_2], \ldots, [S_q, p_q] \rangle$. To do so, we will bound the number of such *ordered* q-tuples, and divide the result by $q!$. Let us now bound the number of ordered q-tuples. Clearly, we may assume that each S_j is nonempty. We first fix the cardinalities of the S_j: this can be done in $\binom{t-1}{q-1}$ ways, since $\sum_j |S_j| = t$ and as each S_j is nonempty. Suppose

$|S_j| = t_j$. Recall from (4.3) that a member $P'_{i,\ell}$ of S_1 can be picked in at most $2x$ ways. Given this choice, since p_1 is a neighbor of $P'_{i,\ell}$, p_1 can be picked in at most dy ways. Next, since the remaining $t_1 - 1$ elements of S_1 are all neighbors of p_1, they can be picked in at most $\binom{dy}{t_1-1}$ ways. Thus, the number of ways of choosing $[S_1, p_1]$ is at most

$$(2x) \cdot (dy) \cdot \binom{dy}{t_1 - 1} \leq (2x) \cdot (dy)^{t_1}. \tag{4.6}$$

Arguing similarly for $[S_2, p_2], \ldots, [S_q, p_q]$, we see, for each fixed choice of $t_1, t_2, \ldots, t_q$, that the number of ordered q-tuples is at most $\prod_{j=1}^{q}((2x) \cdot (dy)^{t_j}) = (2x)^q \cdot (dy)^t \leq 2^t x^q \cdot (dy)^t$. Next, as seen above, (i) there are

$$\binom{t-1}{q-1} \leq \binom{t}{q} \leq e^q t^q / q^q \leq e^t t^q / q^q$$

ways of choosing the t_i; (ii) the number of unordered q-tuples is the number of ordered q-tuples divided by $q!$. Combining with (4.5), the probability of existence of some q-tuple is at most

$$c_3^t x^q (dy)^t (t/q)^q (t/x)^{t+q} / q! \leq c_4^t (t/q)^{2q} (tdy/x)^t, \tag{4.7}$$

where $c_3, c_4 > 0$ are absolute constants; in going from the left-hand side (l.h.s.) to the right-hand side (r.h.s.), we have used the fact that $q! \geq (q/e)^q \geq q^q/e^t$. By calculus, it is evident that $(t/q)^{2q} \leq e^{2t/e}$. Next, by recalling that $z = dt/c_1$ and by using the lower bound on x and upper bound on y from (4.2) and (4.4), we see from (4.7) that the probability of existence of some q-tuple is at most $(c_5 c_1)^t$, where c_5 is an absolute constant which, in particular, is independent of c_1. Finally, since q can only take t values—from 1 to t—the probability that i is dangerous is at most $t(c_5 c_1)^t$, which can be made, say, at most $p/3$ by taking c_1 appropriately small (recall that $t = \lceil \ln p^{-1} \rceil$). Thus, by Markov's inequality, the probability that there are more than kp dangerous i is at most $1/3$. □

We now sketch two ways in which the constant factor c_1 in Theorem 4.1 can be increased; however, these methods do not replace c_1 by any super-constant term. First, in (4.6), we can upper-bound $\binom{dy}{t_1-1}$ by $(dy)^{t_1-1}/(t_1-1)!$. More interestingly, we can include an additional element of randomness in Step 3 of our algorithm as follows, motivated by a "random permutation" idea of [24]. Choose a random permutation σ of all the paths $P_{i,j'}$, and call $P_{i,j'}$ bad iff it and some neighbor of it *that precedes* $P_{i,j'}$ *in* σ, are both tentatively chosen. Call i dangerous iff all its t tentatively chosen paths $P'_{i,j}$ are bad, and note that we can efficiently select one tentatively chosen path for each nondangerous index i, in such a way that these selected paths are edge-disjoint. This idea helps improve the constant c_1.

4.1. Comparison of Theorem 4.1 with previous methods. We now give a brief sketch of an approach of [18], which generalizes the basic algorithm of [4, 1] to cover many new applications. The reader is referred to these three beautiful papers for more details. Suppose we have a set $\mathcal{F} = \{f_1, f_2, \ldots, f_t\}$ of *independent* random variables, each f_j taking values in some domain of cardinality at most γ. Suppose we also have ("bad") events $E_1, E_2, \ldots, E_m$, such that each E_i is determined by the values of the variables in some $F_i \subseteq \mathcal{F}$. Define $u \doteq \max_i |F_i|$. Let us say that E_i *intersects* E_j iff $F_i \cap F_j \neq \emptyset$; we let D denote the maximum number of other E_j that any given E_i intersects. Let $q = \max_i \Pr[E_i]$. One of the main results of [18] is that if

$$qD^9 < 1/8, \tag{4.8}$$

then we can find an assignment for all the f_i in randomized poly$(m, t, \gamma^{uD \log\log m})$ time such that $\bigwedge_{i \in [m]} \overline{E_i}$ holds.

We now sketch this algorithm. A setting $\mathcal{A}$ of some of the random variables f_j is done in a way such that $\Pr[E_i|\mathcal{A}] \leq q^{2/3} \forall i$. More importantly, as seen in section 3.1.1, the problem of setting the remaining f_j gets split into different "components," where each component has at most $D \log m$ events E_i whp; we can handle each component separately. If need be, we repeat the above process, and end up whp with components of size $O(D \log(D \log m))$; the probability of each E_i conditional on the f_j fixed so far will be at most $q^{1/3}$. The LLL and (4.8) show that there is a setting of the variables in each component such that all the E_i are avoided. Since each component is quite small now, we can now conduct an exhaustive search separately in each component. In particular, if γ and u are "small," say bounded by a constant which is true for many applications of the LLL, the condition (4.8) that is somewhat stronger than the "$eq(D+1) \leq 1$" of the LLL, ensures a constructive version of the LLL. However, as pointed out in [18], if γ is "large," say $(t+m)^{\Theta(1)}$, this method does not guarantee polynomial-time algorithms.

Suppose we were to follow the methodology of [4, 1, 18] to try and prove Theorem 4.1. Motivated by the notation above, let us define γ to be the maximum number of paths in any (multi-)set $\mathcal{P}_i$. In Theorem 4.1, consider the case where we wish to choose one path from each $\mathcal{P}_i$; i.e., $p = \Theta(1/k)$. Combined with the work of [14], the approach sketched above would require that the fractional congestion in Theorem 4.1 be $O(1/d\gamma^c)$, for a certain constant $c > 0$; some basic features of the approaches of [4, 1, 18] make it potentially impossible to eliminate this constant. Thus, situations where γ is "large" pose problems if we use the approach of [4, 1, 18]. Via the approach of Step 1 of our algorithm for Theorem 4.1, we can reduce γ to $O(d \log n)$, but reducing γ to $o(d)$, say, is a challenging open problem. Also, as pointed out in [18] and above, situations where γ is "large," say $n^{\Theta(1)}$, do not necessarily admit of polynomial-time solutions via the approach of [4, 1, 18]. Thus, for instances where $\min\{\gamma, d\}$ is, say, $n^{\Theta(1)}$, it is not clear how to use the methods of [4, 1, 18] to derive polynomial-time algorithms.

5. Open questions. The following problems suggest themselves. Can we get a full algorithmic version of the result of [25]? That is, can we get rid of the "κ" term in Theorem 3.2? Another question is whether L' can be replaced by L in Theorem 3.2. Finally, in the notation of section 1.2, Corollary 4.2 shows that "$y^* \leq a_2/(d \ln k)$" is a sufficient condition for efficiently generating edge-disjoint paths. As pointed out in section 1.2, it would be of much interest to see if this can be improved to "$y^* \leq a_1'/d$" for some constant a_1'.

Appendix. The number of rooted u-node ordered d-ary trees. A rooted tree is called *ordered* if the relative order of each node's children is important; different relative orders give different trees. "d-ary" means that each tree node has at most d children.

The following lemma is well known; for an algebraic proof, see, for example, [13]. For $d = 2$, it gives a derivation of the Catalan number, which has a combinatorial proof using a reflection principle. With the help of David Mix Barrington, we generalize this idea to any d.

LEMMA A.1. *The number of rooted u-node ordered d-ary trees is* $\frac{1}{(d-1)u+1}\binom{du}{u}$.

Proof. Let A denote the set of such trees. Let B be the class of strings from $\{1, 2, \ldots, d\}^{du}$ generated by the following grammar:

$$T \to \lambda \mid 1T\,2T\,3T \cdots dT,$$

where λ denotes the empty string. For a tree in A, let us add imaginary edges to nodes, including leaves, so that each node has exactly d outgoing edges, labeled from 1 through d. Note that there are du edges now. Then one can show that $|B| = |A|$ as each string in B corresponds to the sequence of edge labels seen from a depth-first traversal of some tree in A. There is also a one-to-one correspondence between B and the class C of strings from $\{1, *\}^{du}$ generated by the following grammar:

$$S \to \lambda \mid 1S * S * S \cdots * S,$$

with $d-1$ $*$'s above. Another way to see directly why $|A| = |B| = |C|$ is that they all have the same recurrence relation and the same initial condition.

Let D denote the set of strings from $\{1, *\}^{du}$ with exactly u 1's. As $C \subseteq D$ and $|D| = \binom{du}{u}$, it remains to calculate $|D \setminus C|$. Let D' denote the set of strings from $\{1, *\}^{du}$ with exactly $u-1$ 1's. Clearly $|D'| = \binom{du}{u-1} = \frac{u}{(d-1)u+1}\binom{du}{u}$. Next, we show that $|D \setminus C| = (d-1)|D'|$.

It is not hard to verify that C contains exactly those strings in D satisfying the following *prefix condition*:

- In any prefix, the number of $*$'s is at most $d-1$ times the number of 1's.

Any string in $D \setminus C$ or D' must contain a prefix that first violates this condition. Let $w*$ be any such a prefix consisting of k 1's and $(d-1)k+1$ $*$'s, for some $k < u$, with w satisfying the prefix condition. There are exactly $\binom{du-(dk+1)}{u-k} = (d-1)\binom{du-(dk+1)}{u-k-1}$ strings in $D \setminus C$ having $w*$ as prefix, and there are exactly $\binom{du-(dk+1)}{u-k-1}$ strings in D' having $w*$ as prefix. Ranging through all such strings $w*$, every string in $D \setminus C$ is enumerated exactly once, and so is every string in D'. So, $|D \setminus C| = (d-1)|D'|$, and $|C| = |D| - |D \setminus C| = \frac{1}{(d-1)u+1}\binom{du}{u}$. □

The following corollary follows immediately.

COROLLARY A.2. *In a graph of m nodes and degree d, there are at most $s = \frac{m}{(d-1)u+1}\binom{du}{u}$ connected components of size u, and they can be enumerated in deterministic* poly(s) *time.*

Acknowledgment. We thank Alan Frieze for his helpful suggestions.

REFERENCES

[1] N. ALON, *A parallel algorithmic version of the local lemma*, Random Structures Algorithms, 2 (1991), pp. 367–378.

[2] N. ALON, *The strong chromatic number of a graph*, Random Structures Algorithms, 3 (1992), pp. 1–7.

[3] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, Wiley–Interscience Series, John Wiley, New York, 1992.

[4] J. BECK, *An algorithmic approach to the Lovász local lemma*, Random Structures Algorithms, 2 (1991), pp. 343–365.

[5] H. CHERNOFF, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statist., 23 (1952), pp. 493–509.

[6] A. CZUMAJ AND C. SCHEIDELER, *A new algorithmic approach to the general Lovász local lemma with applications to scheduling and satisfiability problems*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, Portland, OR, 2000, pp. 38–47.

[7] P. ERDŐS AND L. LOVÁSZ, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, A. Hajnal et al., eds., Colloq. Math. Soc. János Bolyai 11, North Holland, Amsterdam, 1975, pp. 609–627.

[8] A. FRIEZE, *Edge-disjoint paths in expander graphs*, In Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 2000, pp. 717–725.

[9] Z. Füredi and J. Kahn, *On the dimensions of ordered sets of bounded degree*, Order, 3 (1986), pp. 15–20.
[10] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13-30.
[11] J. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, Department of EECS, MIT, 1996.
[12] H. J. Karloff and D. B. Shmoys, *Efficient parallel algorithms for edge coloring problems*, J. Alg., 8 (1987), pp. 39-52.
[13] D. Knuth, *The Art of Computer Programming*, vol. I, Addison Wesley, London, 1969, p. 396 (Exercise 11).
[14] F. T. Leighton, S. B. Rao, and A. Srinivasan, *Multi-commodity flow and circuit switching*, in Proc. Hawaii International Conference on System Sciences, 1998, pp. 459–465.
[15] M. Lomonosov, *Combinatorial approaches to multiflow problems*, Discrete Appl. Math., 11 (1985), pp. 1–94.
[16] C.-J. Lu, *Deterministic hypergraph coloring and its applications*, in Proc. Workshop on Randomization and Approximation Techniques in Computer Science, Honolulu, HI, 1998, pp. 35–46.
[17] M. Middendorf and F. Pfeiffer, *On the complexity of the disjoint paths problem*, Combinatorica, 13 (1993), pp. 97–107.
[18] M. Molloy and B. Reed, *Further algorithmic aspects of the Local lemma*, in Proc. ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 524–529.
[19] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
[20] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 38 (1994), pp. 683-707.
[21] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[22] J. P. Schmidt, A. Siegel, and A. Srinivasan, *Chernoff-Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.
[23] P. D. Seymour, *On odd cuts and planar multicommodity flows*, J. London Math. Soc., 42 (1981), pp. 178–192.
[24] J. H. Spencer, *The probabilistic lens: Sperner, Turán and Brégman revisited*, in A Tribute to Paul Erdős, A. Baker, B. Bollobás, A. Hajnal, eds., Cambridge University Press, 1990, pp. 391–396.
[25] A. Srinivasan, *An extension of the Lovász Local Lemma, and its applications to integer programming*, in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 1996, pp. 6–15.

SIAM J. COMPUT.
Vol. 31, No. 2, pp. 642–664

ADAPTIVE AND EFFICIENT ALGORITHMS FOR LATTICE AGREEMENT AND RENAMING*

HAGIT ATTIYA† AND ARIE FOUREN†

Abstract. In a shared-memory system, n independent asynchronous processes, with distinct names in the range $\{0,\ldots,N-1\}$, communicate by reading and writing to shared registers. An algorithm is *wait-free* if a process completes its execution regardless of the behavior of other processes. This paper considers wait-free algorithms whose complexity adjusts to the level of contention in the system: An algorithm is *adaptive* (to total contention) if its step complexity depends only on the actual number of active processes, k; this number is unknown in advance and may change in different executions of the algorithm.

Adaptive algorithms are presented for two important decision problems, *lattice agreement* and $(6k-1)$-*renaming*; the step complexity of both algorithms is $O(k \log k)$. An interesting component of the $(6k-1)$-renaming algorithm is an $O(N)$ algorithm for $(2k-1)$-renaming; this improves on the best previously known $(2k-1)$-renaming algorithm, which has $O(Nnk)$ step complexity.

The efficient renaming algorithm can be modified into an $O(N)$ implementation of atomic snapshots using *dynamic single*-writer multi-reader registers. The best known implementations of atomic snapshots have step complexity $O(N \log N)$ using *static single*-writer multi-reader registers, and $O(N)$ using *multi*-writer multi-reader registers.

Key words. shared-memory systems, wait-free computation, atomic read/write registers, renaming, lattice agreement, atomic snapshots

AMS subject classifications. 68P05, 68Q10, 68Q20, 68Q22

PII. S0097539700366000

1. Introduction.

An asynchronous shared-memory system contains n processes running at arbitrary speeds and communicating by reading from and writing to shared registers; processes have distinct names in the range $\{0,\ldots,N-1\}$, $n \leq N$. In a *wait-free* algorithm, a process terminates in a finite number of steps, even if other processes are very slow, or even stop taking steps completely.

The step complexity of many wait-algorithms depends on N; for example, collecting up-to-date information from all processes typically requires reading array indexed with processes' names. Real distributed systems need to accommodate a large number of processes, i.e., N is large, while often only a small number of processes take part in the computation. For such systems, step complexity depending on n or N is undesirable; it is preferable to have step complexity which adjusts to the number of processes participating in the algorithm.

An algorithm is *adaptive* (to total contention) if its step complexity depends only on the total number of processes participating in the algorithm, denoted k; k is unknown in advance and it may change in different executions of the algorithm. The step complexity of an adaptive algorithm adjusts to the number of active processes: It is constant if a single process participates in the algorithm, and it gradually grows as the number of active processes increases.

*Received by the editors February 1, 2000; accepted for publication (in revised form) December 4, 2000; published electronically October 11, 2001. An extended abstract of this paper appeared in proceedings of the 17*th ACM Symposium on Principles of Distributed Computing*, 1998, pp. 277–286.

http://www.siam.org/journals/sicomp/31-2/36600.html

†Department of Computer Science, The Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il, leonf@cs.technion.ac.il). This work was supported by the fund for the promotion of research at the Technion.

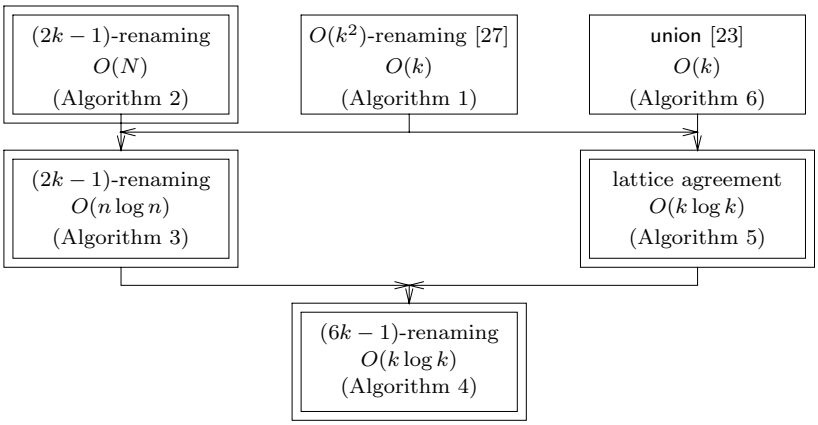


FIG. 1.1. *The algorithms presented in this paper; double boxes indicate the main results.*

A weaker guarantee is provided by *range-independent* algorithms whose step complexity depends only on n, the maximal number of processes; clearly, n is fixed for all executions.[1] The advantage of range-independent algorithms is quite restricted: They require a priori knowledge of n, which is often difficult to determine; moreover, their step complexity is not optimal when the actual number of participating processes is much lower than the upper bound. Yet, as we show, they can be useful tools in the construction of adaptive algorithms.

This paper presents adaptive wait-free algorithms for lattice agreement and renaming, using only read and write operations. Along the way, we improve the step complexity of nonadaptive algorithms for renaming. Figure 1.1 depicts the algorithms presented in this paper.

In the one-shot *M-renaming* problem [10], processes are required to choose distinct names in a range of size $M(k)$, for some bounded function M. This paper does not consider the more general *long-lived* renaming problem [9], in which processes repeatedly *acquire* and *release* names. Adaptive renaming can serve as an intermediate step in adaptive algorithms for other problems [9, 26, 27, 28]: The new names replace processes' original names, making the step complexity depend only on the number of active processes. Our algorithms employ this technique, as well as [6, 7].

An efficient adaptive algorithm for renaming could not be derived from known algorithms: The best previously known algorithm for renaming with linear name space [18] has $O(Nnk)$ step complexity, yielding $O(k^3)$ step complexity (at best) if it can be made adaptive. Thus, we first present a $(2k-1)$-renaming algorithm with $O(N)$ step complexity, which is neither adaptive nor range-independent. This algorithm is based on a new "shrinking network" construction, which we consider to be the novel algorithmic contribution of our paper.

The new linear renaming algorithm is employed in a range-independent algorithm for $(2k-1)$-renaming with $O(n \log n)$ step complexity. Processes start with an adaptive $O(k^2)$-renaming algorithm whose step complexity is $O(k)$; this is a simple modification of the range-independent renaming algorithm of Moir and Anderson [27]. Then, processes reduce the range of names in $O(\log n)$ iterations; each iteration uses our new linear renaming algorithm.

[1] Moir and Anderson [27] use the term "fast," which conflicts with other papers [3, 25].

The range-independent renaming algorithm is used to construct an adaptive $(6k-1)$-renaming algorithm with $O(k \log k)$ step complexity. In this algorithm, processes are partitioned into $O(\log k)$ disjoint sets according to views obtained from an adaptive lattice agreement algorithm (described below). This partition bounds the number of processes in each set and allows them to employ a range-independent $(2k-1)$-renaming algorithm designed for this bound. Different sets use disjoint name spaces; no coordination between the sets is required.

In the *lattice agreement* problem [15], processes obtain comparable (by containment) subsets of the set of active processes. A wait-free lattice agreement algorithm can be turned into a wait-free implementation of an *atomic snapshot* object, with $O(n)$ additional read/write operations [15]. Atomic snapshot objects allow processes to get instantaneous global views ("snapshots") of the shared memory, and thus they simplify the design of wait-free algorithms.

The step complexity of our adaptive algorithm for lattice agreement is $O(k \log k)$. In this algorithm, processes first obtain names in a range of size $O(k^2)$ using the simple algorithm with $O(k)$ step complexity. Based on its reduced name, a process enters an adaptive variant of the tree used in the lattice agreement algorithm of Inoue et al. [23].

Appendix C describes how the shrinking network is modified to get a lattice agreement algorithm with $O(N)$ step complexity, using *dynamic single*-writer single-reader registers; this gives an implementation of atomic snapshots with the same complexity. Previous implementations of atomic snapshots had either $O(N \log N)$ step complexity using *static single*-writer multi-reader registers [16] or $O(N)$ step complexity using *multi*-writer multi-reader registers [23].

The renaming problem was introduced and solved by Attiya et al. [10] for the message-passing model; Bar-Noy and Dolev [17] solved the problem in the shared-memory model. Burns and Peterson [19] considered the *l-assignment* problem—dynamic allocation of l distinct resources to processes. They present a wait-free l-assignment algorithm which assumes $l \geq 2k-1$, where k is the number of processes trying to acquire a resource. All these algorithms have exponential step complexity [21]. Borowsky and Gafni [18] present an algorithm for one-shot $(2k-1)$-renaming using $O(Nnk)$ read/write operations.

Anderson and Moir [9] define long-lived renaming and present range-independent algorithms for one-shot and long-lived renaming; their algorithms use test&set operations. Moir and Anderson [27] introduce a building block, later called a *splitter*, and employ it in range-independent algorithms for long-lived renaming, using read/write operations. Moir and Garay [28, 26] give a range-independent long-lived $O(kn)$-renaming algorithm, using only read/write operations. By combining with a long-lived $(2k-1)$-renaming algorithm [19] they obtain a range-independent long-lived $(2k-1)$-renaming algorithm; its step complexity is dominated by the exponential step complexity of Burns and Peterson's algorithm.

Herlihy and Shavit [22] show that one-shot renaming requires $2k-1$ names. This implies that our range-independent renaming algorithm provides an optimal name space. The name space provided by our adaptive renaming algorithm is not optimal $(M = 6k-1)$; still, it is linear in the number of active processes.

Following the original publication of our paper [12], Afek and Merritt [4] used our algorithms to obtain an adaptive wait-free $(2k-1)$-renaming algorithm, with $O(k^2)$ step complexity.

In another paper [13], we present an adaptive collect algorithm with $O(k)$ step complexity and derive adaptive algorithms for atomic snapshots, immediate snap-

shots, and $(2k-1)$-renaming. That paper emphasizes the modular use of a collect operation to make known algorithms adaptive; the algorithms have higher step complexity than those presented here.

Our algorithms adapt to the total number of participating processes, that is, if a process ever performs a step then it influences the step complexity of the algorithm throughout the execution. More useful are algorithms which adapt to the *current* contention and whose step complexity decreases when processes stop participating. Afek, Dauber, and Touitou [3] present implementations of long-lived objects which adapt to the current contention; they use load-linked and store-conditional operations. Recent papers present algorithms for long-lived renaming [2, 14], collect [6], and snapshots [7] which adapt to the current contention using only read/write operations.

Lamport [25] suggests a *mutual exclusion* algorithm which requires a constant number of steps when a single process wishes to enter the critical section, using read/write operations; when several processes compete for the critical section, the step complexity depends on the range of names. Alur and Taubenfeld [8] show that this behavior is inherent for mutual exclusion algorithms. Choy and Singh [20] present mutual exclusion algorithms whose *time complexity*—the time between consecutive entries to the critical section—is $O(k)$, using only read/write operations. Afek, Stupp, and Touitou [6] use an adaptive collect algorithm to derive an adaptive version of the bakery algorithm [24]; they present another mutual exclusion algorithm in [5]. Recently, Attiya and Bortnikov [11] presented a mutual exclusion algorithm whose time complexity is $O(\log k)$; this algorithm employs an unbalanced tournament tree with the same structure as our adaptive lattice agreement tree.

2. Preliminaries.

2.1. The model. In the shared-memory model, *processes* $p_0, \ldots, p_{n-1}$ communicate by applying *operations* on shared objects. A process p_i is modeled as a (possibly infinite) state machine; process p_i has a distinct name $id_i \in \{0, \ldots, N-1\}$, $n \leq N$.

The shared objects considered in this paper are *atomic read/write registers*, accessed by read and write operations. A read(R) operation does not change the state of R, and returns the current value stored in R; a write(v,R) operation changes the state of R to v. A *multi-writer multi-reader* register allows any process to perform read and write operations. A *single-writer multi-reader* register allows only a single process to perform write operations, and any process to perform read operation. A single-writer multi-reader register is *dynamic* if the identity of the single process writing to the register varies in different executions; otherwise, it is *static*.

An *event* is a computation step by a single process; the process determines the operation to perform according to its state, and its next state according to its state and the value returned by the operation. Computations in the system are captured as sequences of events. An *execution* α is a (finite or infinite) sequence of events $\phi_0, \phi_1, \phi_2, \ldots$. For every $r = 0, 1, \ldots$, if p_i is the process performing the event ϕ_r, then it applies a read or a write operation to a single register and changes its state according to its transition function. There are no constraints on the interleaving of events by different processes, reflecting the assumption that processes are *asynchronous* and there is no bound on their relative speeds.

Consider an execution α of some algorithm A. For process p_i, $step(A, \alpha, p_i)$ is the number of read/write operations p_i performs in α. The step complexity of A in α, denoted $step(A, \alpha)$, is the maximum of $step(A, \alpha, p_i)$ over all processes p_i. A process is *active* in α if it takes a step in α, that is, $step(A, \alpha, p_i) \neq 0$; $k(\alpha)$ denotes the number of active processes in α.

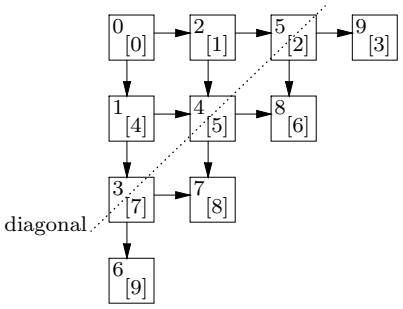


FIG. 2.1. *The grid used for $O(k^2)$-renaming (depicted for $n = 4$).*

Algorithm A is *range-independent* if there is a function $f : \mathcal{N} \mapsto \mathcal{N}$ such that in every execution α of A, $step(A, \alpha) \leq f(n)$. Namely, the step complexity of A in every execution is bounded by a function of the total number of processes (which is known in advance); it does not depend on the range of the initial names.

Algorithm A is *adaptive (to total contention)* if there is a function $f : \mathcal{N} \mapsto \mathcal{N}$ such that in every execution α of A, $step(A, \alpha) \leq f(k(\alpha))$. Namely, the step complexity of A in α is bounded by a function of the number of active processes in α. Clearly, the number of active processes is not known a priori.

A *wait-free* algorithm guarantees that every process completes its computation in a finite number of steps, regardless of the behavior of other processes. Since $k(\alpha)$ is bounded (by n) it follows that adaptive algorithms are wait-free.

2.2. Problems. The *M-renaming* problem [10] requires processes to choose distinct names in a range that depends only on the number of active processes. Namely, there is a function $M : \mathcal{N} \mapsto \mathcal{N}$ such that in every execution α, processes output distinct names in the range $\{0, \ldots, M(k(\alpha)) - 1\}$.

In the *lattice agreement* problem [15], every process p_i outputs V_i a subset of the active processes (e.g., a *view*) such that the following conditions hold:

Self-inclusion: $p_i \in V_i$ for every i.

Comparability: V_i and V_j are comparable (either $V_i \subseteq V_j$ or $V_j \subseteq V_i$) for every i and j.

2.3. Simple $O(k^2)$-renaming in $O(k)$ operations. The first step in our algorithms is a simple adaptive $O(k^2)$-renaming algorithm. This algorithm reduces the range of names to depend only on the number of active processes; later stages use these distinct names, without sacrificing the adaptiveness. We describe this algorithm first since it is employed in both adaptive algorithms presented in this paper.

The basic building block of this algorithm is the *splitter* of Moir and Anderson [27]. A process executing a splitter obtains **down**, **right**, or **stop**. At most one process obtains **stop** and when a single process executes the splitter it obtains **stop**; when two or more processes execute the splitter, not all of them obtain the same value. In this way, the set of processes accessing the splitter is "split" into smaller subsets.

As in [27], splitters are arranged in a grid of size $n \times n$ (Figure 2.1). A process starts at the upper left corner of the grid; the splitters direct the process either to continue (moving right or down in the grid), or to obtain the number associated with the current splitter. The grid spreads the processes so that each process eventually stops in a distinct splitter.

ALGORITHM 1. Adaptive $k(k+1)/2$-renaming.

```
Procedure Adaptive_k(k+1)/2-renaming()
private i, j: integer, initially 0                          // row and column indices
private move: {down, right, stop}, initially down           // direction
1.  while ( move ≠ stop ) do
2.      move := Splitter[i, j]()          // execute splitter in grid position (i, j)
3.      if ( move = down ) then i++                          // increase row
4.      if ( move = right ) then j++                         // increase column
5.  return((i + j)(i + j + 1)/2 + j)      // get a name based on the current splitter

Procedure Splitter[i, j]                                    // from Moir and Anderson [27]
    shared X[i, j]: {⊥} ∪ {0, ..., N − 1}, initially ⊥
    shared Y[i, j]: Boolean, initially false
1.  X[i, j] := id
2.  if ( Y[i, j] ) then return(right)
3.  else Y[i, j] := true
4.      if ( X[i, j] = id ) then return(stop)
5.      else return(down)
```

The difference between the algorithm of Moir and Anderson [27] and our algorithm is that they number splitters by *rows*, while we number splitters by *diagonals*. Splitter (i, j), in row i and column j, $0 \le i < n$ and $0 \le j < n$, is numbered $(i + j)(i + j + 1)/2 + j$. Figure 2.1 shows our numbering; the numbering of Moir and Anderson appears in square brackets.

Algorithm 1 presents pseudocode for the grid and for a splitter.[2]

We say that splitter (i, j) is $i + j$ *steps away* from splitter $(0, 0)$, the top left corner of the grid. As shown in [27, section 3.1], if k processes access the grid then each process stops after $O(k)$ operations in a distinct splitter which is at most $k - 1$ steps away from $(0, 0)$. A simple counting argument shows that these splitters have numbers in the range $\{0, \ldots, k(k + 1)/2 - 1\}$.

THEOREM 2.1. *Algorithm* 1 *solves* $k(k+1)/2$*-renaming with* $O(k)$ *step complexity.*

3. $(2k-1)$-renaming in $O(N)$ operations. As explained in the introduction, the step complexity of adaptive renaming depends on a new linear renaming algorithm, which is neither range-independent nor adaptive.

The algorithm is organized as a network of *reflectors*. A reflector has two distinguished entrances; a process accessing the reflector changes the direction of its movement if another process accessed the reflector, depending on the entrance through which it entered the reflector.

The network consists of N columns, numbered $0, \ldots, N - 1$, from left to right (see Figure 3.1). Column $c = 0, \ldots, N - 1$ contains $2c - 1$ reflectors, numbered $c, c - 1, \ldots, 0, -(c - 1), -c$, from top to bottom. Process q with name c starts at the topmost reflector of column c and descends through column c, until it sees another process accessing the same reflector. Then, q moves left to right towards column $N - 1$; q outputs the row on which it exits column $N - 1$.

[2] Algorithms declare private variables only if their usage in not obvious or their initial value is important.

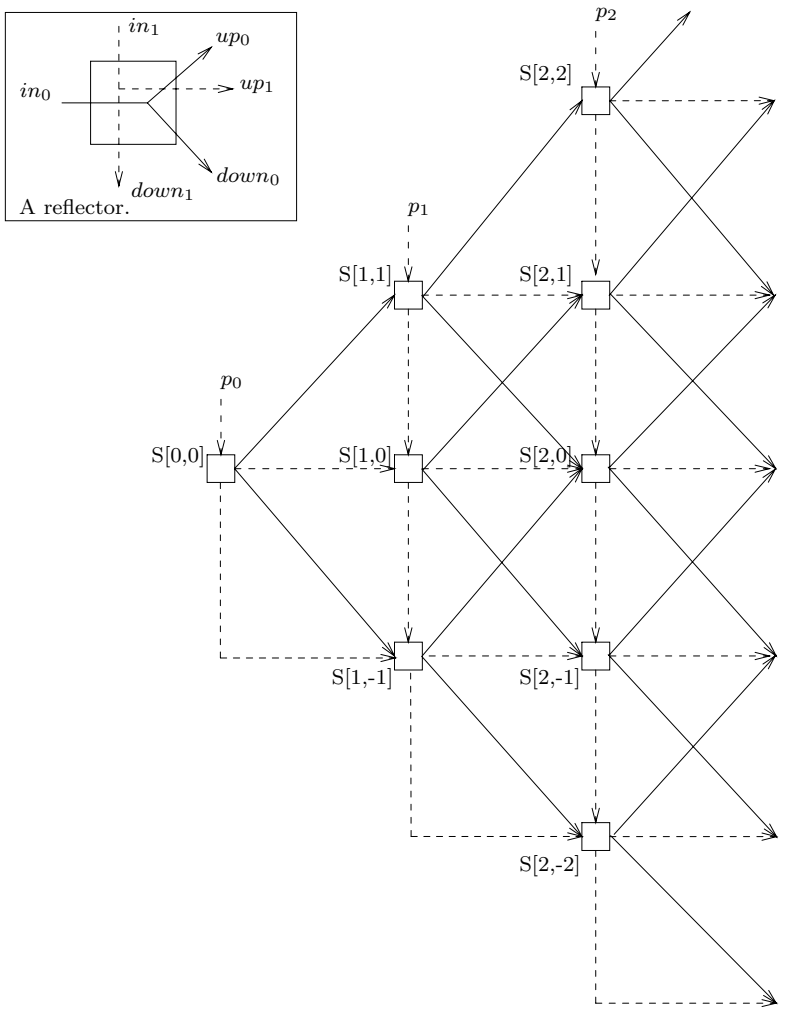


FIG. 3.1. *The network of reflectors for* $(2k-1)$*-renaming (depicted for* $n=3$*).*

For column c, S_{c-1} is the set of processes starting in columns $0,\ldots,c-1$. The main property of the network is that processes in S_{c-1} enter column c on distinct rows among the lowest $2|S_{c-1}|-1$ ones. Therefore, processes in S_{c-1} do not access the same reflectors in column c (or larger); they may interfere only with the single process descending through column c.

Process q descends through column c until it accesses a reflector in row r through which a process in S_{c-1} has passed; then, q moves to column $c+1$, remaining in row r. If process $p \in S_{c-1}$ accesses a reflector which q has passed, then p moves one row up to column $c+1$; if p accesses a reflector which q did not pass, then p moves one row down to column $c+1$. Therefore, processes in S_{c-1} which enter column c on rows $> r$ move one row up; processes in S_{c-1} which enter column c on rows $< r$ move one row down. Process q leaves on one of the free rows between the rows occupied by these two subsets of S_{c-1} (Figure 3.2(c)). Thus, processes in $S_c = S_{c-1} \bigcup \{q\}$ leave column c on distinct rows. Since the new names of the processes are the rows on which they leave the network, they output distinct names.

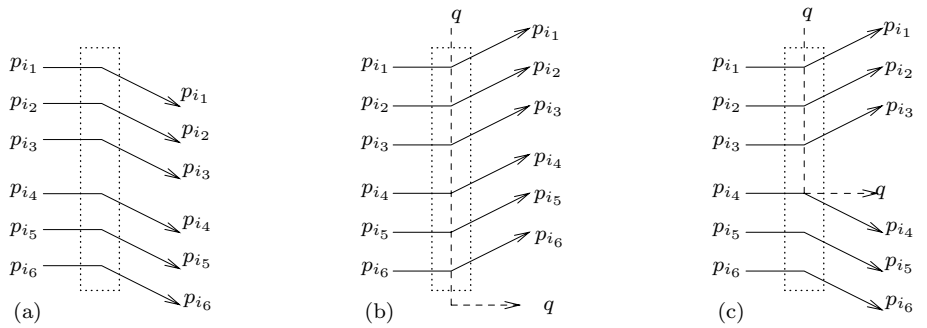


FIG. 3.2. *Illustration for the proof of Lemma* 3.3—*column* $c+1$.

The interaction of processes in column c guarantees that processes in S_{c-1} move to upper rows in column $c+1$ only if q is active; at most two additional rows are occupied (Figure 3.3(b)). If q is not active, then processes leave column c exactly on the same number of rows as they enter (Figure 3.3(a)). Thus, an active process causes at most two rows to be occupied; if there are k active processes, then they leave the network on the lowest $2k-1$ rows.

More formally, a reflector has two *entrances*, in_0 and in_1, two *lower exits*, $down_0$ and $down_1$, and two *upper exits*, up_0 and up_1. A process entering the reflector on entrance in_i leaves the reflector only on exits up_i or $down_i$ (see the top left corner of Figure 3.1). If a single process enters the reflector then it must leave on a lower exit, and at most one process leaves on a lower exit; it is possible that two processes entering the reflector will leave on upper exits. A reflector is easily implemented with two Boolean registers (see Algorithm 2).

The reflectors of column c, denoted $S[c,-c],\ldots,S[c,0],\ldots,S[c,c]$, are connected as follows:

- The upper exit up_0 of $S[c,r]$ is connected to entrance in_0 of $S[c+1,r+1]$.
- The upper exit up_1 of $S[c,r]$ is connected to entrance in_0 of $S[c+1,r]$.
- The lower exit $down_0$ of $S[c,r]$ is connected to entrance in_0 of $S[c+1,r-1]$.
- The lower exit $down_1$ of a reflector $S[c,r]$ is connected to entrance in_1 of reflector $S[c,r-1]$, if $r>-c$; if $r=-c$ (the lowest reflector of column c), then it is connected to entrance in_0 of reflector $S[c+1,r-1]$.

In Algorithm 2, a process with name c starts on entrance in_1 of the upper reflector of column c; it descends through column c (leaving on exit $down_1$) until it sees another

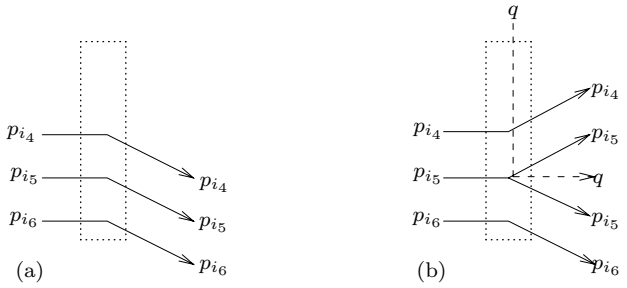


FIG. 3.3. *Illustration for the proof of Lemma* 3.4—*column* $c+1$.

ALGORITHM 2. $(2k-1)$-renaming.

```
Procedure shrink(name : 0, ..., N − 1)                    // renaming algorithm
private col, row: integer, initially name    // start on top reflector of column name
1.  while ( col = name ) do                          // descend through column name
2.      exit := reflector[row,col](1)                                // enter on in_1
3.      if ( exit = up_1 ) then col++
4.      else row−−                                                  // exit = down_1
5.          if ( row < −col ) then col++   // reached the lowest reflector in column
6.  while ( col < N ) do                          // move towards column N − 1
7.      exit := reflector[row, col](0)                               // enter on in_0
8.      if ( exit = up_0 ) then col++; row++;
9.      else col++; row−−;                                          // exit = down_0
10. return(row + N);

Procedure reflector(entrance r : 0,1)
1.  R_r := true ;
2.  if ( R_{1−r} = false ) then return(down_r)
3.  else return(up_r)
```

process or it reaches the bottom of the column. At this point, the process leaves on exit up_1 to the next column, and moves towards column $N-1$; in each column y, it enters exactly one reflector on entrance in_0; it leaves on exit up_0 if it sees another process, or on exit $down_0$, otherwise.

Suppose that p_j enters the reflector on entrance in_i, $i \in \{0,1\}$, and no process enters the reflector on entrance in_{1-i}. Since no process writes to R_{1-i}, p_j reads **false** from R_{1-i} and leaves the reflector on the lower exit, $down_i$. This implies the following lemma.

LEMMA 3.1. *If a single process enters a reflector, then it leaves on a lower exit.*

Similar arguments are used in the proof of the next lemma.

LEMMA 3.2. *If a single process enters a reflector on in_0 and a single process enters the reflector on in_1, then at most one process leaves the reflector on a lower exit.*

Proof. Assume that p_i enters the reflector on in_0 and p_j enters the reflector on in_1. If both processes read **true** from R_1 and R_0, then by the algorithm, $exit(p_i) = up_0$, $exit(p_j) = up_1$, and the lemma holds. Otherwise, without loss of generality, p_i reads **false** from R_1. Since p_i reads **false** from R_1, p_j writes to R_1 in line 1 after p_i reads R_1 at line 2. Therefore, p_j reads R_0 in line 2 after p_i writes to R_0 in line 1. Consequently, p_j obtains **true** from R_0 and by the algorithm, $exit(p_j) = up_1$, which proves the lemma. □

Recall that S_c contains the active processes starting on columns $0, \ldots, c$. For every process $p_i \in S_c$, let $row(p_i, c)$ be the value of the local variable *row* before p_i accesses the first reflector in column $c+1$. The next lemma shows that processes exit a column on distinct rows.

LEMMA 3.3. *For every pair of processes $p_i, p_j \in S_c$, $0 \le c < N$, $row(p_i, c) \neq row(p_j, c)$.*

Proof. The proof is by induction on the column c. In the base case, $c = 0$, the lemma trivially holds since only one process may access a reflector in column 0.

For the induction step, suppose that the lemma holds for column $c \ge 0$; there are

two cases.

Case 1. If no process starts on column $c+1$, then by the algorithm, no reflector in column $c+1$ is accessed on in_1. By Lemma 3.1, every process $p_i \in S_c$ leaves column $c+1$ on exit $down_0$. By the algorithm, we have

$$row(p_i, c+1) = row(p_i, c) - 1$$

(Figure 3.2(a)) and the lemma holds by the induction hypothesis.

Case 2. Suppose that process q starts on column $c+1$. Let $S[c+1, r']$ be the last reflector accessed by q in column $c+1$. That is, q leaves reflectors $S[c+1, c+1], \ldots, S[c+1, r'+1]$ on exits $down_1$; q does not access any of the reflectors $S[c+1, r'-1], \ldots, S[c+1, -(c+1)]$.

By Lemma 3.2, every process $p_i \in S_c$ which enters column $c+1$ on row $r'+1$ or higher, exits column $c+1$ on up_0, and we have

$$row(p_i, c+1) = row(p_i, c) + 1 > r' + 1.$$

By Lemma 3.1, every process $p_i \in S_c$ which enters column $c+1$ on row $r'-1$ or lower, exits column $c+1$ on $down_0$, and we have

$$row(p_i, c+1) = row(p_i, c) - 1 < r' - 1.$$

Now consider process q. By the algorithm, q leaves column $c+1$ either on exit $down_1$ of the lowest reflector in the column, $S[c+1, -(c+1)]$, or on exit up_1 of a reflector $S[c+1, r']$, where $-(c+1) \le r' \le c+1$.

If q leaves reflector $S[c+1, -(c+1)]$ on $down_1$ (Figure 3.2(b)), then by the algorithm,

$$row(q, c+1) = -(c+1) - 1 = r' - 1.$$

If there is a process $p_j \in S_c$ which also accesses $S[c+1, -(c+1)]$, then by Lemma 3.2, p_j leaves $S[c+1, -(c+1)]$ on exit up_0 and therefore,

$$row(p_j, c+1) = -(c+1) + 1 = r' + 1.$$

If q leaves reflector $S[c+1, r']$ on up_1 (Figure 3.2(c)), then by the algorithm, $row(q, c+1) = r'$. By Lemma 3.2, there is a process $p_j \in S_c$ which accesses $S[c+1, r']$; that is, $row(p_j, c) = r'$. By the algorithm

$$row(p_j, c+1) = \begin{cases} r' - 1 & \text{if } p_j \text{ leaves on } down_0, \\ r' + 1 & \text{otherwise.} \end{cases}$$

The induction hypothesis and the above equations imply that in all cases, $row(p_i, c+1) \neq row(p_j, c+1)$, for every pair of processes $p_i, p_j \in S_c$. □

Therefore, processes exit the network on different rows and hence obtain distinct names. The next lemma shows that processes in S_c leave column c on the lowest $2|S_c| - 1$ rows.

LEMMA 3.4. *For every process $p_i \in S_c$, $0 \le c < N$, $-(c+1) \le row(p_i, c) < -(c+1) + 2|S_c| - 1$.*

Proof. The proof is by induction on c. In the base case, $c = 0$. If there is a process p_i such that $id_i = 0$, then by Lemma 3.1, $exit(p_i, 0) = down_1$, since no process accesses reflector $S[0, 0]$ on in_0. Therefore, by the algorithm, we have $row(p_i, 0) = -1$, and the lemma holds.

For the induction step, suppose that the lemma holds for column $c \geq 0$; there are two cases.

Case 1. If no process starts on column $c+1$, then no process accesses reflectors in column $c+1$ on entrance in_1 (Figure 3.3(a)). Therefore, by Lemma 3.1, each process $p_i \in S_c$ exits column $c+1$ on $down_0$, and therefore, $row(p_i, c+1) = row(p_i, c) - 1$. By the induction hypothesis

$$row(p_i, c+1) = row(p_i, c) - 1 \geq -(c+1) - 1 = -((c+1)+1) \ .$$

Also, since $|S_{c+1}| = |S_c|$,

$$row(p_i, c+1) = row(p_i, c) - 1 < -(c+1) + 2|S_c| - 1 - 1 = -((c+1)+1) + 2|S_{c+1}| - 1 \ .$$

The lemma follows from these inequalities.

Case 2. Suppose that process q starts on column $c+1$. By the induction hypothesis, processes from S_c access only the lowest $2|S_c|-1$ reflectors in column $c+1$. Since no process accesses the upper reflectors $S[c+1, c+1], \ldots, S[c+1, -(c+1)+2|S_c|-1]$ on in_0, by Lemma 3.1, q accesses these reflectors until it reaches a reflector $S[c+1, r']$ accessed by another process, or until it reaches the lowest reflector $S[c+1, -(c+1)]$ in the column (Figure 3.3(b)). Therefore, q leaves column $c+1$ either on exit $down_1$ of reflector $S[c+1, -(c+1)]$ or on exit up_1 of a reflector $S[c+1, r']$, where $-(c+1) \leq r' < -(c+1) + 2|S_c| - 1$. By the algorithm, this implies

$$-((c+1)+1) \leq row(q, c+1) < -(c+1) + 2|S_c| - 1$$

since $S_{c+1} = S_c \cup \{q\}$, $|S_{c+1}| = |S_c| + 1$. Therefore,

$$-((c+1)+1) \leq row(q, c+1) < -((c+1)+1) + 2|S_{c+1}| - 2. \tag{3.1}$$

According to the algorithm, for each process $p_i \in S_c$,

$$row(p_i, c+1) = \begin{cases} row(p_i, c) + 1 & \text{if } exit(p_i, c+1) = up_0, \\ row(p_i, c) - 1 & \text{otherwise.} \end{cases}$$

Together with the induction hypothesis, this implies

$$row(p_i, c+1) \geq row(p_i, c) - 1 \geq -(c+1) - 1 = -((c+1)+1). \tag{3.2}$$

Also,

$$\begin{aligned} row(p_i, c+1) &\leq row(p_i, c) + 1 \\ &< -(c+1) + 2|S_c| - 1 + 1 \\ &= -((c+1)+1) + 1 + 2(|S_{c+1}| - 1) - 1 + 1 \\ &= -((c+1)+1) + 2|S_{c+1}| - 1. \end{aligned} \tag{3.3}$$

The lemma follows from inequalities (3.1), (3.2), and (3.3). □

Lemma 3.4 implies that processes leave the network on rows $-N, \ldots, -N + 2|S_{N-1}| - 2$. Since $|S_{N-1}| \leq k$, the names chosen in line 10 are in the range $\{0, \ldots, 2k-2\}$.

Process p_j accesses at most $2id_j + 1$ reflectors in column id_j, and exactly one reflector in each column $id_j + 1, \ldots, N-1$. Each reflector requires a constant number of operations, implying the next theorem.

THEOREM 3.5. *Algorithm 2 solves $(2k-1)$-renaming with step complexity $O(N)$.*

The network consists of $O(N^2)$ reflectors; each reflector is implemented with two registers. Register R_i of a reflector is written only by a process entering the reflector on entrance in_i. Entrance in_1 of a reflector is accessed only by the single process starting on this column, and entrance in_0 is accessed by at most one process (by Lemma 3.3). Therefore, we use $O(N^2)$ *dynamic* single-writer single-reader registers.

4. $(2k-1)$-renaming in $O(n \log n)$ operations. A range-independent $(2k-1)$-renaming can be obtained by combining adaptive $O(k^2)$-renaming and nonadaptive $(2k-1)$-renaming. First, the names are reduced into a range of size $O(n^2)$ (Algorithm 1); these names are used to enter the shrinking network of Algorithm 2, which reduces them into a range of size $(2k-1)$. The shrinking network is started with names of size $O(n^2)$, and hence, the step complexity of this simple algorithm is $O(n^2)$. The algorithm presented in this section obtains $O(n \log n)$ step complexity by reducing the name space gradually in $O(\log n)$ iterations. To do so, distinct copies of shrink (Algorithm 2) are associated with the vertices of a complete binary tree of height $\lceil \log n(n+1)/2 \rceil - 1 \approx 2 \log n$ (Figure 4.1). Each copy of shrink is designed for names in a range of size $4n-2$; that is, it employs a network with $4n-2$ columns.

A process starts Algorithm 3 by acquiring a name using $O(k^2)$-renaming; this name determines from which leaf to start. The process performs the shrinking network associated with each vertex v on the path from the leaf to the root, starting at a column which is determined by the name obtained at the previous vertex: If it ascends from the left subtree of v, then it starts at one of the first $2n-1$ columns of the network; otherwise, it starts at one of the last $2n-1$ columns. The process outputs the name obtained at the root.

The vertices of the tree are numbered in BFS order (Figure 4.1): The root is numbered 1; if vertex v is numbered ℓ, then its left child is numbered 2ℓ, and its right child is numbered $2\ell+1$. The copy of Algorithm 2 associated with a vertex numbered ℓ is denoted shrink$[\ell]$.

LEMMA 4.1. *For every vertex v, processes executing* shrink$[v]$ *obtain distinct temporary names in the range $\{0, \ldots, 2k-2\}$.*

Proof. The proof is by induction on d, the height of v. In the base case, $d = 0$. After executing Algorithm 1, processes get distinct names in the range $\{0, \ldots, k(k+1)/2-1\}$. Therefore, at most one process accesses v from the left executing shrink$[v]$ with temporary name 0, and at most one process accesses v from the right, executing shrink$[v]$ with temporary name $2n-1+0 = 2n-1$. Thus, they execute shrink$[v]$ with different temporary names in the range $\{0, \ldots, 4n-3\}$. Theorem 3.5 implies

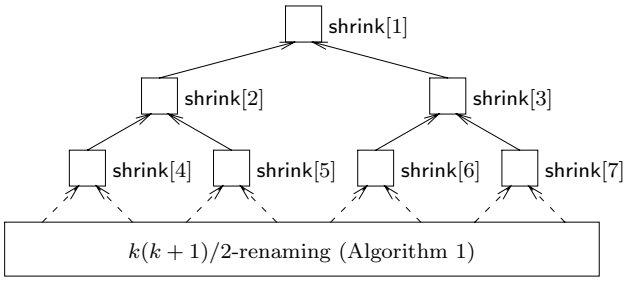


FIG. 4.1. *The range-independent algorithm for $(2k-1)$-renaming (depicted for $n = 3$).*

ALGORITHM 3. Range-independent $(2k-1)$-renaming for n processes.

```
Procedure indRenaming_n()
1.  temp-name := Adaptive_k(k+1)/2-renaming()                    // Algorithm 1
2.  ℓ := 2^h + ⌊temp-name/2⌋   // h = ⌈log n(n+1)/2⌉ − 1 is the height of the tree
3.  side := temp-name mod 2
4.  temp-name := 0
5.  while ( ℓ ≥ 1 ) do
6.      temp-name := shrink[ℓ](temp-name + side · (2n − 1))
7.      side := ℓ mod 2
8.      ℓ := ⌊ℓ/2⌋
9.  return(temp-name)
```

they obtain distinct names in the range $\{0, 1, 2\}$ and therefore, the lemma holds when $d = 0$.

For the induction step, assume the lemma holds for vertices at height d, and let v be a vertex at height $d + 1$. By the induction hypothesis and by the algorithm, processes accessing v from the left child have distinct temporary names in the range $\{0, \ldots, 2k-2\}$, and processes accessing v from the right child have distinct names in the range $\{2n-1, \ldots, 2n+2k-3\}$. Thus, processes execute shrink$[v]$ with distinct names in the range $\{0, \ldots, 4n-3\}$, and obtain distinct names in the range $\{0, \ldots, 2k-2\}$, by Theorem 3.5. □

Therefore, processes obtain distinct names in the range $\{0, \ldots, 2k-2\}$ after completing shrink at the root. A process performs shrink in $h = O(\log n)$ vertices of the tree, and each vertex requires $O(n)$ operations (Theorem 3.5). This implies the following theorem.

THEOREM 4.2. *Algorithm* 3 *solves* $(2k-1)$*-renaming with* $O(n \log n)$ *step complexity.*

5. $(6k-1)$-renaming in $O(k \log k)$ operations. In our adaptive $(6k-1)$-renaming algorithm, a process estimates the number of active processes and performs a copy of the range-independent $(2k-1)$-renaming algorithm (Algorithm 3) designed for this number. Processes may have different estimates of k, the number of active processes, and perform different copies of Algorithm 3. Instead of consolidating the names obtained in the different copies, disjoint name spaces are allocated to the copies.

The number of active processes is estimated by the size of a view obtained from lattice agreement; since views are comparable, the estimate is within a constant factor (see Lemma 5.1).

In Algorithm 4, process p_i belongs to a set S_j if the size of its view is in $(2^{j-1}, 2^j]$. For views obtained in lattice agreement, this partition guarantees that $|S_j| \leq 2^j$, for $j \leq \lceil \log n \rceil$; moreover, if the number of active processes is k, then $|S_j| = 0$, for $j > \lceil \log k \rceil$. There are $\lceil \log n \rceil + 1$ copies of Algorithm 3, denoted indRenaming$_{2^0}$, ..., indRenaming$_{2^{\lceil \log n \rceil}}$. Processes in S_j perform indRenaming$_{2^j}$, designed for 2^j processes, and obtain names in a range of size $2|S_j| - 1$. The name spaces for $S_0, \ldots, S_{\lceil \log n \rceil}$ do not overlap, and their size is linear k (Figure 5.1).

LEMMA 5.1. *If the views of processes in a set* S *satisfy the comparability and self-inclusion properties of lattice agreement, and the size of a view is at most* k*, then* $|S| \leq k$.

Proof. Assume V is the view with maximal size in S. Let V_i be the view of some process $p_i \in S$. The self-inclusion property implies that $p_i \in V_i$, and the comparability

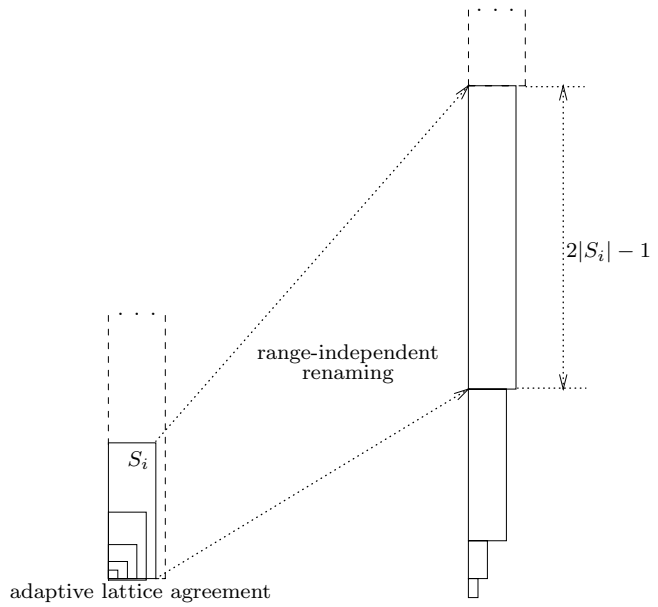


FIG. 5.1. *Adaptive* $(6k-1)$*-renaming.*

ALGORITHM 4. Adaptive $(6k-1)$-renaming.

```
1.  V := AdaptiveLA()                                  // Algorithm 5, presented below
2.  r := ⌈log |V|⌉
3.  temp-name := indRenaming_{2^r}()                   // Algorithm 3
4.  if ( r = 0 ) then return(temp-name)
5.  else return(temp-name + 2^{r+1})
```

property implies that $V_i \subseteq V$. Therefore, $S \subseteq V$, implying that $|S| \leq |V| \leq k$. □

By the algorithm, if process p_i is in S_j, then $|V_i| \leq 2^j$. Lemma 5.1 implies the next lemma.

LEMMA 5.2. *If there are* k *active processes, then* $|S_j| \leq 2^j$*, for* $0 \leq j \leq \lceil \log k \rceil$.

For every process p_i, $|V_i| \leq k$, since the views contain only active processes. Therefore, $p_i \in S_j$ only if $0 \leq j \leq \lceil \log k \rceil$, which implies the next lemma.

LEMMA 5.3. *If there are* k *active processes, then* $|S_j| = 0$ *for* $j > \lceil \log k \rceil$.

By Lemma 5.2, at most 2^j processes invoke $\mathsf{indRenaming}_{2^j}$. Therefore, process p_i invoking $\mathsf{indRenaming}_{2^j}$ obtains $\textit{temp-name}_i \in \{0, \ldots, 2 \cdot 2^j - 2\}$, by Theorem 4.2. By the algorithm, p_i returns $\textit{temp-name}_i + 2^{j+1} \in \{2^{j+1}, \ldots, 2^{j+2} - 2\}$.

The set of names returned by processes performing $\mathsf{indRenaming}_{2^j}$ is denoted $\textit{NameSpace}_j$; the next lemma follows from the algorithm.

LEMMA 5.4. (1) $\textit{NameSpace}_i \bigcap \textit{NameSpace}_j = \emptyset$*, for every* i *and* j*,* $0 \leq i < j \leq \lceil \log n \rceil$.

(2) $\bigcup_{i=0}^{m} \textit{NameSpace}_i \subseteq \{0, \ldots, 4 \cdot 2^m - 2\}$ *for every* $m \leq \lceil \log n \rceil$.

LEMMA 5.5. *If there are* k *active processes, then they return distinct names in the range* $\{0 \ldots 2^{\lceil \log k \rceil + 1} + 2k - 2\}$.[3]

[3]There are $\lceil \log k \rceil + 1$ names of the form $2^{j+2} - 1$, $0 \leq j \leq \lceil \log k \rceil$, which are not used. Therefore, the names obtained in the algorithm can be mapped into a name space of size $6k - \lceil \log k \rceil - 2$.

Proof. If two active processes, p_i and p_j, execute the same copy of indRenaming, then they obtain distinct names by Theorem 4.2; otherwise, they obtain distinct names, by Lemma 5.4(1).

By Lemma 5.3, processes invoke $\mathsf{indRenaming}_{2^j}$ only if $0 \leq j \leq \lceil \log k \rceil$. By Lemma 5.4(2), processes invoking $\mathsf{indRenaming}_{2^j}$, for $0 \leq j < \lceil \log k \rceil$, return names in the range $\{0, \ldots, 2^{\lceil \log k \rceil + 1} - 2\}$. By Theorem 4.2, a process p_i invoking the last nonempty copy $\mathsf{indRenaming}_{2^{\lceil \log k \rceil}}$ obtains a temporary name in the range $\{0, \ldots, 2k - 2\}$. By the algorithm, p_i returns a name in the range $\{2^{\lceil \log k \rceil + 1}, \ldots, 2^{\lceil \log k \rceil + 1} + 2k - 2\}$. Thus, the output names are in a range whose size is not greater than $2^{\lceil \log k \rceil + 1} + 2k - 1$. ∎

Since $6k - 1 \leq 2^{\lceil \log k \rceil + 1} + 2k - 1$, the correctness of the algorithm follows from Lemma 5.5.

If there are k active processes, then each process performs AdaptiveLA (presented in the next section) in $O(k \log k)$ operations. By Lemma 5.3, only copies of indRenaming for less than $2k$ processes are invoked. Therefore, a process completes indRenaming in $O(k \log k)$ operations.

THEOREM 5.6. *Algorithm* 4 *solves* $(6k-1)$*-renaming with* $O(k \log k)$ *step complexity.*

The upper bound on the size of the name space, $6k - 1$, is tight for Algorithm 4. Assume that all processes executing lattice agreement obtain the maximal view (with size k) and access $\mathsf{indRenaming}_{2^{\lceil \log k \rceil}}$. The processes leave the range $\{0, \ldots, 2^{\lceil \log k \rceil + 1} - 2\}$ unused (since it is unknown whether the previous copies of indRenaming are empty or not) and return names in the range $\{2^{\lceil \log k \rceil + 1}, \ldots, 2^{\lceil \log k \rceil + 1} + 2k - 2\}$. If k is not an integral power of 2, then the output names are in a range of size $\leq 2^{\log k + 2} + 2k - 1 = 6k - 1$. If k is an integral power of 2, then the output names are in a range of size $2^{\log k + 1} + 2k - 1 = 4k - 1$.

Merritt (private communication) noted that the names can be reduced by partitioning the active processes into sets of size $a^0, \ldots, a^j, \ldots$ for an integer $a > 2$. Active processes are partitioned into sets $S_0, \ldots, S_j, \ldots$; processes in S_j execute $\mathsf{adaptiveRenaming}_{a^j}$ designed for a^j participants and obtain new names in a range of size $2|S_j| - 1$. As in our algorithm, when k processes are active, $\mathsf{adaptiveRenaming}_{a^j}$ is accessed only for $0 \leq j \leq \lceil \log_a k \rceil$. Processes accessing copies $\mathsf{adaptiveRenaming}_{a^j}$, $0 \leq j < \lceil \log_a k \rceil$, obtain names in a space of size $\sum_{j=0}^{\lceil \log_a k \rceil - 1} (2a^j - 1) \leq 2\frac{a^{\lceil \log_a k \rceil} - 1}{a-1} \leq 2\frac{a^{\log_a k + 1} - 1}{a-1} = 2\frac{ka-1}{a-1}$, which tends to $2k$ when $a \to \infty$. Processes accessing the last nonempty copy, $\mathsf{adaptiveRenaming}_{a^{\lceil \log_a k \rceil}}$, obtain new names in a range of size $2k - 1$. Thus, the size of the total name space is $\leq 4k - 1$ when $a \to \infty$.

6. Lattice agreement in $O(k \log k)$ operations. Our lattice agreement algorithm is based on the algorithm of Inoue et al. [23]. In their algorithm, each process starts at a distinct leaf (based on its name) of a complete binary tree with height $\lceil \log N \rceil - 1$, and climbs up the tree to the root. At each vertex on the path, it performs a procedure which merges together two sets of views, each set containing only comparable views; this procedure is called union. At the leaf, the process uses its own name as input to union; at the inner vertices, it uses the view obtained in the previous vertex as input to union. The process outputs the view it obtains at the root.

Specifically, union takes two parameters, an input view V and an integer $side \in \{0, 1\}$, and returns an output view; its properties are specified by the next lemma [23, Lemma 6].

LEMMA 6.1. *If the input views of processes invoking* union *with* $side = 0$ *are*

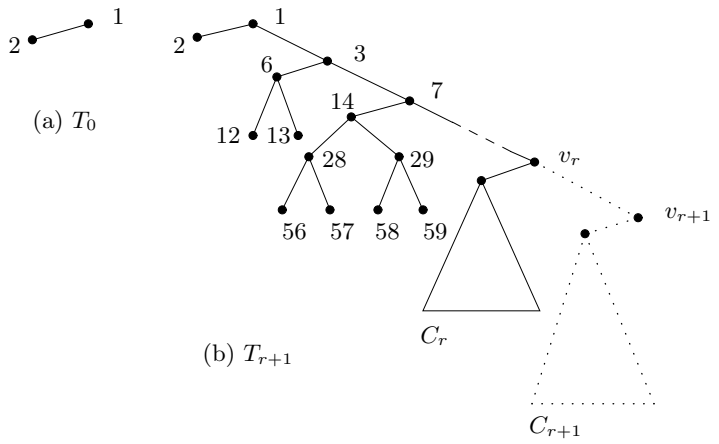


FIG. 6.1. *The unbalanced binary tree used in the adaptive lattice agreement algorithm.*

comparable and satisfy the self-inclusion property, and similarly for the input views of processes invoking union *with side* $= 1$*, then*

(1) *the output views of processes exiting* union *are comparable, and*

(2) *the output view of a process exiting* union *contains its input view.*

Appendix A describes union in detail, and explains the next lemma.

LEMMA 6.2. *The step complexity of* union *is* $O(k)$.

Our adaptive algorithm uses an unbalanced binary tree T_r defined inductively as follows. T_0 has a root v_0 with a single left child (Figure 6.1(a)). For $r \geq 0$, suppose T_r is defined with an identified vertex v_r, which is the last vertex in an in-order traversal of T_r; notice that v_r does not have a right child in T_r. T_{r+1} is obtained by inserting a new vertex v_{r+1} as the right child of v_r, and inserting a complete binary tree C_{r+1} of height $r+1$ as the left subtree of v_{r+1} (Figure 6.1(b)). By the construction, v_{r+1} is the last vertex in an in-order traversal of T_{r+1}.

The vertices of the tree are numbered as follows: The root is numbered 1; if a vertex is numbered ℓ, then its left child is numbered 2ℓ, and its right child is numbered $2\ell+1$ (Figure 6.1).

By the construction, the leaves of T_r are the leaves of the complete binary subtrees $C_0, C_1, \ldots, C_r$. Therefore, the total number of leaves in T_r is $\sum_{j=0}^{r} 2^j = 2^{r+1} - 1$. The following simple lemma, proved in Appendix B, states some properties of T_r.

LEMMA 6.3. *Let* w *be the* i*th leaf of* T_r*,* $1 \leq i \leq 2^{r+1} - 1$*, counting from left to right. Then*

(1) *the depth of* w *is* $2\lfloor \log i \rfloor + 1$*;*

(2) w *is numbered* $2^d(2^{d+2} - 3) + i$*, where* $d = \lfloor \log i \rfloor$.

Algorithm 5 uses $T_{2\log n - 1}$, which has $n^2 - 1$ leaves.[4] A process starts the algorithm by obtaining a new name in a range of size $k(k+1)/2$ (using Algorithm 1). This name determines the leaf at which the process starts to climb up the tree: A process with a new name x_i starts the algorithm at the x_ith leaf of the tree, counting from left to right. Since $k(k+1)/2 \leq n^2 - 1$ for all $n \geq 2$, $T_{2\log n - 1}$ have enough leaves for temporary names in a range of size $k(k+1)/2$. By Lemma 6.3, the x_ith leaf is numbered $2^d(2^{d+2} - 3) + x_i$, where $d = \lfloor \log x_i \rfloor$.

[4]For simplicity, we assume n is a power of 2.

ALGORITHM 5. Adaptive lattice agreement.

```
Procedure AdaptiveLA()
1.  temp-name := Adaptive_k(k+1)/2-renaming()+1          // Algorithm 1
2.  d := ⌊log temp-name⌋
3.  ℓ := 2^d(2^{d+2} − 3) + temp-name      // the leaf corresponding to temp-name
4.  V := {p_i}                                  // the input is the process's name
5.  while ( ℓ ≥ 1 ) do
6.      side := ℓ mod 2                                          // calculate side
7.      ℓ := ⌊ℓ/2⌋                                             // calculate father
8.      V := union[ℓ](V, side)
9.  return(V)
```

As in [23], a distinct copy of union is associated with each inner vertex of the tree. A process performs copies of union associated with the vertices along its path to the root, and returns the view obtained at the root.

Simple induction on the distance of a vertex v from the leaves shows that the views obtained by processes executing union at v satisfy the comparability and self-inclusion properties. In the base case, v is a leaf and the claim is trivial since a single process starts at each leaf; the induction step follows immediately from Lemma 6.1. Hence, the views obtained at the root have the lattice agreement properties.

If there are k active processes, process p_i gets a unique name $x_i \in \{1, \ldots, k(k+1)/2\}$ (line 1) and by Lemma 6.3(2), starts in a leaf ℓ of depth $2\lfloor \log x_i \rfloor + 1$ (line 3). Therefore, p_i accesses at most $2\lfloor \log x_i \rfloor + 1 \leq 2\lfloor \log k(k+1)/2 \rfloor + 1 \leq 4 \log k + 1$ vertices. At each vertex, the execution of union requires $O(k)$ operations (Lemma 6.2). Thus, the total step complexity of the algorithm is $O(k \log k)$, implying the following theorem.

THEOREM 6.4. *Algorithm 5 solves lattice agreement with $O(k \log k)$ step complexity.*

7. Discussion. This work presents *adaptive* wait-free algorithms, whose step complexity depends only on the number of active processes, for lattice agreement and $(6k-1)$-renaming in the read/write asynchronous shared-memory model; the step complexity of both algorithms is $O(k \log k)$.

Clearly, the complexities of our algorithms—the number of steps, the number and the size of registers used—can be improved. For example, an algorithm for $O(k)$-renaming with $O(k)$ step complexity would immediately yield a lattice agreement algorithm with the same step complexity. Also it would be interesting to see if ideas from our efficient algorithms can improve the complexities of algorithms which adapt to the current contention [2, 6].

Appendix A. Procedure union.

Procedure union (from [23]) takes two parameters, a view V and an integer $side \in \{0, 1\}$, and returns a view. Let S_b be the set of processes which call union with side b. If the input views of the processes in S_0 are comparable and the input views of the processes in S_1 are comparable (within each set separately), then *all* the views returned by union are comparable. In addition, the output view of each process contains its input view.

The procedure uses two shared arrays, $Views_0[1, \ldots, n+1]$ and $Views_1[1, \ldots, n+1]$, whose entries contain views (initially empty). Processes use two procedures:

WriteSet which writes a view into an array, and ReadSet which reads the view of maximal size written in an array.

A process invoking union with side b, first writes its input view V into $Views_b$ using WriteSet. Then the process repeatedly reads from the arrays $Views_0$ and $Views_1$ using ReadSet. If the last two reads from $Views_0$ return the same view, and the last two reads from $Views_1$ return the same view, then the process returns the union of these views. As in Afek et al. [1], if two nonoverlapping ReadSet operations return the same maximal view, then the maximal view does not change in between the operations. Since reads from $Views_0$ alternate with reads from $Views_1$, there is an execution interval during which $Views_0$ and $Views_1$ do not change. Therefore, the union of the last views read from the arrays is a snapshot of the maximal size views written in them. If the views written in each array are comparable, then the snapshots of the maximal views, returned by union, are also comparable.

The crux of the algorithm is how ReadSet and WriteSet guarantee linear step complexity. For $b = 0, 1$, the ℓth entry of $Views_b$ corresponds to the unique view of size ℓ in S_b. A process accesses each entry of $Views_b$ at most once. Towards this end, if a process reads a view V in ReadSet from an entry $i < |V|$, then it writes V into entries $i+1, \ldots, |V|$; this guarantees that V will not be overwritten by a smaller view and will be available for other processes. Furthermore, if the maximal size of views in S_b is k, then a process accesses only the first $k+1$ entries of $Views_b$. Thus, a process performs at most $O(|S_0 \bigcup S_1|)$ reads and writes.

The pseudocode appears in Algorithm 6.

In WriteSet(A, V), a process writes V to $A[1], A[2], \ldots, A[|V|]$ (in this order). When there are k active processes, $|V| \leq k$, and hence no process writes to $A[k+1]$.

In ReadSet(A, V), process p_i reads the entries of array A in increasing order of indices, until it reads $\emptyset$; by the property of WriteSet, p_i obtains $\emptyset$ after at most $k+1$ reads. If p_i reads a view W from $A[j]$, $j \leq |W|$, then p_i writes W to the array in the order $A[j+1], A[j+2], \ldots, A[|W|]$ (lines 4–6) and resumes reading from $A[|W|+1]$ (lines 7–11). This requires ReadSet to take an additional parameter V, the maximal view p_i has written or read so far.

Let V_i be the view returned by process p_i. Clearly, the step complexity of union is linear in $|V_i|$ (see [23, Lemma 9]). Since $V_i \subseteq S_0 \cup S_1$, the step complexity of union is $O(|S_0 \cup S_1|) = O(k)$, which implies Lemma 6.2.

Appendix B. Proof of Lemma 6.3.

(1) Suppose that w belongs to a C_d, $d \leq r$. The total number of leaves in $C_0, \ldots, C_{d-1}$ is $\sum_{j=0}^{d-1} 2^j = 2^d - 1$. Therefore, $2^d - 1 < i$. Since w belongs to C_d, $i \leq \sum_{j=0}^{d} 2^j = 2^{d+1} - 1$. Thus, $2^d \leq i < 2^{d+1}$, implying that $d = \lfloor \log i \rfloor$.

Consider the path from the root of T_r to w. The path has length $2d+1$: d edges on the path from the root of T_r to v_d, one edge connecting v_d and the root of C_d, and d edges on the path from the root of C_d to w. Since $d = \lfloor \log w_i \rfloor$, the length of the path is $2\lfloor \log i \rfloor + 1$.

(2) Define a binary string $b[1 \ldots 2d+1]$, which represents the path from the root of T_r to w, as follows: $b[1] = 1$; for any $1 \leq j < 2d+1$, $b[j+1] = 0$, if the $(j+1)$st vertex on the path is the left child of its parent; and $b[j+1] = 1$, otherwise.

The numbering of the vertices in T_r implies that the number of w is represented by the binary string $b[1 \ldots 2d+1]$. The d leftmost bits of b, corresponding to the path from the root of the tree to vertex v_d are $11 \ldots 1$, since all the vertices on the path are right children (see Figure 6.1(b)). The $(d+1)$st bit of b is 0, since the root of C_d

ALGORITHM 6. Procedure union (from [23]).

```
Procedure union(V: view, b: 0,1)                          // side parameter is b
1.  WriteSet(Views_b,V)
2.  current_b := V ; current_{1-b} := ∅
3.  repeat
4.      prev_0 := current_0 ; prev_1 := current_1
5.      current_0 := ReadSet(Views_0,prev_0)
6.      current_1 := ReadSet(Views_1,prev_1)
7.  until ( prev_0 = current_0 and prev_1 = current_1 )
8.  return(current_0 ∪ current_1)

Procedure WriteSet(A: array of registers, V: view)
1.  for j = 1 to |V| do A[j] := V

Procedure ReadSet(A: array of registers, prev: view)
1.  max := prev
2.  p := |max|
3.  while ( p ≤ |max| ) do
4.      for j = p + 1 to |max| do A[j] := max
5.      p := |max|
6.      repeat
7.          p := p + 1
8.          temp := A[p]
9.          if ( |temp| > p ) then max := temp
10.     until ( |max| > p or temp = ∅ )
11. return(max)
```

is the left child of v_d. It remains to calculate the d rightmost bits of b, corresponding to the path from the root of C_d to w.

The total number of leaves in $C_0, \ldots, C_{d-1}$ is $\sum_{j=0}^{d-1} 2^j = 2^d - 1$. Since w is the ith leaf in T_r, it is the $(i - (2^d - 1))$st leaf in C_d, counting from 1. Therefore, the d bits corresponding to the path from the root of C_d to w are the binary representation of the number $(i - (2^d - 1)) - 1$, that is, $(i - 2^d)_2$. Hence,

$$b[1 \ldots 2d+1] = \overbrace{11\ldots1}^{d}0 \cdot (i - 2^d)_2.$$

Therefore, the number of w is $(2^{d+1} - 1)2^{d+1} + i - 2^d = 2^d(2^{d+2} - 3) + i$.

Appendix C. Linear atomic snapshots.

This section presents a linear-time algorithm for lattice agreement using only dynamic single-writer single-reader registers; this algorithm is neither range-independent nor adaptive. It appears here because it is a simple modification of Algorithm 2, and it shows yet another interesting connection between renaming and lattice agreement.

The algorithm uses the network of reflectors of Figure 3.1. In the modified algorithm, reflectors help the processes to collect views, in addition to directing their movements. The modified reflector is implemented with two N-bit registers, R_0 and R_1, that can contain views. Initially, each register contains the empty view, $\emptyset$. A process entering the reflector on entrance in_i writes its local view into register R_i,

ALGORITHM 7. Linear lattice agreement.

```
Procedure LatticeAgreement(name_i : 0, ..., N − 1)
private col, row: integer, initially name_i  // start on top reflector of column name_i
private V, initially {p_i}                                          // the initial view
1.  while ( row = name_i ) do                  // descend through column name_i
2.      ⟨V, exit⟩ := reflector[row,col](V,1)                        // enter on in_1
3.      if ( exit = up_1 ) then col++;
4.      else                                                        // exit = down_1
5.          row−−;
6.          if ( row < −col ) then
7.              col++;                  // reached the lowest reflector in column
8.  while ( col < N ) do                       // move towards column N − 1
9.      ⟨V, exit⟩ := reflector[row,col](V,0)                        // enter on in_1
10.     if ( exit = up_0 ) then col++; row++;
11.     else col++; row−−;                                          // exit = down_0
12. return(V);

Procedure reflector(view V, entrance r: 0,1)
1.  R_r := V ;
2.  if ( R_{1−r} = ∅ ) then return(⟨V, down_r⟩)
3.  else return(⟨V ∪ R_{1−r}, up_r⟩)
```

and then reads the other register, R_{1-i}. If it is $\emptyset$, then the local view of the process does not change, and the process exits on $down_i$; otherwise, the process joins the view written in R_{1-i} with its local view and exits on up_i.

The algorithm which controls the processes' movements in the network remains exactly the same as in Algorithm 2. Processes leaving the network return their local views (instead of the row numbers, as in the renaming algorithm). Pseudocode appears in Algorithm 7; the numbers of the modified lines appear in bold.

For a process p_i and a reflector S, $V_{in}(p_i, S)$ and $V_{out}(p_i, S)$ are the views of p_i before and after accessing S, respectively. The following lemma is a modification of Lemma 3.1. The only difference is related to the process' view after accessing a reflector. By the algorithm, the local view of the process which reads $\emptyset$ from R_{1-i} does not change.

LEMMA C.1. *If a single process p_j enters a reflector S then p_j leaves S on a lower exit and $V_{out}(p_i, S) = V_{in}(p_i, S)$.*

The proof of the following lemma is similar to Lemma 3.2.

LEMMA C.2. *If a single process p_i enters a reflector S on in_0, and a single process p_j enters S on in_1, then*

(1) *at most one of the processes leaves S on a lower exit;*

(2) *if $p \in \{p_i, p_j\}$ goes down, then $V_{out}(p, S) = V_{in}(p, S)$;*

(3) *if $p \in \{p_i, p_j\}$ goes up, then $V_{out}(p, S) = V_{in}(p_i, S) \cup V_{in}(p_j, S)$.*

Recall that S_c is the set of processes which start execution on columns $0, \ldots, c$; let $V(p_i, c)$ be the view with which a process p_i exists column c. The next lemma shows that the views of processes exiting column c are comparable.

LEMMA C.3. *For every pair of processes $p_i, p_j \in S_c$, $0 \le c < N$, if $row(p_i, c) < row(p_j, c)$ then $V(p_i, c) \subseteq V(p_j, c)$.*

Proof. The proof is by induction on the column c. The lemma holds in the base

case, $c = 0$, since at most one process exits column 0. Suppose that the lemma holds for $c \geq 0$.

If there is no active process with name equal to $c+1$, then the reflectors in column $c+1$ are accessed only on entrances in_0. Therefore, by Lemma C.1, the local views of the processes S_c do not change in column $c+1$, and for every process $p_j \in S_c$, $row(p_j, c+1) = row(p_j, c) - 1$. In this case, the lemma holds by the induction hypothesis.

Otherwise, let q be an active process which starts on column $c+1$. Suppose that the last reflector accessed by q on column $c+1$ is $S[c+1, r']$, $-(c+1) \leq r' \leq c+1$. That is, q leaves reflectors $S[c+1, c+1], \ldots, S[c+1, r'+1]$ on $down_1$, and q does not access reflectors $S[c+1, r'-1], \ldots, S[c+1, -(c+1)]$. By Lemmas C.1 and C.2,

- for all $p_j \in S_c$ such that $row(p_i, c) < r'$

$$\begin{aligned} exit(p_i, c+1) &= down_0, \\ row(p_i, c+1) &= row(p_i, c) - 1, \\ V(p_i, c+1) &= V(p_i, c); \end{aligned} \tag{C.1}$$

- for all $p_j \in S_c$ such that $row(p_i, c) > r'$

$$\begin{aligned} exit(p_i, c+1) &= up_0, \\ row(p_i, c+1) &= row(p_i, c) + 1, \\ V(p_i, c+1) &= V(p_i, c) \cup \{q\}. \end{aligned} \tag{C.2}$$

By the algorithm, q leaves column $c+1$ either on exit $down_1$ of the lowest reflector $S[c+1, -(c+1)]$ in the column, or on exit up_1 of $S[c+1, r']$, where $-(c+1) \leq r' \leq c+1$.

If q leaves reflector $S[c+1, -(c+1)]$ on $down_1$, then by Lemma C.1, $row(q, c+1) = -(c+1) - 1$ and $V(q, c+1) = V(q, c)$. The lemma holds by the induction hypothesis and (C.2).

If q leaves reflector $S[c+1, r']$ on up_1, then there is a process $p_j \in S_c$ such that $row(p_j, c) = r'$, by Lemma C.1. By Lemma C.2, $V(q, c+1) = V(p_j, c) \cup \{q\}$. If p_j leaves column $c+1$ on up_0, then $V(p_j, c+1) = V(p_j, c) \cup \{q\}$ and $row(p_j, c+1) = row(p_j, c) + 1 = r' + 1$, by Lemma C.2. If p_j leaves column $c+1$ on $down_0$, then $V(p_j, c+1) = V(p_j, c)$ and $row(p_j, c+1) = r' - 1$, by Lemma C.2. In both cases, the lemma holds by the induction hypothesis, (C.1), and (C.2). □

Lemma C.3 with $c = N - 1$ implies that the local views of the processes leaving the network are comparable. Since p_i initial view contains its identifier and the local view never decreases, we have the self-inclusion property. The step complexity of the algorithm is calculated as in Theorem 3.5. The network contains $O(N^2)$ reflectors; each reflector uses two bounded dynamic single-writer single-reader registers. This proves the following theorem.

THEOREM C.4. *Algorithm 7 solves the lattice agreement problem with step complexity $O(N)$ using $O(N^2)$ dynamic single-writer single-reader registers of length N bits.*

Applying the transformation of [15], which requires $O(N)$ additional operations on single-writer multi-reader registers, we get the following result.

THEOREM C.5. *There is an implementation of an atomic snapshot object such that every operation on the object requires $O(N)$ read/write operations on dynamic single-writer multi-reader registers. The implementation uses $O(N^2)$ registers of length N bits.*

LEMMA C.6. *If the processes executing* RestrictedLA$_{2^j}$ *have distinct names in the range* $\{0, \ldots, 2^j - 1\}$*, then the following conditions hold for every vertex* v*:*

(1) *the output views of processes exiting* union *at* v *are comparable, and*

(2) *the output view of a process exiting* union *at* v *contains its input view.*

Proof. Assume the vertex v is numbered ℓ. Let $InView(v, p)$ and $OutView(v, p)$ be the local views of a process p before and after the execution of union$[\ell]$. Let $InViews_i(v)$ be the set of views of processes that invoke union$[\ell]$ with side i, $i = 0, 1$, and let $OutViews(v)$ be the set of views returned by union$[\ell]$.

The proof of the lemma is by induction on the height, h, of v. In the base case, $h = 0$, v is a leaf. At most one process accesses union$[\ell]$ on each of its entrances, since processes have distinct names in the range $\{0, \ldots, 2^j - 1\}$. Therefore, the views in $InViews_i(v)$ are comparable for $i = 0, 1$. Conditions (1) and (2) follow from Lemma 6.1.

For the induction step, suppose that the lemma holds for every vertex at height h; we prove that it also holds for a vertex v at height $h + 1$. Let u be the left child of v and let w be the right child of v. By the algorithm, $InViews_0(v) = OutViews(u)$ and $InViews_1(v) = OutViews(w)$.

Since u is at height h, the views in $OutViews(u)$ are comparable by the induction hypothesis; similarly, the views in $OutViews(w)$ are comparable. Therefore, the views in $OutViews(v)$ are comparable by Lemma 6.1. This proves condition (1).

If process p_i invokes union$[\ell]$ with side 0 then $InView(v, p_i) = OutView(u, p_i)$. By the induction hypothesis, $p_i \in OutView(u, p_i) = InView(v, p_i)$. By Lemma 6.1, $InView(v, p_i) \subseteq OutView(v, p_i)$, and hence $p_i \in OutView(v, p_i)$, which proves condition (2). A similar argument holds if p_i invokes union with side 1. ∎

Acknowledgments. We thank Yehuda Afek and Eli Gafni for helpful discussions, Yossi Levanoni for comments on an earlier version of the paper, and the reviewers for many suggestions on how to improve the organization and presentation.

REFERENCES

[1] Y. AFEK, H. ATTIYA, D. DOLEV, E. GAFNI, M. MERRITT, AND N. SHAVIT, *Atomic snapshots of shared memory*, J. Assoc. Comput. Mach., 40 (1993), pp. 873–890.

[2] Y. AFEK, H. ATTIYA, A. FOUREN, G. STUPP, AND D. TOUITOU, *Long-lived renaming made adaptive*, in Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, ACM, Atlanta, 1999, pp. 91–103.

[3] Y. AFEK, D. DAUBER, AND D. TOUITOU, *Wait-free made fast*, in Proceedings of the 27th ACM Symposium on Theory of Computing, ACM, Las Vegas, 1995, pp. 538–547.

[4] Y. AFEK AND M. MERRITT, *Fast, wait-free* $(2k-1)$*-renaming*, in Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, ACM, Atlanta, 1999, pp. 105–112.

[5] Y. AFEK, G. STUPP, AND D. TOUITOU, *Long-Lived Adaptive Splitter and Applications*, manuscript, 1999.

[6] Y. AFEK, G. STUPP, AND D. TOUITOU, *Long-lived and adaptive collect with applications*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 262–272.

[7] Y. AFEK, G. STUPP, AND D. TOUITOU, *Long-lived and adaptive atomic snapshot and immediate snapshot*, in Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, Portland, OR, 2000, pp. 71–80.

[8] R. ALUR AND G. TAUBENFELD, *Results about fast mutual exclusion*, in Proceedings of the Real-Time Systems Symposium, IEEE, Phoenix, 1992, pp. 12–22.

[9] J. H. ANDERSON AND M. MOIR, *Using local-spin k-exclusion algorithms to improve wait-free object implementation*, Dist. Comp., 11 (1997), pp. 1–20.

[10] H. ATTIYA, A. BAR-NOY, D. DOLEV, D. PELEG, AND R. REISCHUK, *Renaming in an asynchronous environment*, J. Assoc. Comput. Mach., 37 (1990), pp. 524–548.

[11] H. Attiya and V. Bortnikov, *Adaptive and efficient mutual exclusion*, in Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, Portland, OR, 2000, pp. 91–100.

[12] H. Attiya and A. Fouren, *Adaptive wait-free algorithms for lattice agreement and renaming*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico, 1998, pp. 277–286.

[13] H. Attiya and A. Fouren, *An adaptive collect algorithm with applications*, submitted. Available at www.cs.technion.ac.il/∼hagit/pubs/AF99ful.ps.gz, Aug. 1999.

[14] H. Attiya and A. Fouren, *Polynomial and adaptive long-lived* $(2k-1)$*-renaming*, in Proceedings of the 14th International Conference on Distributed Computing, Toledo, Spain, Lecture Notes in Comput. Sci. 1914, Springer-Verlag, Berlin, 2000, pp. 149–163.

[15] H. Attiya, M. Herlihy, and O. Rachman, *Atomic snapshots using lattice agreement*, Dist. Comp., 8 (1995), pp. 121–132.

[16] H. Attiya and O. Rachman, *Atomic snapshots in* $O(n \log n)$ *operations*, SIAM J. Comput., 27 (1998), pp. 319–340.

[17] A. Bar-Noy and D. Dolev, *A partial equivalence between shared-memory and message-passing in an asynchronous fail-stop distributed environment*, Math. Sys. Theory, 26 (1993), pp. 21–39.

[18] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 41–52.

[19] J. E. Burns and G. L. Peterson, *The ambiguity of choosing*, in Proceedings of the 8th ACM Symposium on Principles of Distributed Computing, Edmonton, AB, Canada, 1989, pp. 145–158.

[20] M. Choy and A. K. Singh, *Adaptive solutions to the mutual exclusion problem*, Dist. Comp., 8 (1994), pp. 1–17.

[21] A. Fouren, *Exponential Examples for Two Renaming Algorithms*, http://www.cs.technion.ac.il/∼hagit/pubs/expo.ps.gz (August 1999).

[22] M. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. Assoc. Comput. Mach., 46 (1999), pp. 858–923.

[23] M. Inoue, W. Chen, T. Masuzawa, and N. Tokura, *Linear-time snapshot using multi-writer multi-reader registers*, in Proceedings of the 8th International Workshop on Distributed Algorithms, Terschelling, The Netherlands, 1994, Lecture Notes in Comput. Sci 857, Springer-Verlag, Berlin, 1994, pp. 130–140.

[24] L. Lamport, *A new solution of Dijkstra's concurrent programming problem*, Comm. ACM, 18 (1974), pp. 453–455.

[25] L. Lamport, *A fast mutual exclusion algorithm*, ACM Trans. Comput. Syst., 5 (1987), pp. 1–11.

[26] M. Moir, *Fast, long-lived renaming improved and simplified*, Sci. Comput. Programming, 30 (1998), pp. 287–308.

[27] M. Moir and J. H. Anderson, *Wait-free algorithms for fast, long-lived renaming*, Sci. Comput. Programming, 25 (1995), pp. 1–39.

[28] M. Moir and J. A. Garay, *Fast long-lived renaming improved and simplified*, in Proceedings of the 10th International Workshop on Distributed Algorithms, Bologna, Italy, Lecture Notes in Comput. Sci. 1511, Springer-Verlag, Berlin, 1996, pp. 287–303.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 665–682

ALGORITHMS FOR CAPACITATED VEHICLE ROUTING*

MOSES CHARIKAR†, SAMIR KHULLER‡, AND BALAJI RAGHAVACHARI§

Abstract. Given n identical objects (pegs), placed at arbitrary initial locations, we consider the problem of transporting them efficiently to n target locations (slots) with a vehicle that can carry at most k pegs at a time. This problem is referred to as k-delivery TSP, and it is a generalization of the traveling salesman problem. We give a 5-approximation algorithm for the problem of minimizing the total distance traveled by the vehicle.

There are two kinds of transportations possible—one that could drop pegs at intermediate locations and pick them up later in the route for delivery (preemptive) and one that transports pegs to their targets directly (nonpreemptive). In the former case, by exploiting the freedom to drop, one may be able to find a shorter delivery route. We construct a nonpreemptive tour that is within a factor 5 of the optimal preemptive tour. In addition we show that the ratio of the distances traveled by an optimal nonpreemptive tour versus a preemptive tour is bounded by 4.

Key words. vehicle routing, traveling salesman problem, approximation algorithms, graphs

AMS subject classifications. 90B06, 90C27, 68Q20, 68R10, 05C38, 05C85

PII. S0097539701392056

1. Introduction.

Vehicle routing and delivery problems have been widely studied in computer science and operations research. Many of these problems are NP-hard, and a lot of research has been done on analyzing heuristics to find "good" solutions to these problems. These transportation problems occur in real life in areas such as robotics and transportation of packages. Methods for obtaining "good" solutions to the problems are of great practical significance. For example, Casco, Golden, and Wasil [9] report that combining deliveries and pickups for supermarkets led to an industry wide savings of $160 million a year. The problem that we consider in this paper is that of transporting a single commodity from a set of suppliers to a set of demand points using a vehicle of limited capacity.

One way to analyze the performance of a heuristic is to compute the worst-case ratio between the cost of a solution produced by the algorithm to the cost of an optimal solution. If this ratio is bounded by ρ, we refer to this algorithm as an approximation algorithm with performance ratio ρ or simply as a ρ-approximation algorithm.

***k*-delivery TSP.** Given n identical pegs placed at arbitrary locations, a vehicle with a maximum capacity of k pegs, and n slots (demand points), each requiring a peg, the problem is to find a shortest tour for the vehicle in which all the pegs can be transported to their slots without exceeding the capacity of the vehicle. This

*Received by the editors December 1, 1999; accepted for publication (in revised form) January 30, 2001; published electronically October 23, 2001. A preliminary version of this paper appeared in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, Dallas, TX, 1998, pp. 349–358.

http://www.siam.org/journals/sicomp/31-3/39205.html

†Department of Computer Science, Stanford University, Stanford, CA 94305 (moses@cs.stanford.edu). This author was supported by ARO MURI grant DAAH04-96-1-0007 and NSF award CCR-9357849 with matching funds from IBM, the Schlumberger Foundation, the Shell Foundation, and Xerox Corporation.

‡Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (samir@cs.umd.edu). This author's research was supported by National Science Foundation grants CCR-9501355 and CCR-9820965.

§Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688 (rbk@utdallas.edu). This author's research was supported by National Science Foundation grants CCR-9409625 and CCR-9820902.

problem is referred to as k-delivery TSP. The traveling salesman problem (TSP) is a special case of k-delivery TSP, since replacing each vertex of the TSP by a peg and a slot yields an instance of the 1-delivery TSP. In this instance, the vehicle has to simply find a shortest tour that visits all the pegs (i.e., a TSP solution), since any peg that is picked up by the vehicle can immediately be delivered to the slot in the same location. Therefore, the k-delivery TSP is NP-hard [16], even when $k = 1$. The distances between the given points satisfy the triangle inequality since the vehicle can always take a shortest path between any two points. Replacing the distances between each pair of points by the shortest-path distance between them ensures that the triangle inequality is satisfied.

Haimovich and Rinnooy Kan [17] studied a *special case* of this problem when all the pegs are located at one central depot and are delivered with a vehicle of capacity k. They provided an approximation algorithm that obtains a performance ratio of 3. For geometric graphs—graphs induced by points in the plane with Euclidean distances as edge-weights—they provided a polynomial-time approximation scheme for constant k. Christofides [11] surveyed various issues, including problem formulation and algorithms, related to the vehicle routing problem, where the vehicles originate at a central depot. Asano, et al. [6] gave a polynomial-time approximation scheme for the same special case in the geometric setting, when k is $O(\log n/\log\log n)$. Anily and Hassin [2] demonstrated an algorithm that obtains a ratio of 2.5 for the 1-delivery TSP. The first constant factor approximation algorithm for the general problem was given by Chalasani, Motwani, and Rao [10]. We will refer to their algorithm as the "CMR algorithm." They obtained an approximation ratio of 9.5. They also gave better algorithms for the cases $k = 1$ and $k = \infty$ that yield ratios of 2 in both cases. Independent of our work, Anily and Bramel [1] showed that a modification of the CMR algorithm improves the approximation ratio to $7 - \frac{3}{k}$. (In fact, we show that one can obtain a better bound of 6.5 by modifying the CMR approach.) They also gave another algorithm with an approximation ratio of $c(k) + \frac{1}{2}\log_2 k$, where $2 \le c(k) \le 3$.

Our results. We summarize the results presented in this paper below:

- For the k-delivery TSP, we provide a natural approximation algorithm that runs in polynomial time and show that its performance ratio is at most 5. Since the proof is complex, we first prove a simpler bound of 6.5; this proof contains some of the basic ideas.
- For geometrical instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20] can be used to obtain an $(1+\epsilon)$ approximation of the TSP, and this leads to an approximation factor of $4(1+\epsilon)$ for these instances.
- We also describe a *simple* algorithm that finds a preemptive tour whose length is at most five times the length of an optimal preemptive tour. (We will shortly explain what we mean by a preemptive tour.)

Significance of our work. We explain below how our algorithm fundamentally differs from previous algorithms and why it is likely to return far better solutions in practice than they do. The previous methods for solving the general k-delivery TSP suffer from the following drawback. They start with two tours, one containing all the pegs (source nodes) and the other containing all the slots (delivery nodes). The basic idea is to traverse the cycle of pegs, collecting k pegs, then switch over to the other cycle and deliver the k pegs, repeating the process until all pegs are delivered. The delivery route thus alternates between the two cycles; it turns out that it is easy

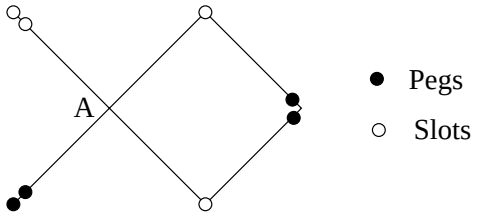


FIG. 1.1. *Example to show that intermediate drops help even for* $k = 2$.

to analyze the cost of shuttling between the cycles in this scheme. Such a scheme suffers from the drawback that the individual tours for the pegs and slots do not take advantage of the proximity of pegs and slots. In most instances, one can do better by alternating pickups and deliveries without waiting for the vehicle to become either completely full or completely empty. We derive a natural algorithm that *uses a single tour* containing all points, combining pickups and deliveries arbitrarily, and taking "corrective" action only when the vehicle becomes either full or empty. The main hurdle is in proving a good analysis of this more natural scheme. Our analysis shows significantly better approximation ratios for our algorithm than the previous algorithms. In addition, preliminary experimental studies show that our algorithm returns much better solutions.

Note. We will assume that we can start the vehicle's tour at any location. This is slightly different from previous work on this problem, which assumes that the starting point is fixed. With a fixed starting point, our method still applies with an additive factor of 1 in the approximation factors in the worst case.

Preemptive tours A fundamental issue is that of *preemptive* versus *nonpreemptive* traversals. In a preemptive traversal, pegs may be dropped at intermediate locations; in other words, we may pick up a peg and leave it at some location, and return later to collect it and deliver it. In a nonpreemptive solution, we carry a peg from its source to its destination without ever unloading it from the vehicle at intermediate nodes. The nature of the problem and algorithms can change if drops are permitted.

Figure 1.1 shows an example in which the best preemptive tour is shorter than the best nonpreemptive tour. The edges shown cost 1 unit each. A tour with a capacity-2 vehicle that leaves a peg at point A and delivers it later costs 8 units. If we are not allowed to drop a peg at intermediate locations, we incur a higher cost (regardless of where the tour starts), and an optimal nonpreemptive tour costs 10 units. This raises a very fundamental question—*what is the worst-case ratio of the cost of an optimal nonpreemptive tour to that of an optimal preemptive tour?* Our example shows that the ratio is at least $\frac{5}{4}$.

We show that the ratio between the optimal nonpreemptive and preemptive tours is at most 4 by showing that, given a preemptive tour of length L, we can find a nonpreemptive tour of length at most $4L$. This theorem is proven by using a variety of different ideas. One interesting method is a general technique for simulating a preemptive tour of a unit-capacity vehicle by a nonpreemptive tour *that travels the same distance* (Lemma 2.7).

Related work. A closely related problem is the *stacker-crane* problem. This problem also involves making deliveries with a vehicle of capacity k. In the stacker-crane problem, the objects are *not* identical and each object has a specific target

destination. The goal is to find a shortest tour that performs the transportation. For the unit-capacity case, Frederickson, Hecht, and Kim [15] gave an algorithm with an approximation factor of 1.8. For the case when the underlying metric is a tree, Frederickson and Guan [13, 14] have given fast algorithms to compute optimal solutions for the preemptive case and fast approximation algorithms for the nonpreemptive case. (The problems are NP-hard even for trees.) Knuth [19, section 5.4.8] discusses Karp's work [18] on the problem for paths and trees. Fast algorithms were given by Atallah and Kosaraju [7] for the cases in which the graph is either a simple cycle or path. The algorithms are slightly faster for the cases when drops are permitted. Frederickson [12] showed improved running times for a cycle when no drops are allowed. The issue of tours under various types of restrictions has also been investigated by Arkin, Hassin, and Klein [3].

Outline of the paper. In section 2 we describe an approximation algorithm for the k-delivery TSP that obtains an approximation ratio of 5. We first prove a simpler bound of 6.5 in section 2, and then, in section 3, we provide a better analysis of our algorithm. We also show how to "convert" a preemptive tour to a nonpreemptive tour. We prove that the total length of the nonpreemptive tour obtained by our algorithm is at most four times the length of the preemptive tour. In section 4, we describe a simple algorithm that finds a preemptive tour whose length is at most five times the length of an optimal preemptive tour.

Notation. An optimal nonpreemptive tour for the k-delivery TSP is denoted by C_k, and an optimal preemptive tour is denoted by C'_k. We will use C_k or C'_k to denote the length of the tour as well, and one can distinguish between the two meanings from the context.

2. An approximation algorithm for k-delivery TSP.

In this section, we provide an approximation algorithm and an analysis for its performance. We also show how to simulate a preemptive tour by a nonpreemptive tour. In particular, we show that $C_1 \leq kC'_k$ (Lemma 2.7). (This also shows the interesting result that $C_1 = C'_1, C_2 \leq 2C'_2, C_3 \leq 3C'_3$.)

We prove the following theorems.

THEOREM 2.1. *Consider an arbitrary instance of k-delivery TSP. There is a polynomial-time approximation algorithm that finds a nonpreemptive tour whose length is at most five times the length of an optimal tour (possibly preemptive). In the special case when the points are specified on the plane, and edge-costs are specified by Euclidean distances, the approximation ratio can be improved to $4(1+\epsilon)$ for any constant $\epsilon > 0$.*

THEOREM 2.2. *The length of an optimal nonpreemptive tour of a k-delivery TSP instance is at most four times the length of an optimal preemptive tour, i.e., $C_k \leq 4C'_k$.*

2.1. Overview of the algorithm. The main idea is the following: First construct a tour of *all* the given points. Starting from some initial vertex, we traverse the tour, picking up pegs, and delivering them to slots, on-line. In other words, when the vehicle passes through a node with a peg, it picks up the peg, and when it passes through a slot, it drops a peg there. We show that if the vehicle has unbounded capacity, then there is always a starting point such that the vehicle can complete all deliveries without ever running out of pegs. If the vehicle has bounded capacity k, the simple scheme outlined above does not work directly. We need to address the following two situations: (a) the vehicle is full when a peg is visited, and (b) the vehicle is empty when a slot is visited.

In the following discussion, we assume that k is even. We will later describe how to handle the case when k is odd. The performance ratio is at most $5\frac{1}{6}$ for odd k. We treat the vehicle as full when it has $k/2$ pegs on it, and the remaining capacity is used as a "buffer." The tour is broken up into segments of 3 kinds: (i) *neutral* segments with equal number of pegs and slots, (ii) *positive* segments that have $k/2$ more pegs than slots, and (iii) *negative* segments that have $k/2$ more slots than pegs. Neutral segments are processed as mentioned in the above scheme. In most practical situations, most parts of the tour constructed may be neutral segments, and in this case our algorithm would do very well since neutral segments are processed by traversing them only once. By definition, there are as many positive segments as negative segments. We compute a minimum-weight perfect matching between the positive and the negative segments. When the vehicle is passing through a positive segment on its tour, it delivers the excess pegs to the negative segment to which the positive segment is matched.

The main difficulty in analyzing such a scheme is that the cost of a matching between the positive and the negative segments has to be bounded with respect to an optimal tour. Note that the matching does not include all nodes in the original problem, and therefore it could possibly be arbitrarily expensive. Another complication is that the segments do not have the same number of points on them. The techniques used in the previous results [1, 10] do not yield a bound on the cost of such a matching. We show how to bound the cost of the matching with respect to an optimal tour and use it to derive a better approximation bound. We now describe the algorithm in detail.

2.2. The algorithm.

1. Construct a tour T that visits all the points.
2. Fix a reference point P on the tour T.
3. Traverse the tour T in some direction starting from P.
4. Compute the *excess* function $\text{EXCESS}(e)$ for each edge e of T. $\text{EXCESS}(e) = \text{PEGS}(e) - \text{SLOTS}(e)$, where $\text{PEGS}(e)$ is the number of pegs encountered before e is traversed, and $\text{SLOTS}(e)$ is defined analogously as the number of slots encountered before e.
5. For each value of $i \in [0, k/2)$ do
   (a) Break the tour into pieces by removing all edges with $\text{EXCESS}(e) \equiv i \bmod k/2$. Call these edges *cut* edges. Figure 2.1 shows a sample tour and a plot of the excess function for a counterclockwise traversal of the tour (with $k = 6$ and $i = 0$).
   (b) We get p-pieces (positive pieces), n-pieces (negative pieces), and 0-pieces (zero or neutral pieces) as follows.
   - A p-piece is one where the *excess* function is x on the cut edge preceding the piece and $x + k/2$ on the cut edge following the piece; for all edges e in the piece, $\text{EXCESS}(e) \in (x, x + k/2)$.
   - An n-piece is one where the *excess* function is x on the cut edge preceding the piece and $x - k/2$ on the cut edge following the piece; for all edges e in the piece, $\text{EXCESS}(e) \in (x - k/2, x)$.
   - A 0-piece is one where the *excess* function is x on the cut edge preceding the piece and x on the cut edge following the piece; for all edges e in the piece, either $\text{EXCESS}(e) \in (x - k/2, x)$ (such a piece is called *decreasing*) or $\text{EXCESS}(e) \in (x, x + k/2)$ (such a piece is called *increasing*).

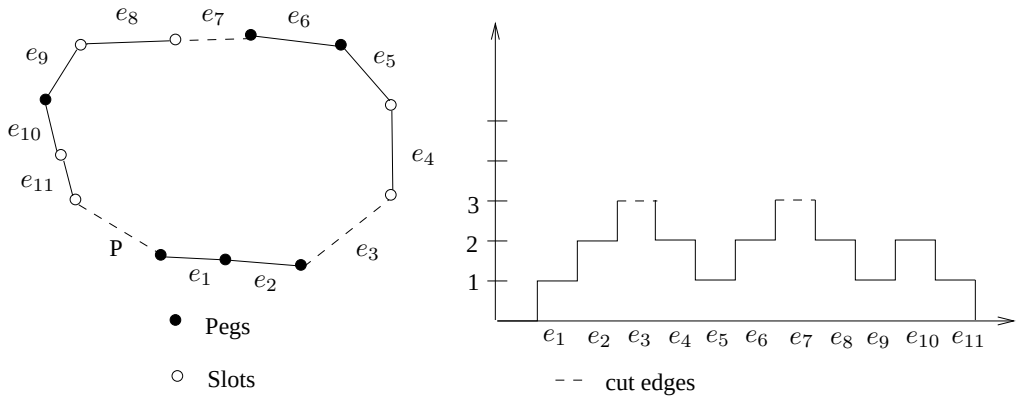


FIG. 2.1. *A tour and the excess function plotted for $k = 6$ and $i = 0$.*

(c) Compute a matching on the p- and n-pieces as follows:
- Construct a weighted bipartite graph $B = (V^+, V^-, E)$ on p- and n-pieces as follows: There is one vertex in V^+ for each p-piece and one vertex in V^- for each n-piece. For each p-piece u^+ and each n-piece w^-, the edge (u^+, w^-) has weight equal to the minimum-weight edge connecting a vertex in u^+ and a vertex in w^-.
- Compute a minimum-weight perfect matching M in the bipartite graph B. A p-piece and an n-piece that are matched to each other are said to form a p/n-pair.

(d) Now traverse the tour starting from any point in both clockwise and counterclockwise directions as follows (assume for now that we start each traversal at the beginning of a piece with exactly $k/2$ pegs; Lemma 2.3 shows how this assumption is unnecessary):
- On encountering a 0-piece, we move along the piece picking up pegs and delivering them. Since we start with $k/2$ pegs, we can do this in a single traversal and reach the end of the piece with $k/2$ pegs. Note that we leave the piece with $k/2$ pegs.
- When we encounter the first piece of a p/n-pair, we service the pair as follows: Suppose we encounter a p-piece P^+ which is matched with n-piece P^- by edge $e \in M$. Note that there must be an edge e' corresponding to e, such that e' connects some vertex in P^+ to some vertex in P^-, and the weight of e' is the same as that of e. Traverse P^+, performing pickups and deliveries, until e' is encountered. Now, move to the beginning of P^- and traverse P^- performing pickups and deliveries. Then move back to the point in P^+ where we left off and continue performing pickups and deliveries. Note that after servicing a p/n-pair in this way, we finish with $k/2$ pegs. The case when the n-piece is encountered before the p-piece is handled similarly.
- When we encounter the second piece of a p/n-pair, we simply traverse the piece without performing any pickups and deliveries (since it has already been serviced).

(e) For both of the tours (clockwise and counterclockwise) described above,

find a valid starting point (i.e., a starting point such that we never run out of pegs and the number of pegs carried is at most k). Lemma 2.3 guarantees the existence of such a starting point.

6. Return a shortest tour from amongst the k tours constructed (two for each $i \in [0, k/2)$).

2.3. Analysis: A weaker bound. We can prove an upper bound of 5 on the approximation ratio achieved by the above algorithm. Since the proof is complex, we prove a simpler bound of 6.5 that contains the basic ideas. In fact, for the simpler proof we can fix $i = 0$ in the above algorithm. To prove the better bound, we need to try all values of i. Section 3 contains the proof for the bound of 5. We also prove the relationship between the tour generated by the algorithm and C'_k (Theorem 2.2).

LEMMA 2.3. *A valid starting point is guaranteed to exist in step* 5(e) *of the algorithm.*

Proof. Suppose we started the tour constructed by the algorithm at the beginning of any piece with a vehicle preloaded with $k/2$ pegs. Then the following invariant is maintained throughout the tour: The vehicle has exactly $k/2$ pegs when it traverses a cut edge from one piece to another. We can verify that the number of pegs carried by the vehicle always lies in the interval $[0, k]$. This would be a valid tour except for the fact that we assumed that we initially started with $k/2$ pegs. Let $n(e)$ be the number of pegs carried by the vehicle as it traverses edge e (beginning with $k/2$ pegs). Consider the edge $e_{\min}$, where $n(e)$ reaches its minimum value, say, x (break ties arbitrarily). Suppose we start a new vehicle with no pegs from edge $e_{\min}$. Let $n'(e)$ be the number of pegs carried by the new vehicle as it traverses edge e. It can be verified that $n'(e) = n(e) - x$. By the choice of $e_{\min}$, this ensures that the number of pegs carried by the vehicle always lies in the interval $[0, k]$. This proves the existence of a valid starting point for the tour and, in fact, also gives a simple method to find such a starting point. □

Based on the above lemma, we can also show the following lemma.

LEMMA 2.4. *There is a polynomial-time approximation algorithm for the* ∞*-delivery TSP (i.e., the vehicle has infinite capacity) with a performance ratio of* 1.5. *For geometric instances on the plane, the approximation ratio is* $1 + \epsilon$ *for any* $\epsilon > 0$.

Proof. Let T_{OPT} be a minimum length tour of all the points. Since C'_k is a tour of the points, $T_{OPT} \leq C'_k$. The algorithm constructs a tour T of all the points. For the algorithm, we will assume that an α-approximation of the TSP tour is used, and therefore the weight of the tour is at most $\alpha T_{OPT} \leq \alpha C'_k$. If Christofides's algorithm is used, then $\alpha \leq 1.5$. For geometric instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20] can be used to obtain a $(1 + \epsilon)$ approximation of the TSP tour. In this case, $\alpha = 1 + \epsilon$ for any constant $\epsilon > 0$. Using the same ideas as in the above lemma, we can show that there is always a valid starting point on this tour such that we never run out of pegs. □

In step 5(c) we find a matching M on the p- and n-pieces. We need to bound the weight of the matching M. We cannot bound the weight of the matching by the method used in [10] since the matching is not being computed in a graph that includes all the pegs and slots. We use a different approach to bound the weight of the matching.

Let A be a minimum-weight perfect matching between pegs and slots, where a peg must be matched to a slot. We use A to denote the matching and its weight. One can distinguish between the two meanings from the context.

2.3.1. Bounding the matching *A*. The following lemmas derive an upper bound on the ratio of the weight of matching A to the weight of an optimal preemptive solution. Even though there are other ways of proving this directly, these proof methods also show how to "simulate" a preemptive solution with a nonpreemptive solution.

Our main goal is to prove the following theorem.

THEOREM 2.5. $A \leq \frac{k}{2} C'_k$.

This theorem follows once we establish the following lemmas.

LEMMA 2.6 (see [10]). $A \leq \frac{1}{2} C_1$.

We now prove that a nonpreemptive unit-capacity vehicle can simulate a preemptive capacity-k vehicle with an increase of the length of the tour by a factor of k. One could attempt to prove this by walking around the cycle k times, but the problem is that we may attempt to pick up a peg that is not yet "available."

LEMMA 2.7. $C_1 \leq k \cdot C'_k$.

Proof. Consider a tour C'_k which delivers pegs to slots and is allowed intermediate dropping points. We will show that we can convert this tour into one of length $k \cdot C'_k$, where the pegs are carried to slots with no intermediate drops by a unit-capacity vehicle. Assume the vehicle of capacity k starts at s and returns to s.

We construct an auxiliary multigraph $G = (V, E)$ from the tour as follows. The vertex set of the graph is defined to be

$$V = \{s\} \cup \{x | x \text{ is an intermediate drop point}\}.$$

Imagine that we store the pegs on numbered compartments in the vehicle. The numbers are $1 \ldots k$. The tour starts from s and visits pegs/slots and intermediate drop points. We can view the tour as "segments," where it goes back and forth between the vertices of the graph we constructed (see Figure 2.2). We create k edges in G for each movement done by the vehicle between two vertices of G. Each edge corresponds to a numbered compartment. We associate with each such edge the pegs which were placed into this compartment and the slots which were serviced by removing a peg from this compartment. Thus each edge is associated with an alternating sequence of pegs and slots. As the vehicle moves from vertex x to vertex y of G, each numbered compartment undergoes the following various changes:

(1) Lose a peg. (Leave x with a peg; arrive at y with no peg.) The corresponding edge is associated with a peg/slot sequence of the form $sps \ldots ps$. (Here p represents a peg and s a slot.)
(2) Gain a peg. (Leave x with no peg; arrive at y with a peg.) The corresponding edge is associated with a sequence of the form $psp \ldots sp$.
(3) Move without carrying anything from x to y. (We may load and unload pegs in this compartment during the motion.) The corresponding edge is associated with a sequence of the form $psp \ldots ps$ (possibly empty).
(4) Move carrying a peg from a vertex to another vertex. (We may unload and load pegs in this compartment during the motion.) The corresponding edge is associated with a sequence of the form $sps \ldots sp$ (possibly empty).

Each edge corresponding to a compartment is a labeled edge in the multigraph. Type (1) edges are labeled as $-$ edges. Type (2) edges are labeled as $+$ edges. We subdivide edges of type (3) by a vertex xy. The edge from x to xy gets the label $+$, and the edge from xy to y is labeled as a $-$ edge. We subdivide edges of type (4) by a vertex xy. The edge from x to xy gets the label $-$, and the edge from xy to y is labeled as a $+$ edge. We can think of a $+$ edge as being associated with an odd length alternating

peg/slot sequence that begins and ends with a peg. Similarly, a $-$ edge can be thought of as being associated with an odd length alternating peg/slot sequence that begins and ends with a slot.

LEMMA 2.8. *For each vertex of the auxiliary multigraph G, the number of $+$ labels incident to it is equal to the number of $-$ labels incident to it.*

Proof. For each compartment, the contribution to the $+$ and $-$ label is the same. Consider a vertex v that is a dropping point. The number of pegs that are carried to it are carried away. (Formally, each vertex v occurs a number of times on the tour. There are 4 cases: (a) come with a peg, leave it here, and go $(+v+)$; (b) come with a peg, leave with a peg $(+v-)$; (c) come with no peg, go with no peg $(-v+)$; (d) come with no peg, go with a peg $(-v-)$. The number of cases (a) and (d) are the same since (a) increases peg count and (d) decreases peg count.) □

In order to construct a solution that does not drop pegs at intermediate locations, we need to construct an alternating sequence of pegs and slots which visits every peg and every slot. Find an Euler tour that alternates using $+$ and $-$ edges in this multigraph. (When we enter a vertex on a $+$ edge we can leave on a $-$ edge and vice versa.) The Euler tour can be interpreted as a nondrop tour solution for a vehicle with unit capacity. The unit-capacity vehicle simply traverses the edges in the order of the Euler tour. When we traverse a particular edge, we service the sequence of pegs and slots associated with that edge. The definition of $+$ and $-$ edges ensures that we encounter pegs and slots in an alternating fashion. This therefore gives a nondrop tour solution with unit capacity. This completes the proof of Lemma 2.7. □

We illustrate this construction by an example in Figure 2.2. Suppose we are given the preemptive tour that starts at s, pick up a peg on path A, and drop it off at the first drop off point. We then take the loop marked B performing one delivery, come back to pick up the dropped off peg, and take the loop marked C. We return with a peg, drop it off at the same place, and then take the path D, drop off a second peg, come back to pick up the first peg on path E, deliver it on path F, pick up the second peg, and take path G, returning to s.

Corresponding to this traversal, we construct the auxiliary multigraph G. We have three vertices to begin with. Path A gains a peg and is marked $+$. Path B is a loop on which we leave without a peg and return without a peg; we subdivide this edge and it is a $+-$ edge. Path C is a loop on which we leave with a peg and return with a peg, and so we subdivide it and mark it a $-+$ edge. In a similar manner we finish the construction of the multigraph. An Euler tour in this multigraph that alternates between $+$ and – edges is easy to find. This corresponds to a nonpreemptive traversal.

2.3.2. Bounding the matching M. Let W be a matching (which we call a *wiggly matching*) that matches pegs to slots such that all points are matched except for some $k/2$ pegs in each p-piece and $k/2$ slots in each n-piece. We use the *wiggly matching* to bound the cost of the matching M as follows.

LEMMA 2.9.

$$M \leq \frac{2}{k}(A + W). \tag{2.1}$$

Proof. Consider the symmetric difference of A and W. A is a perfect matching and W is a matching where all the pegs and slots in the 0-pieces are matched, and in each p- (n-) piece, there are $\frac{k}{2}$ unmatched pegs (slots). The symmetric difference has even length cycles and paths. Since A is a perfect matching, the symmetric difference

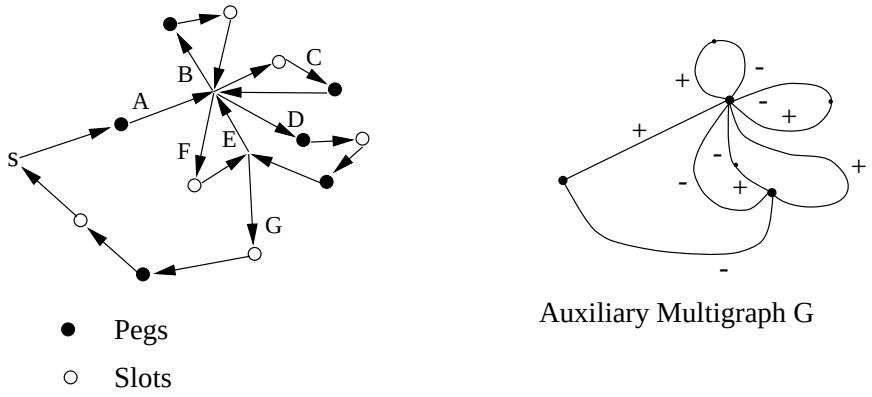


FIG. 2.2. *Illustrating construction of auxiliary multigraph $G = (V, E)$ for a unit-capacity vehicle.*

is a collection of disjoint augmenting paths (with respect to W) that start and end with edges from A at vertices of degree 1 in the graph $A \cup W$. The only vertices having degree 1 are pegs in the p-pieces and slots in the n-pieces. Each augmenting path with respect to W has a peg at one end and a slot at the other end. This gives us paths from pegs to slots such that each p-piece has $k/2$ paths emerging from it and each n-piece has $k/2$ paths ending on it. This collection of paths can be decomposed into $k/2$ perfect matchings between p- and n-pieces (because the edges of a d-regular bipartite graph can be decomposed into d perfect matchings [8]). It follows that the weight of the minimum-weight perfect matching on the p- and n-pieces is at most the average weight of the $\frac{k}{2}$ matchings we found. By triangle inequality, the weight of the symmetric difference is at most the sum of the weights of the matchings. This proves the lemma. □

Recall that T is the tour of all the points constructed by the algorithm. Let $T^{(\pm)}$ be the total length of the p/n-pieces and let $T^{(0)}$ be the total length of the 0-pieces. Let $T^{(C)}$ be the total length of the cut edges. Then $T = T^{(\pm)} + T^{(0)} + T^{(C)}$.

2.3.3. Bounding the matching W.

LEMMA 2.10. *There exists a wiggly matching (a matching that matches pegs to slots such that all points are matched except for some $k/2$ pegs in each p-piece and $k/2$ slots in each n-piece) W such that*

$$W \leq \frac{k}{2}\left(\frac{1}{2}T^{(\pm)} + T^{(0)}\right). \tag{2.2}$$

Proof. We will construct the *wiggly matching* separately for 0- and p/n-pieces. A 0-piece can be traversed in one of the two directions by a vehicle with no pegs initially such that pegs are picked up and delivered without having to back up. Suppose this vehicle operates like a stack, pushing and popping pegs. For each $j \in [0, k/2)$, consider the pegs which were placed in location j of this stack and the slots which were serviced by popping a peg from location j of this stack. This defines a path of the form $psp \ldots s$, where p stands for a peg and s stands for a slot. We match each peg on this path with the slot immediately following it. The total length of the matching edges is at most the length of the path. However, the length of the path is at most the length of the 0-piece. Since we construct $k/2$ paths, the total length of the matching within this 0-piece is at most $k/2$ times the length of the 0-piece.

A p-piece can be traversed (in either direction) by a vehicle with no pegs initially such that pegs are picked up and delivered without having to back up. The vehicle ends up with $k/2$ pegs at the end. Analogous to the construction for a 0-piece, we define a path corresponding to each location j of the stack. Each path is of the form $psp \dots sp$. Now we can match pegs with slots either starting from the left or the right. In either case, one peg is left unmatched. We choose the lower cost matching between the two. The total cost of the two matchings is exactly the cost of the path. Thus the smaller of the two matchings costs at most half the length of the path. Again the length of each path is at most the length of the p-piece. Since we construct $k/2$ paths, the total length of the matching within this p-piece is at most $k/4$ times the length of the p-piece. A similar construction can be done for n-pieces. Hence the lemma follows. □

Theorem 2.5 bounds the weight of the matching A to be at most $\frac{k}{2}C'_k$. Substituting into (2.1) for A from the theorem and for W from (2.2), we get

$$M \leq C'_k + \frac{1}{2}T^{(\pm)} + T^{(0)}. \tag{2.3}$$

Proof of Theorems 2.1 *and* 2.2. Note that in the two tours constructed by the algorithm, each 0-piece is traversed once and each cut edge is traversed once. Also the first of each p/n-pair is traversed once while the second is traversed thrice. Averaging over the two traversals, we charge each p/n-piece twice. Each edge in M is traversed twice.

Hence the average length of the two tours constructed is at most

$$\begin{aligned} 2M + T^{(0)} + 2T^{(\pm)} + T^{(C)} &\leq 2C'_k + 3T^{(0)} + 3T^{(\pm)} + T^{(C)} \\ &\leq 2C'_k + 3T \\ &\leq 2C'_k + 3\alpha C'_k = (2 + 3\alpha)C'_k. \end{aligned}$$

Since $\alpha \leq 1.5$, we obtain a worst-case approximation ratio of 6.5. We can also view this as a constructive proof bounding the ratio of C_k to C'_k. In that case $\alpha = 1$ and we obtain an upper bound of 5 on the ratio. In section 3 we show a bound of $(2+(2-\frac{4}{k^2})\alpha)C'_k$. Using this improved bound, we get an approximation ratio of 5, and we can bound the ratio of C_k to C'_k by 4. Also, for geometric instances (consisting of points in the plane with Euclidean distances), using an $(1+\epsilon)$-optimal tour [4, 5, 20] in the first step of the algorithm, we obtain a ratio of $4(1+\epsilon)$. □

If we are given a fixed starting point q, we obtain a tour that starts and ends at q as follows. We apply the algorithm as usual. Suppose this constructs a tour that starts and ends at q', which is a peg without loss of generality. We first move from q to q', traverse the tour constructed by the algorithm, and, on reaching q' again, move back to q. We pay an additional cost of twice the distance between q and q', which is at most the cost of the optimal tour, since the optimal tour includes two paths between q to q'. This adds 1 to our approximation ratios.

3. A better analysis. Notice that the previous analysis did not use the fact that we try different break points and average over them. We now present a better analysis that builds on the basic ideas from section 2.3 to show that the average tour length for the k tours constructed is actually at most $(2 + (2 - \frac{4}{k^2})\alpha)C'_k$.

Examining the proof of Lemma 2.10, we observe that the proof constructs $k/2$ paths within each piece of the tour. The contribution to the total length of the final tour is the average path length in the p/n-pieces plus twice the average path length in

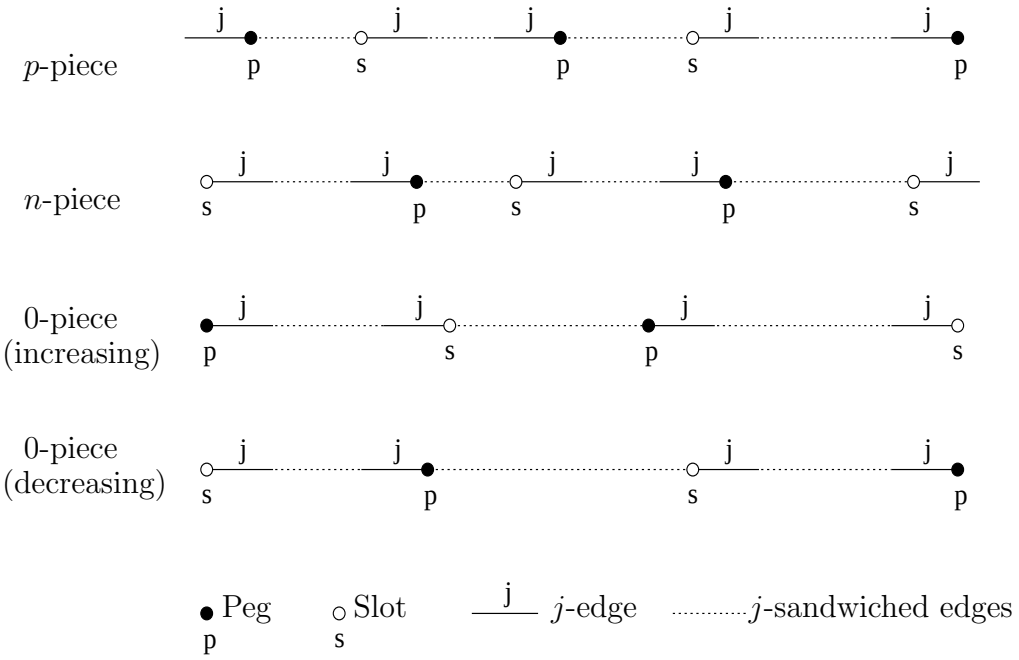


FIG. 3.1. *Illustrating the definition of paths in terms of the excess function.*

the 0-pieces. Here, "average" refers to the average length of the $k/2$ paths. The earlier proof bounded each path length by the length of the entire piece. We will improve the analysis by getting better estimates for the average path lengths by averaging over all values of i used to define the cut edges of the tour.

For a node v (peg or slot) in the tour T, define BEFORE(v) to be the edge preceding v in T and AFTER(v) to be the edge following v in T.

It will be useful to identify the paths constructed by the proof of Lemma 2.10 with the values of the *excess* function on edges within a piece. In the following discussion, we fix a value of i used to define the cut edges. For a given i, for every integer $j \in [0, k/2)$, we construct a path in each piece based on the values of the *excess* function as follows. An edge e is said to be a j-edge if EXCESS(e) $\equiv j \pmod{k/2}$.

For a p/n-piece, the path consists of the pegs p such that BEFORE(p) is a j-edge and the slots s such that AFTER(s) is a j-edge. For a 0-piece, we define the paths differently depending on whether the piece is *increasing* or *decreasing*. For an *increasing* 0-piece, the path consists of the pegs p such that AFTER(p) is a j-edge and the slots s such that BEFORE(s) is a j-edge. For a *decreasing* 0-piece, the path consists of the slots s such that AFTER(s) is a j-edge and the pegs p such that BEFORE(p) is a j-edge. See Figure 3.1 for an illustration of this definition of paths. Observe that by this definition, for all cases, the path for $j = i$ always has zero length. For a p-piece, it consists of a single peg. For an n-piece, it consists of a single slot. For a 0-piece, the path for $j = i$ is empty. The reader can verify that the paths constructed by this definition are identical to the paths constructed in the proof of Lemma 2.10.

Let $l_{ij}^{(\pm)}$ be the total length of the paths corresponding to j (by the above correspondence) within the p/n-pieces when i is used to define the cut edges. We define

$l_{ij}^{(0)}$ similarly for the paths within the 0-pieces.

Call an edge e in a piece *j-sandwiched* if e is not a j-edge and there is some j-edge before e and some j-edge after e in the piece. (The j-edges need not be immediately before and after e.) Note that the property of being a j-edge is independent of the value of i; however, the property of being a j-sandwiched edge depends on the value of i used to define the cut edges. In any piece, the path corresponding to j (described above) consists of j-sandwiched edges and j-edges Thus the length of this path is bounded by the sum of the total length of the j-sandwiched edges and the total length of the j-edges in the piece.

Let $p_{ij}^{(\pm)}$ be the total length of the j-sandwiched edges within the p/n-pieces when i is used to define the cut edges. We define $p_{ij}^{(0)}$ similarly for the 0-pieces. Let $q_{ij}^{(\pm)}$ be the total length of the j-edges within the p/n-pieces when i is used to define the cut edges. We define $q_{ij}^{(0)}$ similarly for the 0-pieces. By the above definitions, we have

$$l_{ij}^{(\pm)} \le p_{ij}^{(\pm)} + q_{ij}^{(\pm)},$$
$$l_{ij}^{(0)} \le p_{ij}^{(0)} + q_{ij}^{(0)}.$$

Also, when i is used to define the cut edges, let $T_i^{(\pm)}$ be the total length of the p/n-pieces, let $T_i^{(0)}$ be the total length of the 0-pieces, and let $T_i^{(C)}$ be the total length of the cut edges.

Note that

$$\sum_{j\neq i} q_{ij}^{(\pm)} = T_i^{(\pm)},$$
$$\sum_{j\neq i} q_{ij}^{(0)} = T_i^{(0)}.$$

Now, the analysis of the previous section bounded the average length of the two tours produced when a particular value of i is used to define the cut edges. In terms of the notation introduced, this bound can be written as

$$\begin{aligned}
&\frac{4}{k}A + \frac{2}{k}\sum_{j\neq i} l_{ij}^{(\pm)} + \frac{4}{k}\sum_{j\neq i} l_{ij}^{(0)} + 2T_i^{(\pm)} + T_i^{(0)} + T_i^{(C)} \\
&\le 2C'_k + \frac{2}{k}\sum_{j\neq i}\left(p_{ij}^{(\pm)} + q_{ij}^{(\pm)}\right) + \frac{4}{k}\sum_{j\neq i}\left(p_{ij}^{(0)} + q_{ij}^{(0)}\right) + 2T_i^{(\pm)} + T_i^{(0)} + T_i^{(C)} \\
&= 2C'_k + \frac{2}{k}\sum_{j\neq i} p_{ij}^{(\pm)} + \frac{4}{k}\sum_{j\neq i} p_{ij}^{(0)} + \left(2+\frac{2}{k}\right)T_i^{(\pm)} + \left(1+\frac{4}{k}\right)T_i^{(0)} + T_i^{(C)}.
\end{aligned}$$

Let L be the length of the constructed tour averaged over all the $k/2$ values of i. Summing up the above bound over all values of i, we get

$$\begin{aligned}
\frac{k}{2}L \le kC'_k + \underbrace{\frac{2}{k}\sum_i\sum_{j\neq i} p_{ij}^{(\pm)}}_{S_1} + \underbrace{\frac{4}{k}\sum_i\sum_{j\neq i} p_{ij}^{(0)}}_{S_2} + \left(2+\frac{2}{k}\right)\underbrace{\sum_i T_i^{(\pm)}}_{S_3} \\
+ \left(1+\frac{4}{k}\right)\underbrace{\sum_i T_i^{(0)}}_{S_4} + \underbrace{\sum_i T_i^{(C)}}_{S_5}.
\end{aligned} \tag{3.1}$$

We will compute a bound for sum of the last five terms of the right-hand side of (3.1) in terms of T.

Consider a particular edge e of the tour. Let us calculate its contribution to the various terms in (3.1). Suppose e is an r-edge. For each value of $i \in [0, k/2), i \neq r$, we scan to the left and right of e for the closest i-edges before and after e. Let A_i be the portion of the path between (and not including) these two i-edges. When i is chosen to define the cut edges of the tour, A_i is a (p/n- or 0-) piece. Accordingly, we label the segment A_i as p/n or 0. Note that we use the term segment as opposed to piece, as a segment A_i may or may not be a piece depending on the value used to define the cut edges.

Edge e contributes to $l_{ij}^{(\pm)}$ iff A_i is a p/n segment and e is j-sandwiched within A_i, i.e., iff A_j is completely contained within A_i. The edge e contributes to $l_{ij}^{(0)}$ iff A_i is a 0-segment and e is j-sandwiched within A_i, i.e., iff A_j is completely contained within A_i. The edge e contributes to $T_i^{(\pm)}$ iff A_i is a p/n segment and contributes to $T_i^{(0)}$ iff A_i is a 0-segment. It contributes to $T_i^{(C)}$ iff $i = r$.

Claim. If A_i is a p/n segment, it cannot be contained in any other segment A_j ($j \neq i$).

Proof. Since A_i is a p/n segment, the *excess* function changes by $k/2$ from one end to the other. Hence there must be some point within it where the *excess* function is $j \bmod k/2$. Hence one of the end points of A_j must be within A_i, proving the claim. □

Suppose x of the segments A_i are 0-segments and $(\frac{k}{2} - x - 1)$ of them are p/n segments. Let us calculate the number of times edge e is counted in the various terms in (3.1).

The contribution to S_1 is the number of pairs (A_j, A_i) such that A_j is contained in A_i and A_i is a p/n-segment By the above claim, A_j must be a 0-segment. Thus the number of such pairs is at most $x(\frac{k}{2} - x - 1)$. The contribution to S_2 is the number of pairs (A_j, A_i) such that A_j is contained in A_i and A_i is a 0-segment. By the above claim, A_j must be a 0-segment. Note that if A_j is contained in A_i, then A_i is not contained in A_j. Thus the number of such pairs is at most $\frac{x(x-1)}{2}$. The contribution to S_3 is $\frac{k}{2} - x - 1$. The contribution to S_4 is x. The contribution to S_5 is 1.

Thus the contribution of edge e to the sum of the last five terms in (3.1) is at most

$$\begin{aligned}&\frac{2}{k}x\left(\frac{k}{2} - x - 1\right) + \frac{4}{k}\frac{x(x-1)}{2} + \left(2 + \frac{2}{k}\right)\left(\frac{k}{2} - x - 1\right) + \left(1 + \frac{4}{k}\right)x + 1\\ &= k - \frac{2}{k} - \frac{2x}{k} \leq k - \frac{2}{k}.\end{aligned}$$

Hence the sum of the last five terms in (3.1) is at most $(k - \frac{2}{k})T$. Using this bound in (3.1), we get

$$\frac{k}{2}L \leq kC'_k + \left(k - \frac{2}{k}\right)T,$$

and hence

$$\begin{aligned}L &\leq 2C'_k + \left(2 - \frac{4}{k^2}\right)T\\ &\leq 2C'_k + \left(2 - \frac{4}{k^2}\right)\alpha C'_k.\end{aligned}$$

Since $\alpha = 1.5$, we obtain a $5 - \frac{6}{k^2}$-approximation algorithm. As before, we can also view this as a constructive proof bounding the ratio of C_k to C'_k. In that case $\alpha = 1$ and we obtain an upper bound of $4 - \frac{4}{k^2}$ on the ratio.

3.1. Odd values of k. When $k > 1$ is odd, we can use our algorithm using $k - 1$ as the capacity of the vehicle. Theorem 2.5 and Lemmas 2.6 and 2.7 are true for all values of k, independent of its parity. In Lemmas 2.9 and 2.10, k should be replaced by $k - 1$. If the proof is repeated with these modifications, the performance ratio increases by an additive term of $\frac{2}{k-1}$. The performance ratio in this case is at most $5 - \frac{6}{(k-1)^2} + \frac{2}{k-1} \leq 5.1667$.

4. A preemptive algorithm. In this section we describe a simple strategy that also achieves an approximation factor of 5 for the preemptive k-delivery TSP. Our algorithm outputs a preemptive solution (i.e., may drop pegs at intermediate locations during the course of the algorithm), and the length traveled by the vehicle is compared to the length of an optimal preemptive solution.

This algorithm is a modification of the strategy given by Chalasani, Motwani, and Rao [10].

4.1. The CMR Algorithm. We first review the algorithm given by Chalasani, Motwani, and Rao [10] for this problem. We then show that a modification of the algorithm improves the approximation ratio to 6.5 without preemption and 5 with preemption. We will assume for simplicity that n is a multiple of k and that k is even. The former assumption can be made to hold by adding at most $k - 1$ dummy peg/slot pairs.

1. Find tours (of almost minimum weight) T_p and T_s of the pegs and slots points, respectively. (This step could be implemented using Christofides's heuristic for the TSP.)
2. Break T_p and T_s into paths containing k vertices each, by deleting every kth edge from each cycle.
3. View each k-path as a "supernode" and construct an auxiliary complete bipartite graph which has one vertex for each of the supernodes in T_p and T_s. The weight of an edge in this bipartite graph is the shortest distance between a pair of points belonging to the respective supernodes.
4. Find a minimum-weight perfect matching M in this bipartite graph.
5. Traverse the tour T_p and at the end of each segment use the matching M to transport the k pegs to the corresponding delivery point (supernode) in T_s.

The total length of such a tour is shown to be at most $3T_p + 2T_s + 2M \leq 4.5C_k + 3C_k + \frac{2}{k}A$. As shown in [10], $A \leq kC_k$. Using Christofides's approximation for the TSP, T_p and T_s are at most $1.5C_k$. The approximation ratio obtained is therefore at most 9.5. For geometrical instances, such as points in the plane, the algorithms of Arora [4, 5] and Mitchell [20] can be used to obtain an $(1 + \epsilon)$ approximation of the TSP, and this leads to better approximation factors for these instances.

Theorem 2.5 shows that $A \leq \frac{k}{2}C'_k$. Since $C'_k \leq C_k$, we also have $A \leq \frac{k}{2}C_k$. Using this improved upper bound on the weight of A, the approximation ratio of the algorithm improves from 9.5 to 8.5. A small change to the algorithm improves it further to an approximation factor of 7 as follows. We can traverse T_p in two ways. A clockwise traversal and a counterclockwise traversal give us two tours whose total length is at most $4T_p + 4T_s + 4M \leq 12C_k + 4(\frac{C_k}{2})$. The smaller of these two tours has length at most the average of these two tour lengths, which is $7C_k$.

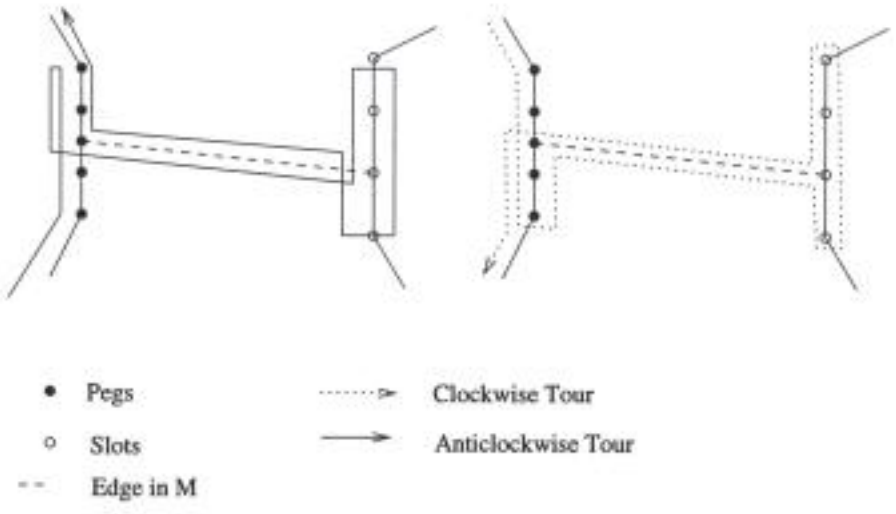


FIG. 4.1. *Two tours in T_p.*

Figure 4.1 illustrates why each segment of the tour T_p is charged at most four times by the two tours. Observe that each segment of T_p is traversed once in one of the tours and thrice in the other tour.

4.2. An improved nonpreemptive algorithm. We now present an improved algorithm that obtains an approximation ratio of 6.5.

1. Find tours T_p and T_s as before.
2. Break T_p and T_s into paths containing $k/2$ vertices each.
3. View each segment of T_p and T_s as a supernode and construct the auxiliary bipartite graph as before.
4. Find a minimum-weight perfect matching M in this auxiliary graph. Mark the segments of T_p sequentially as $B_1, B_2, \ldots$ around the cycle. Each edge in M matches a segment in T_p to a segment in T_s. Let the segment matched to $B_i, i = 1, 2, \ldots$, be called R_i, and let M_i be the edge of M connecting B_i and M_i.
5. The delivery schedule is as follows. Assume that the vehicle starts at the beginning of segment B_1 with $k/2$ pegs, and proceeds along T_p, picking the pegs in its path. For $i = 1, 2, \ldots$, when it reaches the vertex in B_i incident to the matched edge (M_i) in M, it travels across this edge and delivers to R_i the pegs that were collected from B_{i-1}. (When $i = 1$ we deliver the pegs we started with.) After delivering the pegs, it retraces back on M_i and continues along T_p.
6. Finally, when the vehicle returns to the starting location, it is carrying $k/2$ pegs. Lemma 2.3 guarantees that there exists a valid starting point on this traversal such that the vehicle never runs out of pegs or exceeds its carrying capacity.

The algorithm above generates a valid vehicle routing without violating the capacity constraints of the vehicle for the following reason. When it is on a segment

of B_i that precedes M_i on T_p, it is carrying $k/2$ pegs from B_{i-1} and the other pegs that have been collected from B_i. Since B_i has at most $k/2$ pegs, the total number of pegs that it is carrying does not exceed k. On reaching M_i, the vehicle goes to R_i and delivers the $k/2$ pegs that were collected from B_{i-1}. Therefore, when it returns to B_i and resumes its journey, it reaches the end of B_i with $k/2$ pegs that were on B_i.

The reason that the algorithm gives a better approximation ratio is as follows. Observe that the algorithm goes around T_p only once (except for segment B_1) instead of twice. This decreases the length traveled. However, the cost of M is now more since there are twice as many segments as before in each of T_s and T_p (because the segments have only $k/2$ vertices each).

Extending the analysis from the previous algorithm, we get $M \leq C'_k$. The vehicle traverses T_p once, T_s twice, and M twice. We get the following result: The distance traveled by the vehicle in the scheme devised by the above algorithm is at most $T_p + 2T_s + 2M+ \leq 6.5C_k$. Therefore, the approximation ratio is at most 6.5.

4.3. An improved preemptive algorithm. In this section, we consider the vehicle routing problem when the vehicle is allowed to drop some pegs at intermediate points in the route and pick them up later for delivery. We show that the previous algorithm can be modified into a preemptive algorithm with an approximation ratio of a little over 5.

In the previous algorithm, instead of delivering the pegs directly to the slots in R_i, when we cross M_i, we do the following: The vehicle crosses M_i and leaves $k/2$ pegs there to be delivered to the slots later. Once the tour along the peg cycle is completed, the vehicle switches to the slot tour, and delivers the pegs, but this time picking them up at intermediate points on the T_s as it reaches them. In all, the vehicle travels around each of the cycles T_p and T_s once each, twice around M, and once extra on the segments B_1 and R_1. By selecting B_1 and R_1 appropriately, over all possible segments, our algorithm obtains a tour with a ratio of $5 + k/n \approx 5$.

Acknowledgments. We are grateful to Greg Frederickson for useful discussions. We thank Shoshana Anily and Julien Bramel for making their paper available to us. We also thank Rafi Hassin for useful comments on an earlier draft of this paper.

REFERENCES

[1] S. Anily and J. Bramel, *Approximation algorithms for the capacitated traveling salesman problem with pick-ups and deliveries*, Naval Res. Logist., 46 (1999), pp. 654–670.

[2] S. Anily and R. Hassin, *The swapping problem*, Networks, 22 (1992), pp. 419–433.

[3] E. Arkin, R. Hassin, and L. Klein, *Restricted delivery problems on a network*, Networks, 29 (1997), pp. 205–216.

[4] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problem*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 2–11.

[5] S. Arora, *Nearly linear time approximation schemes for Euclidean TSP and other geometric problems*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 554–563.

[6] T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama, *Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 275–283.

[7] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with applications to minimizing robot arm travel*, SIAM J. Comput., 17 (1988), pp. 849–869.

[8] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier, New York, 1977.

[9] D. O. Casco, B. L. Golden, and E. A. Wasil, *Vehicle routing with backhauls: Models, algorithms and case studies*, in Vehicle Routing: Methods and Studies, B. Golden and A. Assad, eds., North-Holland, Amsterdam, 1988, pp. 127–147.

[10] P. Chalasani, R. Motwani, and A. Rao, *Algorithms for robot grasp and delivery*, in Procedings of the 2nd International Workshop on Algorithmic Foundations of Robotics, Toulouse, France, 1996.

[11] N. Christofides, *Vehicle routing*, in The Traveling Salesman Problem, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley and Sons, New York, 1985, pp. 431–448.

[12] G. N. Frederickson, *A note on the complexity of a simple transportation problem*, SIAM J. Comput., 22 (1993), pp. 57–61.

[13] G. N. Frederickson and D. J. Guan, *Non-preemptive ensemble motion planning on a tree*, J. Algorithms, 15 (1993), pp. 29–60.

[14] G. N. Frederickson and D. J. Guan, *Preemptive ensemble motion planning on a tree*, SIAM J. Comput., 21 (1992), pp. 1130–1152.

[15] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, SIAM J. Comput., 7 (1978), pp. 178–193.

[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, 1978.

[17] M. Haimovich and A. H. G. Rinnooy Kan, *Bounds and heuristics for capacitated routing problems*, Math. Oper. Res., 10 (1985), pp. 527–542.

[18] R. M. Karp, *Two combinatorial problems associated with external sorting*, in Combinatorial Algorithms, Courant Comput. Sci. Sympos. 9, Algorithmics Press, New York, 1973, pp. 17–29.

[19] D. E. Knuth, *The Art of Computer Programming, Vol.* 3*: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[20] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., 28 (1999), pp. 1298–1309.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 683–705

OPTIMAL SAMPLING STRATEGIES IN QUICKSORT AND QUICKSELECT*

CONRADO MARTÍNEZ† AND SALVADOR ROURA†

Abstract. It is well known that the performance of quicksort can be improved by selecting the median of a sample of elements as the pivot of each partitioning stage. For large samples the partitions are better, but the amount of additional comparisons and exchanges to find the median of the sample also increases. We show in this paper that the optimal sample size to minimize the average total cost of quicksort, as a function of the size n of the current subarray size, is $a \cdot \sqrt{n} + o(\sqrt{n})$. We give a closed expression for a, which depends on the selection algorithm and the costs of elementary comparisons and exchanges. Moreover, we show that selecting the medians of the samples as pivots is not the best strategy when exchanges are much more expensive than comparisons. We also apply the same ideas and techniques to the analysis of quickselect and get similar results.

Key words. quicksort, quickselect, sorting, selection, sampling, median-of-$(2k+1)$, analysis of algorithms, divide-and-conquer

AMS subject classifications. 68P10, 68Q25, 68W05, 68W40

PII. S0097539700382108

1. Introduction.

Quicksort [8] and quickselect [7] are among the most thoroughly studied algorithms. Quicksort sorts an array A of n elements by rearranging A around one of its elements—the *pivot*—so that the elements smaller than the pivot are to its left and larger elements are to its right. After the array has been partitioned, quicksort is recursively applied to the subarrays on either side of the pivot. Quickselect[1] selects the mth element (equivalently, the element of *rank* m in ascending order, the mth order statistic) out of n using the same divide-and-conquer principle. Once the array is partitioned and the pivot brought into its correct position, say j, either the sought element is the pivot ($m = j$) or the algorithm recursively continues in the appropriate subarray: the left subarray if $m < j$, the right subarray if $m > j$. Excellent sources for background information, implementation, variants, and their analysis include [3, 12, 13, 22, 23, 24].

In *median-of-three quicksort* [25] the pivot of each recursive stage is the median of a sample of three elements. This variant is easily generalized to select the $(k+1)$th element of a sample of size $s = 2k+1$ as the pivot. Van Emden [26] showed that the average number of comparisons to sort an array of size n is $q(k) \cdot n \ln n + \mathcal{O}(n)$, where $q(k)$ steadily decreases from $q(0) = 2$ to $q(\infty) = 1/\ln 2$. The median-of-three strategy also improves the performance of quickselect [10].

The basic problem with large samples is that the savings achieved because of more balanced partitions can be swamped in practice by the time spent in finding the median of the samples. This cost shows up in the lower order terms of the

*Received by the editors December 8, 2000; accepted for publication (in revised form) June 12, 2001; published electronically October 23, 2001. This research was supported by ESPRIT LTR 20244 (ALCOM-IT), CICYT TIC97-1475-CE, DGES PB95-0787 (KOALA), DGES PB98-0926 (AEDRI), and CIRIT 1997SGR-00366 (SGR).

http://www.siam.org/journals/sicomp/31-3/38210.html

†Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, E-08034 Barcelona, Catalonia, Spain (conrado@lsi.upc.es, roura@lsi.upc.es). The first author was also supported by ACI-CONICYT DOG 2320 (Catalonia-Chile Cooperation Program) and CYTED (RITOS Network).

[1] Also known as Hoare's FIND algorithm.

average performance of quicksort and quickselect and cannot be disregarded unless n is impractically large.

McGeoch and Tygar [17] analyzed the expected number of comparisons of quicksort for various sampling strategies. They considered fixed-size sampling and hybrid strategies, which first use a sample whose size is a function of the size of the array to be sorted, and fixed-size samples in subsequent recursive calls. They proved that these strategies are worse than using samples of size $\Theta(\sqrt{n}\,)$, where n denotes the size of the (sub-)array to be sorted in each recursive invocation.

We show in this paper that the optimal sample size as a function of the size n of the current subarray is $a \cdot \sqrt{n} + o(\sqrt{n}\,)$ and determine the value of the constant a in terms of the cost of the median-finding algorithm, taking into account the cost of comparisons and exchanges. Furthermore, we consider the more general setting where we pick the $(p+1)$th element in the sample with $0 \le p < \lfloor s/2 \rfloor$. (The case for $\lfloor s/2 \rfloor \le p < s$ reduces to the former by symmetry.)

This paper is organized as follows. In section 2 we set up the basic recurrences for quicksort and quickselect. In section 3 we analyze the variants with fixed-size sampling. We rederive known results such as those of van Emden [26] and those of quickselect with median-of-three [10]. As far as we know, our results regarding the total cost of quicksort and of quickselect with fixed $s > 3$ are original.

In section 4 we study the surprising behavior of quicksort when the cost of an exchange exceeds by far that of a comparison: on the one hand, the optimal value of p is not $p = \lfloor (s-1)/2 \rfloor$ but a function of the ratio between the cost of an exchange and the cost of a comparison; on the other hand, if we always select the medians of the samples as pivots, then fixed-size sampling is better than using samples whose sizes grow with n.

In sections 5 and 6 we tackle the study of the variants with sample sizes depending on n. We analyze quickselect first, since its analysis is easier. We prove in section 5 that if the size of the sample grows with n and is $o(n)$, then the average number of comparisons and exchanges to select an item of random rank are $2n + o(n)$ and $n/2 + o(n)$, respectively. By analyzing the lower order terms, we prove that the optimal sample size is $\Theta(\sqrt{n}\,)$ and give an explicit formula for the constant factor of the main term, which depends on the cost of the algorithm to select pivots and on the cost of elementary comparisons and exchanges.

We consider quicksort in section 6. We prove that when $s = \omega(1)$ and $s = o(n)$ the average number of comparisons is $n \log_2 n + o(n \log n)$ and the average number of exchanges is $\frac{1}{4} n \log_2 n + o(n \log n)$. Using similar techniques to those in section 5 we prove that the optimal sample size for quicksort is $\Theta(\sqrt{n}\,)$ as well and also find the constant factor of the main term. Moreover, we prove that the best pivots are the medians of the samples only when exchanges are not too expensive.

In section 7 we show that using fixed-size samples reduces the constant factor in the main term $\Theta(n^2)$ of the variance of quickselect. Another result with practical implications is that the variance of quickselect is $\mathcal{O}(\max\{n \cdot s, n^2/s\})$ when $s = \omega(1)$. For quicksort, we conjecture that similar results hold. We also discuss tuned implementations of partitioning for quicksort and quickselect, which avoid making redundant comparisons and exchanges.

A preliminary version of this work appears in [15] (see also [16, 20]). Some of the material there is also presented here, in order to make this paper more self-contained.

2. Preliminaries. A basic assumption of this paper is that s, the size of the samples, is a function of the size n of the (sub-)array to be sorted. We will not write

down the dependence of s on n most of the time, though. We also assume $s(n) = o(n)$ and, of course, $s(n) \geq 1$. Finally, we assume that the given array contains a random permutation of distinct elements.

We will denote p (respectively, q) the number of elements in the sample smaller than (respectively, larger than) the pivot. Therefore the pivot is the $(p+1)$th element $(0 \leq p < s)$ in the sample, and $q = s - 1 - p$. For the particular case where the pivot is the median of a sample of odd size, we write $s = 2k + 1$, where $k + 1$ is the rank of the pivot, with $p = q = k$.

Most analyses of quicksort and quickselect consider only the number of comparisons. In this paper, we investigate the average *total cost*, that is, the sum of the cost of comparisons and exchanges. We take the cost of a single comparison as the unit and denote ξ the cost of an exchange relative to that of a comparison.

Both quicksort and quickselect operate by selecting a pivot and then partitioning the current subarray w.r.t. the pivot. This requires $n + 1$ comparisons irrespective of s and p.[2] Let $\mathsf{X}(n, s, p)$ denote the average number of exchanges during a single partition stage. We consider here a very common partitioning algorithm, which scans the subarray from left to right until an element larger than the pivot is found, then from right to left until an element smaller than the pivot is found, then exchanges the two designated elements, and then iterates the process again. The partitioning ends when the two scanning pointers meet [22, 23, 24].[3] The following lemma provides the value of $\mathsf{X}(n, s, p)$ for any n, s, and p such that $0 \leq p < s \leq n$. Its proof is given in Appendix B.

LEMMA 1. *The average number of exchanges to partition a random array of size n when the pivot is the $(p+1)$th element of a sample of s elements is*

$$\mathsf{X}(n, s, p) = \frac{(p+1)(q+1)}{(s+1)(s+2)} \frac{(n+1)(n+2)}{n-1} - \frac{n}{n-1}.$$

Let $\mathsf{S}(s, p)$ denote the average total cost of the algorithm to select the $(p+1)$th out of s elements. (This algorithm may or may not be quickselect or one of its variants.) Efficient selection algorithms work in linear time on the average, so we can safely assume $\mathsf{S}(s, p) = \beta \cdot s + o(s)$ for some constant β that depends on the selection algorithm, the costs of comparisons and exchanges, and typically on the ratio $\psi = p/s$. For instance, if we count only comparisons ($\xi = 0$), then $\beta = \frac{3}{2} - \left|\frac{1}{2} - \psi\right|$ for Floyd–Rivest's SELECT [4], $\beta = 2 - 2(\psi \ln \psi + (1-\psi)\ln(1-\psi))$ for Hoare's quickselect [11], and $\beta = 2 + 3\psi(1-\psi)$ for quickselect with median-of-three [10]. An important assumption that we make is that the selection process preserves the randomness of the input array, copying the sample to a separate area if necessary. Otherwise, the recurrences below would not be accurate (see section 7.2).

Let $\pi_{n,j}^{(s,p)}$ be the probability that the $(p+1)$th element of a sample of s elements is the $(j+1)$th element of a random permutation of size n. It is clear that

$$\pi_{n,j}^{(s,p)} = \frac{\binom{j}{p}\binom{n-1-j}{q}}{\binom{n}{s}}, \qquad 0 \leq p < s \leq n, \quad 0 \leq j < n. \tag{2.1}$$

[2] If partitioning uses sentinels, then the number of comparisons can be reduced to $n - 1$, but the main conclusions of this work do not significantly change.

[3] For other randomness-preserving partitioning schemes (e.g., Lomuto's partition, as cited by Bentley [2]) the main conclusions of this paper apply with minor quantitative differences.

The denominator is the number of ways to pick a sample of size s out of n elements; the numerator is the number of ways to choose p elements smaller than the pivot times the number of ways to choose q elements larger than the pivot.

Now we are ready to set the recurrences for Q_n, the average total cost of quicksort, and, for F_n, the average total cost of quickselect averaged w.r.t. the n possible values of the rank of the sought element. F_n has been investigated in [14, 18, 19] as a particular case of *multiple quickselect* with $p = 1$ in the two last references. (There, p is the number of sought elements, not the number of elements smaller than the pivot within the sample.) The term *grand average* has been used to describe F_n and its generalizations for multiple quickselect (see, for instance, [13, 19]). Let $\mathsf{F}_n^{(m)}$ denote the cost of selecting the mth element out of n with quickselect and let F_n denote the cost of selecting an element of random rank. Then

$$F_n = \mathbb{E}[\mathsf{F}_n] = \frac{1}{n} \sum_{1 \le m \le n} \mathbb{E}\left[\mathsf{F}_n^{(m)}\right],$$

but the analogous equation does not hold for the random variables.

For quicksort using samples of size s, the nonrecursive cost of selecting the pivot (the pth element of the sample) and partitioning is

$$\mathsf{S}(s,p) + n + 1 + \mathsf{X}(n,s,p) \cdot \xi,$$

and hence

$$\begin{aligned} Q_n &= \mathsf{S}(s,p) + n + 1 + \mathsf{X}(n,s,p) \cdot \xi \\ &\quad + \sum_{0 \le j < n} \Pr\{\text{the pivot is the } (j+1)\text{th}\} \cdot (Q_j + Q_{n-1-j}) \\ &= \mathsf{S}(s,p) + n + 1 + \mathsf{X}(n,s,p) \cdot \xi + \sum_{0 \le j < n} \left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) \cdot Q_j, \end{aligned} \tag{2.2}$$

where we have used obvious symmetries in the last step.

The recurrence for F_n is a bit more involved. The nonrecursive cost is the same as in quicksort. However, the probability that a recursive call with an instance of size j is made is now $\pi_{n,j}^{(s,p)} \cdot \frac{j}{n}$, as we are looking for an element of random rank. Hence, applying the same symmetries as in the recurrence for quicksort,

$$F_n = \mathsf{S}(s,p) + n + 1 + \mathsf{X}(n,s,p) \cdot \xi + \sum_{0 \le j < n} \frac{j}{n} \left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) \cdot F_j. \tag{2.3}$$

3. Fixed-size samples. We consider here the solution of (2.2) and (2.3) when $s = \Theta(1)$. This analysis is almost straightforward using the continuous master theorem (CMT) [20, 21]. We briefly review this theorem (Theorem 18) and the necessary definitions in Appendix A for the reader's convenience.

First of all, the *toll function* (the nonrecursive cost) t_n in the recurrences for Q_n and F_n is

$$t_n = \left(1 + \frac{(p+1)(q+1)}{(s+1)(s+2)} \cdot \xi\right) \cdot n + \mathcal{O}(1).$$

Notice that $\mathsf{S}(s,p)$ and several other terms are accounted for in the $\mathcal{O}(1)$-term above. Now we tackle the analysis of quicksort. The first step to apply the CMT

requires that we compute the *shape function* $w^{(s,p)}(z)$ of the recurrence, which essentially is a continuous approximation of the discrete weights. In our instance we have

$$w^{(s,p)}(z) = \lim_{n\to\infty} n \cdot \left(\pi^{(s,p)}_{n,z\cdot n} + \pi^{(s,p)}_{n,n-1-z\cdot n}\right) = \frac{s!}{p!\,q!}\left(z^p(1-z)^q + z^q(1-z)^p\right).$$

Since t_n is linear, the limiting const-entropy is evaluated as $\mathcal{H}(s,p) = 1 - \int_0^1 z^1\, w^{(s,p)}(z)\, dz = 0$ for all s and p. Hence, Case 2 of the CMT states that $Q_n = t_n \cdot \ln n / \operatorname{VE}(s,p)$, where $\operatorname{VE}(s,p)$ denotes the limiting log-entropy. We use $\operatorname{VE}(s,p)$ for this value after van Emden, who first used this function for the particular case where $p = q = k$ and $\xi = 0$ [26]. Proposition 19(b) in Appendix B is useful in the following computation:

$$\operatorname{VE}(s,p) \stackrel{\triangle}{=} -\int_0^1 z^1 \ln z \cdot w^{(s,p)}(z)\, dz = H_{s+1} - \frac{(p+1)H_{p+1} + (q+1)H_{q+1}}{s+1},$$

where $H_n = \sum_{1\le j\le n} 1/j = \ln n + \gamma + \mathcal{O}(1/n)$ denotes the nth *harmonic number*, and $\gamma = 0.577\ldots$ is Euler's constant. Therefore, $Q_n = q_\xi(s,p)\cdot n\ln n + o(n\log n)$, where

$$q_\xi(s,p) \stackrel{\triangle}{=} \left(1 + \frac{(p+1)(q+1)}{(s+1)(s+2)}\cdot\xi\right) / \operatorname{VE}(s,p). \tag{3.1}$$

It is possible to get more information about Q_n by subtracting the main order term from Q_n and setting up a recurrence for the remaining part, i.e., for $R_n = Q_n - q_\xi(s,p)\, nH_n$. Applying the CMT again we get $R_n = \mathcal{O}(n)$. As a conclusion we have the following theorem.

THEOREM 2. *The average total cost of quicksort with fixed-size samples is* $Q_n = q_\xi(s,p)\cdot n\ln n + \mathcal{O}(n)$.

The analysis of quickselect follows similar steps. Computing the shape function yields[4] $w^{(s,p)}(z) = s!/(p!\,q!)\left(z^{p+1}(1-z)^q + z^{q+1}(1-z)^p\right)$. The limiting const-entropy is now

$$\mathcal{H}(s,p) = 1 - \frac{(p+1)(p+2)+(q+1)(q+2)}{(s+1)(s+2)} = \frac{2(p+1)(q+1)}{(s+1)(s+2)},$$

which is strictly positive for every s and p. Therefore $F_n = f_\xi(s,p)\cdot n + o(n)$, where

$$f_\xi(s,p) \stackrel{\triangle}{=} \left(1 + \frac{(p+1)(q+1)}{(s+1)(s+2)}\cdot\xi\right) / \mathcal{H}(s,p) = \frac{(s+1)(s+2)}{2(p+1)(q+1)} + \frac{\xi}{2}.$$

As before, we can estimate the lower order terms of F_n by iteratively using the CMT. Applying it twice, we get a precise asymptotic estimate for the second order term and the order of magnitude of the third order term (see [16] or [20] for additional details).

THEOREM 3. *The average total cost of quickselect with fixed-size samples is*

$$F_n = f_\xi(s,p)\cdot n + \left(\mathsf{S}(s,p) - 2 - \xi - \frac{3(q+1) - p(q+5)}{(s+1)(s+2)}\cdot\xi\right)\cdot\frac{\ln n}{VE(s,p)} + \mathcal{O}(1).$$

[4] We use the same symbol to denote quickselect's shape function and quicksort's shape function, even though they are different.

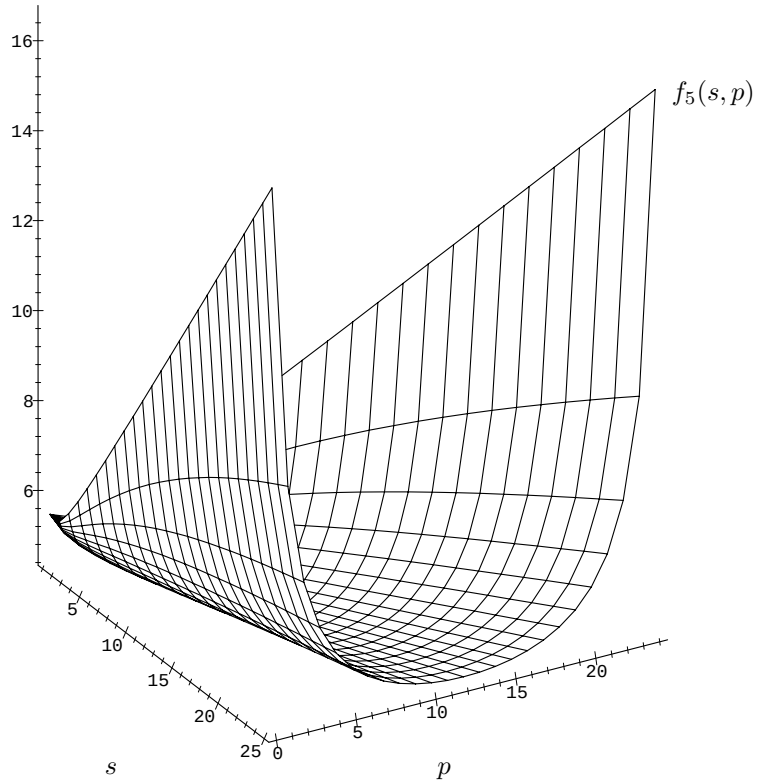


FIG. 1. *Values of* $f_5(s,p)$.

In section 4 we examine in depth the behavior of $q_\xi(s,p)$, the constant factor of the main term of Q_n. The behavior of $f_\xi(s,p)$ in the main term of F_n is much simpler. For any fixed s, the value of p that minimizes $f_\xi(s,p)$ is $p=(s-1)/2$ if s is odd, and $p=(s-2)/2$ (also $p=s/2$ by symmetry) if s is even, regardless of the value of ξ. Figure 1 depicts a typical situation with $\xi=5$. Moreover, for any $k\geq 0$ we have that $f_\xi(2k+1,k)=f_\xi(2k+2,k)=2+1/(k+1)+\xi/2$ is a decreasing function of k. Hence, our best choice for p is always the median of the sample and to let s be "as large as possible." In section 5 we precisely characterize the best way to do this.

4. Exchanges in quicksort with fixed-size sampling. In this section we study which values of s and p minimize the constant factor $q_\xi(s,p)$ of the $n\ln n$-term in the average cost of quicksort with fixed-size sampling. Given s and ξ, let $p^*(s,\xi)$ be the rank of the optimal pivot. For moderately large values of ξ (this includes the case $\xi=0$, where we take into account only comparisons) the optimal pivot is $p^*=\lfloor (s-1)/2\rfloor$, and $q_\xi(2k+1,k)$ is a strictly decreasing function of k. By contrast, for large values of ξ the best fixed-size strategy is indeed to have large samples, but the pivot should be an element of rank $p^*\neq\lfloor (s-1)/2\rfloor$.

For convenience we will work with the quotients $\psi=p/s$ rather than with the actual ranks p for the rest of this section. Figure 2 shows the function $q_{30}(s,p)$ with two valleys symmetrically disposed at both sides of the line $\psi=1/2$. For each s, we take the smallest p with minimum $q_{30}(s,p)$ to define ψ^*, which is clearly smaller than

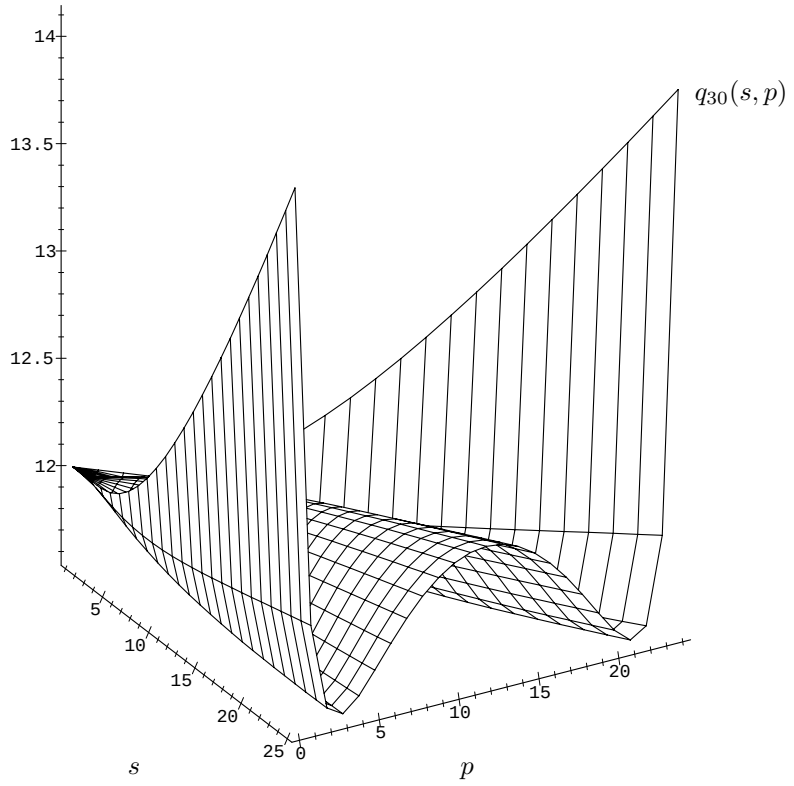


FIG. 2. *Values of* $q_{30}(s,p)$.

1/2.

In Figure 3 we plot the (scaled) values of the function

$$q_\xi(\psi) \stackrel{\triangle}{=} \lim_{s\to\infty} q_\xi(s, \psi\cdot s + o(s)) = -\frac{1+\psi(1-\psi)\xi}{\psi\ln\psi + (1-\psi)\ln(1-\psi)}$$

for $\xi = 0, 5, 15, 20$, and 30. Note that the denominator is, besides the limiting value $\lim_{s\to\infty} \mathrm{VE}(s, \psi\cdot s)$, the entropy for the probabilities ψ and $1-\psi$ (with the minus sign). It appears often enough to deserve a special name: for $0 < x < 1$, let

$$h(x) \stackrel{\triangle}{=} -x\ln x - (1-x)\ln(1-x). \tag{4.1}$$

The function is symmetric around $\psi = 1/2$ with only one minimum located at $\psi^* = 1/2$ when ξ is not larger than a threshold value τ; otherwise, we have a local maximum at $\psi = 1/2$ and two absolute minima at $\psi^* < 1/2$ and at $1-\psi^* > 1/2$. In this last case ψ^* is the unique solution in the interval $(0, 1/2)$ of the equation $\partial q_\xi(\psi)/\partial\psi = 0$, or, equivalently, of

$$\ln\psi + \xi\,\psi^2\ln\psi = \ln(1-\psi) + \xi(1-\psi)^2\ln(1-\psi). \tag{4.2}$$

The threshold value τ is given by the solution of the equation

$$\left.\frac{\partial^2 q_\xi(\psi)}{\partial\psi^2}\right|_{\psi=1/2} = 0,$$

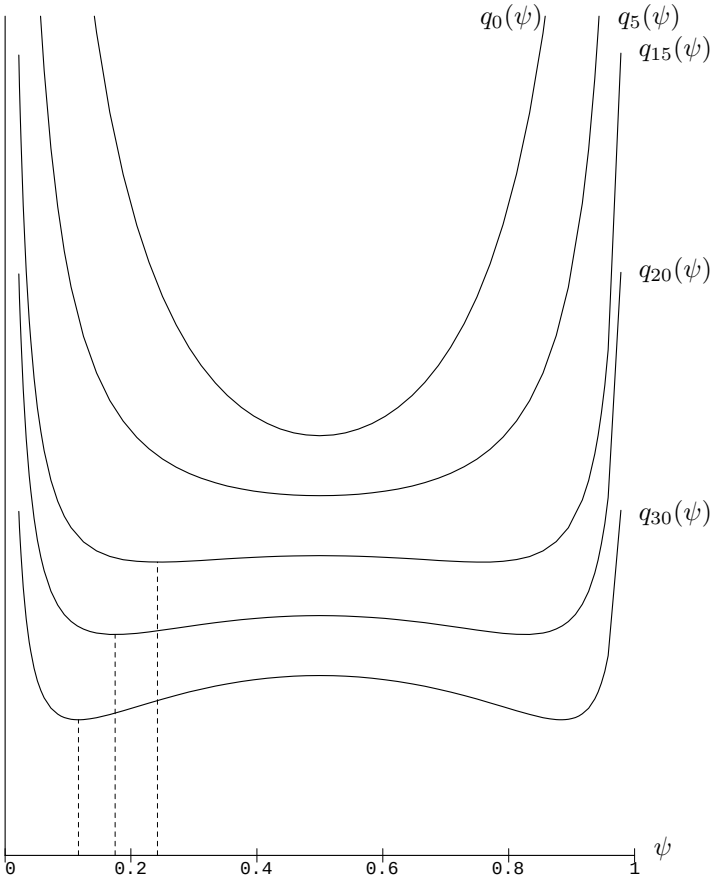


FIG. 3. *Values of* $q_\xi(\psi)$ *for* $\xi = 0, 5, 15, 20,$ *and* 30.

which is $\tau = 4/(2\ln 2 - 1) \approx 10.35480$. In Figure 4 we see that $\psi^*(\xi) = 1/2$ for $\xi \in [0, \tau]$, whereas $\psi^*(\xi)$ strictly decreases for $\xi > \tau$, tending to 0 as ξ grows.

For large s we know that p^* is roughly $\psi^* \cdot s$. However, now the question is whether taking large samples is the best choice (on the asymptotic regime) among fixed-size sampling strategies. The next theorem states just that.

THEOREM 4. *For any* s *and* p, $q_\xi(s,p) > q_\xi(\psi^*)$.

Proof. Let $u(z) = s!/p!\,q! \cdot z^p(1-z)^q$. From (3.1) and Proposition 19, we have

$$q_\xi(s,p) = \frac{\int_0^1 u(z)\,(1 + z(1-z)\xi)\,dz}{\int_0^1 u(z)h(z)\,dz}.$$

We will use the following fact: Let $f(z)$, $g(z)$ be positive functions over the interval $[0,1]$. Let z^* be the location of a minimum of $f(z)/g(z)$, and assume $0 < z^* < 1$ and $g(z) \neq 0$ for $0 < z < 1$. Then, as $f(z) \geq g(z)f(z^*)/g(z^*)$ and $u(z)$ is also positive in the interval $[0,1]$, it follows that

$$\frac{\int_0^1 u(z)f(z)\,dz}{\int_0^1 u(z)g(z)\,dz} \geq \frac{\int_0^1 u(z)\,[g(z)f(z^*)/g(z^*)]\,dz}{\int_0^1 u(z)g(z)\,dz} = \frac{f(z^*)}{g(z^*)}.$$

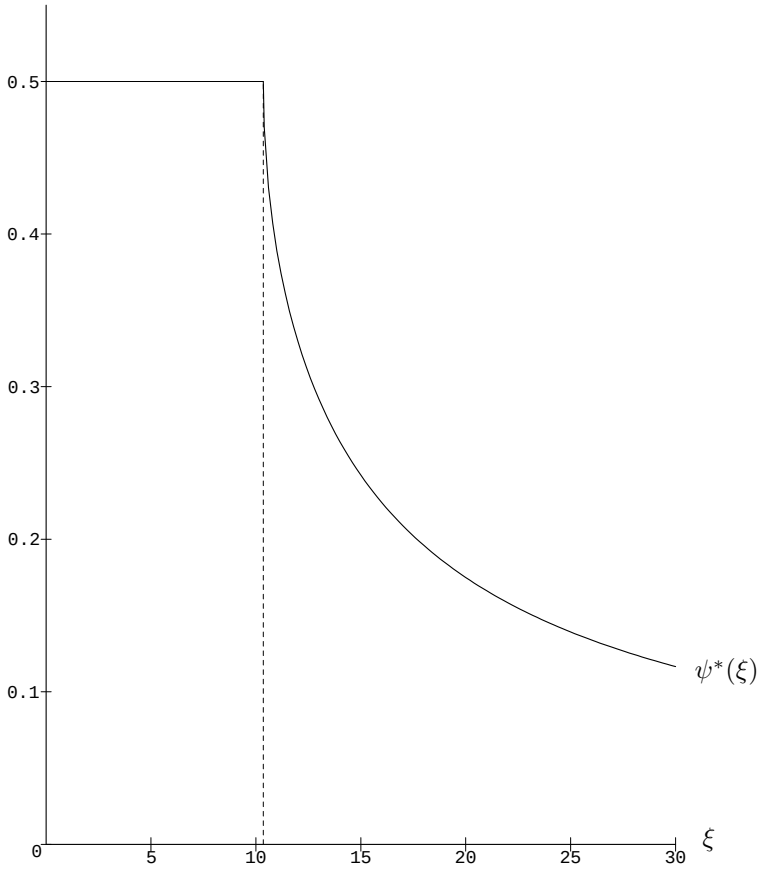


FIG. 4. *Values of* $\psi^*(\xi)$.

Now taking $f(z) = 1+z(1-z)\xi$, $g(z) = h(z) = -z\ln z-(1-z)\ln(1-z)$, and since ψ^* is the minimum of $q_\xi(\psi) = f(\psi)/g(\psi)$, we have $q_\xi(s,p) \geq q_\xi(\psi^*)$, and the statement of the theorem is almost proved. Finally, notice that for any ξ there is always an interval $[\psi_1, \psi_2]$ with $0 \leq \psi_1 < \psi_2 \leq 1$ such that $q_\xi(\psi) > q_\xi(\psi^*(\xi))$ and $u(z) > 0$ for every ψ in (ψ_1, ψ_2). Thus $q_\xi(s,p) > q_\xi(\psi^*)$. □

An intuitive explanation for the theorem above goes as follows. Assume that we had a black-box routine such that, given an array of size n and a value $0 < \psi < 1$, it returned the $\lfloor\psi \cdot n\rfloor$th element in the array *at no cost*. Using such a routine (with some fixed ψ) in the pivot-finding stage of quicksort provides a variant of quicksort with toll function $(1 + \psi(1-\psi)\xi)$, log-entropy $h(\psi)$, and average total cost $q_\xi(\psi)\, n\ln n + o(n\log n)$. Thus the optimal choice for ψ is ψ^*. Now, as we are deprived from such a routine, we try to get the best possible estimate of the $\lfloor\psi^* \cdot n\rfloor$th element in the array, which can be done by taking large samples and selecting the $\lfloor\psi^* \cdot s\rfloor$th element.

Several remarks are in order. On the one hand, the fact that ψ^* tends to 0 as ξ tends to ∞ can be informally described as a smooth transition from quicksort to selection sort. Selection sort is a good sorting method when exchanges are extremely expensive because it minimizes their number, and quicksort behaves exactly as selec-

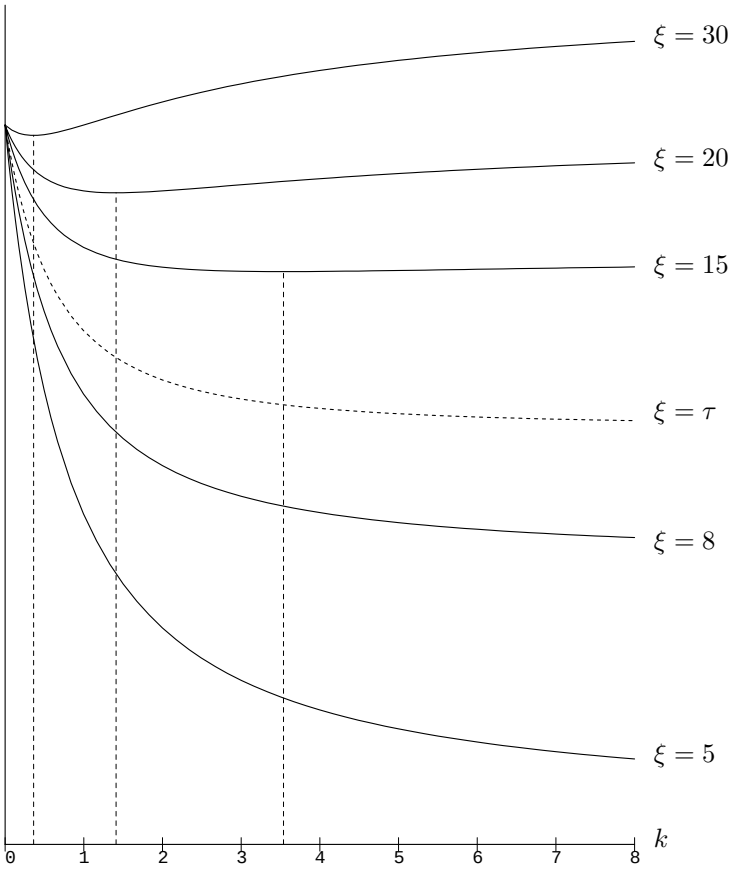


FIG. 5. *Values of* $q_\xi(k)$ *for* $\xi = 5,\ 8,\ \tau,\ 15, 20,$ *and* 30.

tion sort when the smallest element in the array is always selected as the pivot. On the other hand, we should be aware that the analysis of the case $\xi > \tau$ is mainly of theoretical interest, since we should sort an array of pointers to the actual records rather than sorting the records themselves if data movements were too expensive.

Now we restrict our attention to the variants of quicksort that always take the medians of samples of fixed size as the pivots (and therefore are not the best theoretical alternatives when $\xi > \tau$); that is, we take $s = 2k + 1$ and $p = q = k$ for some $k \geq 0$. This time we have

$$q_\xi(k) \stackrel{\triangle}{=} q_\xi(2k+1, k) = \frac{1 + (k+1)/(4k+6) \cdot \xi}{H_{2k+2} - H_{k+1}}.$$

The (scaled) shape of this function is shown in Figure 5 for several values of ξ. The plot actually depicts the extension of $q_\xi(k)$ to the real numbers using the continuous function $\Psi(z) = d \ln \Gamma(z)/dz$; recall that $\Psi(n+1) = H_n - \gamma$ for positive integers [1]. For $\xi = 5$, $\xi = 8$, and $\xi = \tau$ (dashed in the figure) the function $q_\xi(k)$ steadily decreases with k in accordance with what we know. This behavior changes as soon as $\xi > \tau$, as $q_\xi(k)$ has one minimum at finite distance $k^*(\xi)$. Observe in Figure 5 that the location of the minima (which is shown for $\xi = 15$, 20, and 30) tends to 0

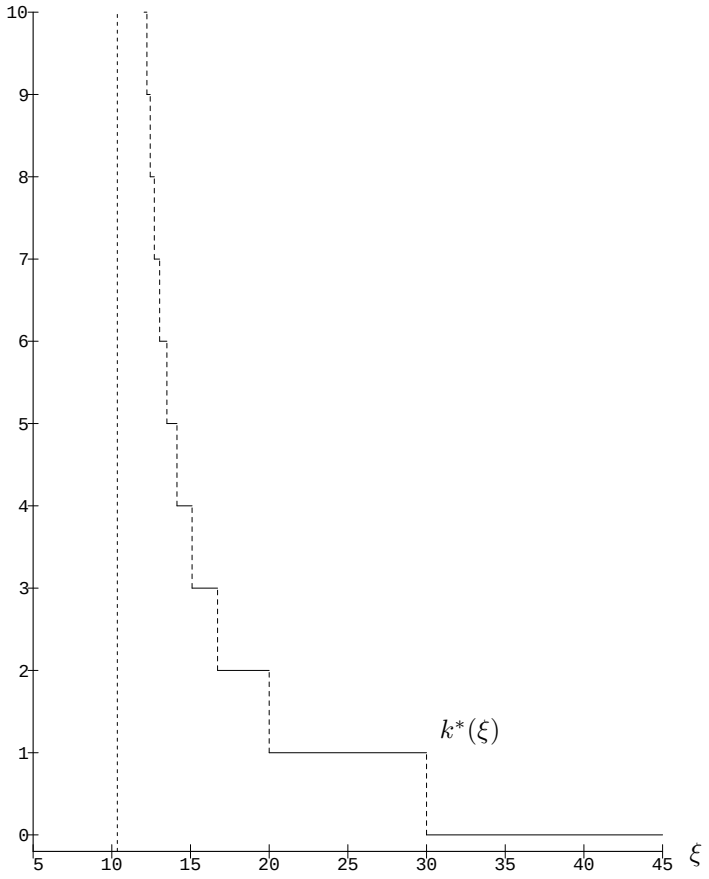


FIG. 6. *Values of* $k^*(\xi)$.

when ξ grows.

Figure 6 shows the function $k^*(\xi)$. There is a vertical asymptote as $\xi \to \tau^+$. For values of ξ larger than 30 we have $k^* = 0$ (plain quicksort). Note that k^* is not well defined for some values of ξ. For instance, 1 and 2 compete as optimal choices for k when $\xi = 20$.

The function

$$\xi^*(k) \triangleq \frac{4}{4(H_{2k} - H_k)\frac{k+1}{2k+3} - 1 + \frac{1}{2k+1}}$$

is the pseudoinverse of $k^*(\xi)$ in the sense that k is the optimal choice if ξ belongs to the open interval $(\xi^*(k+1), \xi^*(k))$. Therefore, $q_{\xi^*(k)}(k) = q_{\xi^*(k)}(k-1)$ for any $k > 0$. For instance, we have $k^* = 1$ when $20 < \xi < 30$ because $\xi^*(2) = 20$ and $\xi^*(1) = 30$. By convention we take $\xi^*(0) = \infty$.

As a conclusion, we could say that a good estimation of the median of the (sub-) array is always profitable as far as comparisons are concerned. However, regarding exchanges we have a trade-off between the short-term profit of selecting a pivot far away from the median (since only a few exchanges would be needed in that stage) and the long-term gains of selecting a pivot closer to the median (since an even

partition means fewer recursive stages and fewer exchanges to be done in the long run). Roughly speaking, this is why small samples—and consequently, poor estimations of the medians—are preferable when the important quantity to minimize is the number of exchanges, i.e., when ξ is large.

5. Optimal samples for quickselect. We already know from section 3 that selecting the medians of the samples is the best choice to minimize the average cost of quickselect. Therefore, we will assume $s = 2k+1$ and $p = q = k$ for the rest of this section, and we will simplify the notation for functions like $\mathsf{X}(n, 2k+1, k)$, writing $\mathsf{X}(n, k)$ instead. Here we will consider the case where $k = k(n)$ is a function of the current subarray size n.

In principle it pays to have samples as large as possible. However, if $s = \Theta(n)$ the average total cost is not optimal, since the cost of the selection of the pivot is of the same order of magnitude as the cost of the partition. Hence, we shift our attention towards samples whose size increases with n but are sublinear, that is, $s = \omega(1)$ and $s = o(n)$.

Under the hypotheses above, $s \to \infty$ when $n \to \infty$, so the coefficient of n in F_n should be the limit of $f_\xi(s, s/2)$ for $s \to \infty$. The following theorem rigorously establishes our intuition. The CMT itself cannot be applied here because the shape function for this case is not well defined; we require more basic results that we also review in Appendix A.

THEOREM 5. *Let $s = 2k+1$ and $p = q = k$, where $k = \omega(1)$ and $k = o(n)$. Then the average total cost of quickselect to find an element of random rank out of n is $F_n = (2 + \xi/2)\, n + o(n)$.*

Proof. Let $F_n(t)$ be the average total cost of quickselect when we use samples of fixed size $2t+1$, and let $\pi^{(t)}_{n,j} = \pi^{(2t+1,t)}_{n,j}$. The const-entropy associated to $F_n(t)$ (see Appendix A) is

$$\mathcal{H}_n(t) = 1 - \frac{2}{n^2} \sum_{0 \le j < n} j^2\, \pi^{(t)}_{n,j} + o(1),$$

which by Proposition 21 in Appendix B has limit $\mathcal{H}(t) = \lim_{n\to\infty} \mathcal{H}_n(t) = 1/2 - 1/(4t+6)$. Fix any $t' \ge t$. Since x^2 is a convex function, the probabilities $\pi^{(.)}_{n,j}$ are symmetric in j and $n-1-j$, and the weights for t' are more concentrated around $(n-1)/2$ than the weights for t, it follows that $\sum_{0\le j<n} j^2 \pi^{(t')}_{n,j} \le \sum_{0\le j<n} j^2 \pi^{(t)}_{n,j}$, and hence $\mathcal{H}_n(t') \ge \mathcal{H}_n(t)$ if n is large enough.

On the other hand, the const-entropy associated to F_n is

$$\mathcal{H}_n = 1 - \frac{2}{n^2} \sum_{0 \le j < n} j^2 \pi^{(k(n))}_{n,j} + o(1).$$

Since $k = \omega(1)$, for any fixed t we have $k(n) > t$ and $\mathcal{H}_n \ge \mathcal{H}_n(t)$ as long as n is large enough. Hence $\mathcal{H} = \lim_{n\to\infty} \mathcal{H}_n \ge \lim_{n\to\infty} \mathcal{H}_n(t) = \mathcal{H}(t)$. Moreover, $\mathcal{H} \ge 1/2 - 1/(4t+6)$ for any t, so $\mathcal{H} \ge 1/2$. Furthermore, the upper bound $\mathcal{H}_n \le 1/2$ can be easily derived using the probabilities

$$\pi_{n,j} = \begin{cases} 1 & \text{if } j = \lfloor (n-1)/2 \rfloor, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $\mathcal{H} = 1/2$. Since under the hypotheses the toll function is $t_n = (1 + \xi/4) \cdot n + o(n)$, we can conclude from (A.2) that $F_n = (1 + \xi/4)\, n/\,\mathcal{H} + o(n) = (2 + \xi/2)\, n + o(n)$. ∎

If we consider only comparisons ($\xi = 0$) Theorem 5 states that the average cost of quickselect is $2n+o(n)$, well above the $1.25n+o(n)$ comparisons needed on average to locate an element of random rank using Floyd and Rivest's SELECT algorithm [4] but well below the $3n + o(n)$ comparisons of standard quickselect. Using rather different techniques than ours, Grübel [6] has shown that $\mathsf{W}_n = \sup_{1\le m\le n}\{\mathsf{F}_n^{(m)}\}$ converges in probability to $2n$ if $k = \omega(1)$ and $k = o(\frac{n}{\log n})$.

Theorem 5 leaves open which is the optimal sample size. To find it we introduce the function

$$\mathcal{F}(n,k) = -\frac{1}{2} + \sum_{0\le j<n} \frac{j^2}{n^2}\left(\pi_{n,j}^{(k)} + \pi_{n,n-1-j}^{(k)}\right) = -\frac{1}{2} + \frac{2}{n^2}\sum_{0\le j<n} j^2\pi_{n,j}^{(k)},$$

which is only a slight variation of the const-entropy associated to F_n (see the proof of Theorem 5). As $k = \omega(1)$ and $k = o(n)$, we have $F_n = (2+\xi/2)\cdot n + G_n$ for some function $G_n = o(n)$. Substituting in the recurrence for F_n we get

$$F_n = n+1+\mathsf{S}(k)+\mathsf{X}(n,k)\cdot\xi+2\sum_{0\le j<n}\frac{j}{n}\pi_{n,j}^{(k)}G_j+(1+\xi/4)\cdot n+(2+\xi/2)\cdot\mathcal{F}(n,k)\cdot n.$$

The asymptotic behavior of $\mathsf{X}(n,k)$ and $\mathcal{F}(n,k)$ when k grows with n is stated in the following lemmas.

LEMMA 6. *If $k = \omega(1)$ and $k = o(n)$, then*

$$\mathsf{X}(n,k) = \frac{n}{4} - \frac{n}{8k} + \Theta\left(\frac{n}{k^2}\right).$$

Proof. Setting $s = 2k+1$ and $p = k$ in Lemma 1 yields

$$\mathsf{X}(n,k) = \frac{(k+1)n^2 - (k+3)n + (2k+2)}{2(2k+3)(n-1)}.$$

Now it is a simple matter to rewrite the equality above to get the statement of the lemma. □

LEMMA 7. *If $k = \omega(1)$ and $k = o(n)$, then*

$$\mathcal{F}(n,k) = \frac{1}{4k} + \Theta(\max\{1/k^2, 1/n\}).$$

Proof. Setting $s = 2k+1$ and $p = k$ in Proposition 21 of Appendix B we get

$$\mathcal{F}(n,k) = \frac{1}{4k} - \frac{3}{4k(2k+3)} - \frac{3}{2n} + \frac{3}{2(2k+3)n} + \frac{1}{(2k+3)n^2}.$$ □

Since $G_n = o(F_n)$, it is rather intuitive that the contribution of the sum of G_j's to the asymptotic location of k^* is irrelevant. The argument can be formalized as follows. Fix any $\delta > 0$. By hypothesis, there exists some N such that $|G_j| \le \delta\cdot j$ for every $j \ge N$. On the other hand, we are assuming $k = \omega(1)$, so for large n we have $k \ge N$ and $\pi_{n,j}^{(k)} = 0$ for every $j < N$. Therefore,

$$2\sum_{0\le j<n}\frac{j}{n}\cdot\pi_{n,j}^{(k)}\cdot|G_j| \le 2\delta\cdot\sum_{0\le j<n}\frac{j^2}{n}\cdot\pi_{n,j}^{(k)} = \left(\delta\cdot\mathcal{F}(n,k)+\frac{\delta}{2}\right)\cdot n$$

when n is large enough. Thus

$$F_n = n + 1 + \mathsf{S}(k) + \mathsf{X}(n,k) \cdot \xi + E_1(n) \cdot n + E_2(n) \cdot \mathcal{F}(n,k) \cdot n, \tag{5.1}$$

where we have the bounds $(1+\xi/4)-\delta/2 \leq E_1(n) \leq (1+\xi/4)+\delta/2$ and $(2+\xi/2)-\delta \leq E_2(n) \leq (2+\xi/2)+\delta$. Notice that the bounds for $E_1(n)$ and $E_2(n)$ do not depend on k. Moreover, $\mathsf{S}(k) \sim 2\beta\, k$ for some constant β, where β depends on the chosen selection algorithm. Then it is easy to see that, for large n, k^* belongs to the interval $[k_1, k_2]$, where k_1 minimizes the function of k

$$2(1+\delta)\,\beta\, k - \frac{(1+\delta)\cdot\xi}{8k} \cdot n + \left[\left(2+\frac{\xi}{2}\right) - \delta\right] \frac{(1-\delta)}{4k} \cdot n,$$

that we obtain by plugging the asymptotic estimates of $\mathsf{S}(k)$, $\mathsf{X}(n,k)$, and $\mathcal{F}(n,k)$ and the lower bounds on $E_1(n)$ and $E_2(n)$ in (5.1), disregarding those terms not depending on k. Similarly, k_2 minimizes the function of k

$$2(1-\delta)\,\beta\, k - \frac{(1-\delta)\cdot\xi}{8k} \cdot n + \left[\left(2+\frac{\xi}{2}\right) + \delta\right] \frac{(1+\delta)}{4k} \cdot n.$$

This yields

$$k_1 = \sqrt{\frac{((4+\xi)-2\delta)(1-\delta)-(1+\delta)\cdot\xi}{16(1+\delta)\beta}} \cdot \sqrt{n},$$

$$k_2 = \sqrt{\frac{((4+\xi)+2\delta)(1+\delta)-(1-\delta)\cdot\xi}{16(1-\delta)\beta}} \cdot \sqrt{n}.$$

The reasoning above holds for every $\delta > 0$, no matter how small we choose it. Therefore, we can conclude the following theorem.

THEOREM 8. *Let $s^* = 2k^* + 1$ be the optimal sample size w.r.t. the average total cost of quickselect to select a random element when the median of the sample is used as the pivot. Then $k^* = \sqrt{n/4\beta} + o(\sqrt{n}\,)$.*

The procedure that we have just outlined here will be used again in section 6. Observe that it has been enough to consider the main terms of $\mathsf{S}(k)$, $\mathsf{X}(n,k)$, and $\mathcal{F}(n,k)$ to find the main term of k^*. In section 6 we will avoid dealing with the tedious details by disregarding terms of small order and by obtaining minima as if these terms did not exist. We have already shown here that it can be done safely and yields asymptotically valid conclusions.

Finally, notice that $\Theta(\sqrt{n}\,)$-sampling improves the worst-case complexity of quickselect from $\Theta(n^2)$ to $\Theta(n^{3/2})$; the same happens with quicksort [17]. This is valid no matter what the rank of the sought element is and even if the selection of pivots is made using an algorithm with quadratic worst case.

6. Optimal samples for quicksort. As shown in section 4, picking the median of the sample as the pivot is not always the best choice. Therefore, we set $p = \psi \cdot s + o(s)$ for some fixed $0 < \psi \leq 1/2$. We do not necessarily assume $\psi = \psi^*(\xi)$.

The following theorem provides the main term of the total cost of quicksort when we make the size of the sample grow with the size of the input. The proof (which we do not give here) uses an argument quite similar to that of Theorem 5, finding matching lower and upper bounds for the limiting log-entropy of Q_n. (For further details see [16, 20].)

THEOREM 9. *Let $p = \psi \cdot s + o(s)$, where $0 < \psi < 1$ and $s = \omega(1)$ and $s = o(n)$. Then the average total cost of quicksort is $Q_n = q_\xi(\psi) \cdot n \ln n + o(n \log n)$.*

If we measure only the number of comparisons ($\xi = 0$), the theorem above states that *any* sample size $s = \omega(1)$ and $s = o(n)$ with $\psi = 1/2$ is asymptotically optimal w.r.t. the main term of quicksort, as the expected number of comparisons for all of them is $Q_n \sim n \log_2 n$.

To investigate the optimal sample size we will assume $p = \lceil \psi \cdot (s+1) \rceil - 1$, but any other discretization satisfying $\lim_{s\to\infty}(p/s) = \psi$ would yield similar results.

Let us introduce

$$\mathcal{Q}(n, s, \psi) = -\left(H_n - h(\psi)\right) \frac{n-1}{n} + \sum_{0 \le j < n} \left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) \frac{jH_j}{n},$$

where $h(x) = -x \ln x - (1-x)\ln(1-x)$ (see (4.1)). The function $\mathcal{Q}(n, s, \psi)$ plays a role analogous to that of $\mathcal{F}(n, k)$ in the analysis of quickselect. By Theorem 9 we can decompose Q_n as $Q_n = q_\xi(\psi)\, nH_n + R_n$, where $R_n = o(n \log n)$. The recurrence for quicksort (2.2) can be then rewritten as

$$\begin{aligned} Q_n = {} & \mathsf{S}(s, \psi) + n + 1 + \mathsf{X}(n, s, \psi) \cdot \xi + \sum_{0 \le j < n} \left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) \cdot R_j \\ & + \Big(H_n - h(\psi)\Big) \cdot q_\xi(\psi) \cdot (n-1) + \mathcal{Q}(n, s, \psi) \cdot q_\xi(\psi) \cdot n. \end{aligned}$$

Let $\chi_\psi(s)$ be the oscillating factor induced by taking ceilings: $\chi_\psi(s) = \lceil \psi \cdot (s+1) \rceil - \psi \cdot (s+1)$. We have the following lemma for the asymptotic behavior of $\mathcal{Q}(n, s, \psi)$.

LEMMA 10. *If $s = \omega(1)$, $s = o(n)$, and $p = \lceil \psi \cdot (s+1) \rceil - 1$ for some ψ such that $0 < \psi \le 1/2$, then $\mathcal{Q}(n, s, \psi) = 1/2s - (\ln(1-\psi) - \ln\psi) \cdot \chi_\psi(s)/s + \mathcal{O}\left(\max\left\{1/n, 1/s^2\right\}\right)$.*

Proof. Using Proposition 20(b) in Appendix B yields

$$\begin{aligned} \sum_{0 \le j < n} \left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) jH_j = {} & \left(H_n - \mathrm{VE}(s, p)\right) \cdot (n-1) \\ & + \frac{p+1}{s+1}(H_{p+1} - H_{q+1}) + \frac{q+1}{s+1}(H_{q+1} - H_{p+1}) + \frac{1}{p+1} + \frac{1}{q+1} - \frac{2}{s+1} - 1. \end{aligned}$$

The last four terms above are $\mathcal{O}(1)$. On the other hand, from the equalities $p + 1 = \psi \cdot (s+1) + \chi_\psi(s)$, $H_n = \ln n + \gamma + 1/(2n) + \Theta(n^{-2})$, and $\ln(1 + 1/x) = 1/x + \Theta(x^{-2})$, it is not difficult to deduce that

$$\begin{aligned} & -H_{s+1} + \frac{p+1}{s+1} \cdot H_{p+1} + \frac{q+1}{s+1} \cdot H_{q+1} \\ & = -h(\psi) + \frac{1}{2s} - \Big(\ln(1-\psi) - \ln\psi\Big) \frac{\chi_\psi(s)}{s} + \Theta\left(\frac{1}{s^2}\right), \end{aligned}$$

and the lemma follows after a few simple manipulations. ∎

The factor $\ln(1-\psi) - \ln\psi$ in the statement of Lemma 10 is zero only when $\psi = 1/2$. Therefore, the perturbation $\chi_\psi(s)/s$ has to be taken into account whenever we are not selecting the median of the sample as the pivot.

As pointed out in section 2, $\mathsf{S}(s, p)$ is linear with the constant of proportionality typically dependent on the quotient $\psi = p/s$. Thus we assume $\mathsf{S}(s, p) = \mathsf{S}(s, \psi) = \beta \cdot s + o(s)$ for some constant $\beta = \beta(\psi)$. We also need the asymptotic properties of

$\mathsf{X}(n, s, \psi)$ given in the next lemma. It extends Lemma 6 for $\psi < 1/2$. The proof, very similar to others in this work, is omitted.

LEMMA 11. *If $s = \omega(1)$, $s = o(n)$, and $p = \lceil \psi \cdot (s+1) \rceil - 1$ for some ψ such that $0 < \psi \leq 1/2$, then*

$$\mathsf{X}(n, s, \psi) = \psi(1-\psi)n\left(1 - \frac{1}{s}\right) + (1-2\psi)\cdot\chi_\psi(s)\cdot\frac{n}{s} + \mathcal{O}\left(\max\left\{n/s^2, 1\right\}\right).$$

Therefore, by a reasoning identical to that of section 5, we can obtain the leading term of the optimal sample size s^* by minimizing

$$\beta\cdot s + \frac{n}{s}\cdot\left(\frac{q_\xi(\psi)}{2} - \psi\cdot(1-\psi)\cdot\xi\right) + \frac{n}{s}\cdot\Big((1-2\psi)\cdot\xi - (\ln(1-\psi) - \ln\psi)\cdot q_\xi(\psi)\Big)\cdot\chi_\psi(s). \tag{6.1}$$

We will consider two variants of quicksort with sample sizes depending on n. The first one is setting $\psi = \psi^*(\xi)$, i.e., choosing the optimal ratio p/s, and the second one is setting $\psi = 1/2$, that is, always picking the median of the sample (if $\xi \leq \tau$, both variants are identical, as $\psi^* = 1/2$ in this case). Fortunately, in any of these two variants the last term of (6.1) vanishes because of (4.2). Therefore, we can follow similar steps to those in the analysis of quickselect to obtain the two theorems below.

THEOREM 12. *Let s^* denote the optimal sample size w.r.t. Q_n when we use the $\lceil \psi^*(s+1) \rceil$th element of the sample as the pivot. Then*

$$s^* = \sqrt{\left(\frac{q_\xi(\psi^*)}{2} - \psi^*\cdot(1-\psi^*)\cdot\xi\right)\frac{1}{\beta(\psi^*)}}\cdot\sqrt{n} + o\left(\sqrt{n}\right).$$

THEOREM 13. *Let s^* denote the optimal sample size w.r.t. Q_n when the median of the sample is used as the pivot. Then*

$$s^* = \sqrt{\left(\frac{4 - (2\ln 2 - 1)\cdot\xi}{8\ln 2}\right)\frac{1}{\beta(1/2)}}\cdot\sqrt{n} + o\left(\sqrt{n}\right).$$

Notice that Theorem 13 makes no sense when $\xi > \tau$, as the factor multiplying $\sqrt{n}$ would be the square root of a negative number. This provides a further check for the conclusions of section 4 and is consistent with the observation that the optimal samples in that situation have constant size.

In [17] a dynamic-programming algorithm was used to compute several exact values of the optimal sample size for quicksort to minimize the average number of comparisons, assuming that standard quickselect was the median-finding algorithm. The authors reported that their data suggests a best power-law fit proportional to $n^{0.588}$. Also, for a tuned variant of quicksort (see section 7.2) they found that the best fit is proportional to at least $n^{0.475}$.

Figure 7 shows those exact values as a staircase plot. The continuous curve is the leading term of the optimal sample size s^* as stated by Theorem 13, setting $\xi = 0$ (minimize comparisons) and $\beta(1/2) = 2(1 + \ln 2)$ (use standard Hoare's quickselect algorithm). It is in very good accordance with the exact values, even for small values of n. We conjecture that the $o(\sqrt{n})$ term in s^* is actually $\Theta(1)$.

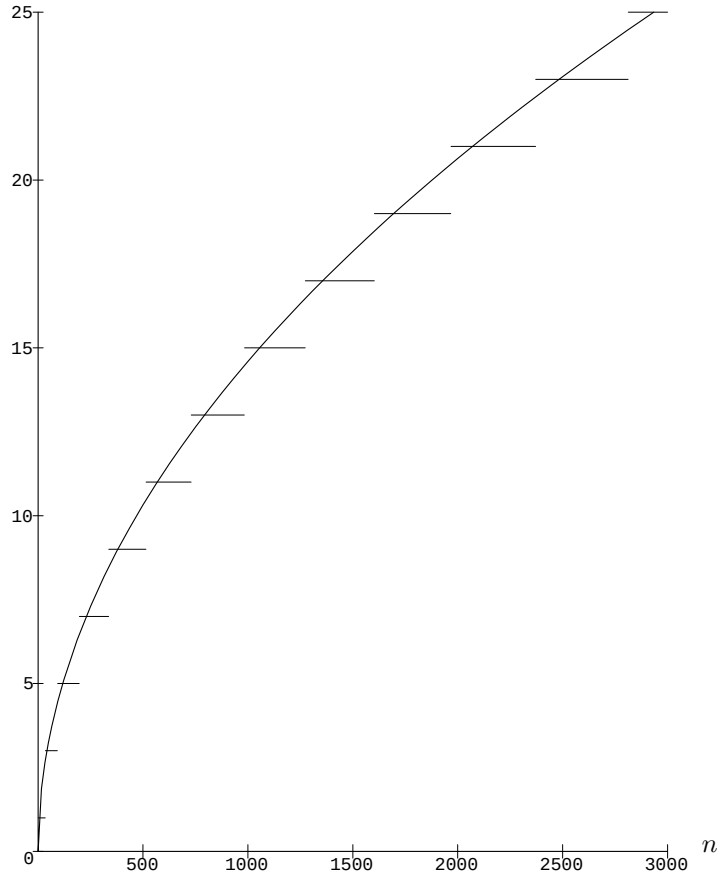


FIG. 7. *Exact values of the optimal samples to minimize the average number of comparisons compared with the main term of s^* (Theorem 13).*

7. Related issues.

7.1. Variance of quickselect. The variance of the number of comparisons of standard quickselect is $\Theta(n^2)$ [14] (see also [9, 18]). Thus, there is a low but nonnegligible probability that the number of comparisons actually made is much larger than the expected number. In this subsection we will study the variance of the number of comparisons of quickselect with sampling, specifically, for the case where the selected pivot is the median of the sample. This analysis is somewhat more complicated than the analysis of its expected performance.

It is worth noting again that we are investigating here the variance of the cost of quickselect when the rank of the sought element is given by a uniformly distributed random variable, not the average of the variances of the costs of selecting each element (see the remark on page 2).

We assume that the median-finding algorithm to select the pivots has quadratic variance. (If the variance were subquadratic, the basic conclusions of this section would not change.) A key observation that makes the analysis relatively simple is that the number of comparisons needed to select the pivot, the number of comparisons made to partition the array, and the number of comparisons made in further

invocations of quickselect in smaller subarrays are mutually independent. If the random variable Z_n is the sum of two independent random variables, X_n and Y_n, and $\{A_j\}_{0\le j<n}$ is a family of independent events, then

$$\begin{aligned}\mathbb{E}\big[Z_n^2\big] &= \mathbb{E}\big[(X_n+Y_n)^2\big] = \mathbb{E}\big[X_n^2\big] + 2\,\mathbb{E}[X_n]\,\mathbb{E}[Y_n] + \mathbb{E}\big[Y_n^2\big] \\ &= \mathbb{E}\big[X_n^2\big] + 2\,\mathbb{E}[X_n]\,(\mathbb{E}[Z_n] - \mathbb{E}[X_n]) + \mathbb{E}\big[Y_n^2\big] \\ &= 2\,\mathbb{E}[X_n]\,\mathbb{E}[Z_n] + \mathbb{E}\big[X_n^2\big] - 2\,\mathbb{E}[X_n]^2 + \sum_{0\le j<n} \Pr\{A_j\}\,\mathbb{E}\big[Y_n^2 \mid A_j\big]\,.\end{aligned}$$

Applying the equation above with

X_n = number of comparisons made to select the pivot and partition the array,
Y_n = number of comparisons made in further recursive invocations,

and A_j = "the pivot is the $(j+1)$th", we can easily obtain recurrences for $F_n^{(2)}$, the second moment of the number of comparisons made by quickselect to find an element of random rank out of n. We first consider the situation where $s = 2k+1 = \Theta(1)$. Then the recurrence for $F_n^{(2)}$ is

$$\begin{aligned}F_n^{(2)} &= 2(n+\Theta(1))F_n - (n+\Theta(1))^2 + \sum_{0\le j<n} w_{n,j}^{(k)} F_j^{(2)} \\ &= \frac{3k+5}{k+1}\,n^2 + \mathcal{O}(n) + \sum_{0\le j<n} w_{n,j}^{(k)} F_j^{(2)}.\end{aligned}$$

The weights and shape function are the same as for the analysis of the expected cost of quickselect with fixed-size samples, but the toll function is now quadratic. Therefore, the limiting const-entropy is

$$\mathcal{H}(k) = 1 - \frac{k+3}{2(2k+3)} = \frac{3(k+1)}{2(2k+3)},$$

and $F_n^{(2)} = (3k+5)(4k+6)n^2/(3(k+1)^2) + o(n^2)$. Finally, subtracting F_n^2 from $F_n^{(2)}$ we get the variance of quickselect.

THEOREM 14. *The variance of the number of comparisons made by quickselect when using samples of fixed size $s = 2k+1$ is*

$$V_n = \frac{2k+3}{3(k+1)^2}\,n^2 + o(n^2) = \left(\frac{2}{3k} + \mathcal{O}\left(\frac{1}{k^2}\right)\right) n^2 + o(n^2).$$

As expected, sampling not only reduces the expected number of comparisons, it also reduces its variance: if $v(k)$ denotes the coefficient of n^2 in V_n, we have $v(0) = 1, v(1) = 5/12, v(2) = 7/27\ldots$, and $\lim_{k\to\infty} v(k) = 0$. This suggests that for samples of increasing size ($k = \omega(1)$, $k = o(n)$) the variance is $o(n^2)$. Indeed, this is what actually happens.

The analysis of this case goes along the same lines as the analysis of the expected value F_n in section 5. Similar techniques to those used in the proof of Theorem 5 yield here that the limiting entropy associated to $F_n^{(2)}$ is $\mathcal{H} = 3/4$. Since the toll function is $3n^2 + o(n^2)$, we have $F_n^{(2)} = 4n^2 + o(n^2)$, and the variance is $V_n = F_n^{(2)} - F_n^2 = 4n^2 + o(n^2) - (2n + o(n))^2 = o(n^2)$.

A refinement of this calculation is possible since we know that, because of (5.1), $F_n = 2n + n/(2k) + \Theta(k) + \Theta\left(n/k^2\right)$. Substituting this estimate into the recurrence for $F_n^{(2)}$ and using $F_j^{(2)} \sim 4j^2$ to compute $\sum_{0\le j<n} w_{n,j}^{(k)} F_j^{(2)}$, we get $F_n^{(2)} = 4n^2 + \Theta\left(\max\left\{nk, n^2/k\right\}\right)$. We thus have the following theorem.

THEOREM 15. *The variance V_n of the number of comparisons made by quickselect when the samples are of size $2k+1$, where $k = \omega(1)$ and $k = o(n)$, is $V_n = \mathcal{O}\left(\max\left\{n^2/k, nk\right\}\right)$.*

Notice that the upper bound given above is $o(n^2)$ for any k such that $k = \omega(1)$ and $k = o(n)$. When $k = \Theta(\sqrt{n}\,)$ the variance is $\mathcal{O}(n^{3/2})$ and the standard deviation is $\mathcal{O}(n^{3/4})$. This is the right choice, since it simultaneously minimizes the average number of comparisons (Theorem 8) and the order of magnitude of the variance.

Concerning quicksort, we have not been able to analyze the effect of sampling in the variance of the number of comparisons. We conjecture that for $s = \omega(1)$ and $s = o(n)$ the variance of the number of comparisons satisfies $V_n = \mathcal{O}\left(\max\left\{n^2/s, ns\right\}\right)$, as in Theorem 15.

7.2. Tuning performance. We might avoid redundant comparisons and exchanges as follows [17]: if the input subarray is $A[l\,..\,u]$, we would take the segments $A[l\,..\,l+p]$ and $A[u-q+1\,..\,u]$ as our sample of $s = s(u-l+1)$ elements and apply a slightly modified version of quickselect, which would bring the $(p+1)$th element of the sample to $A[l]$, and put all smaller elements in $A[l+1\,..\,l+p]$ and all larger elements in $A[u-q+1\,..\,u]$. Then the partition of $A[l\,..\,u]$ around the pivot at $A[l]$ would initially set the left scanning pointer to $l+p+1$ and the right scanning pointer to $u-q$. The number of comparisons would then be reduced to $n+2-s$.

This selection-plus-partition combination is more efficient than the standard mechanism assumed in previous sections, but it does not preserve randomness, because the selection of the pivot reorganizes the elements of the sample. There seems not to exist—for general s—an efficient way to perform selection in-place and partition without redundant comparisons and exchanges while preserving randomness. However, we disregard the small amount of sortedness that the selection process introduces and assume that (2.2) and (2.3) are still valid, yielding at least good approximate results.

The steps and computations are absolutely analogous to those of previous sections. The results are, qualitatively speaking, identical. For instance, a straightforward computation—very similar but even simpler than that in the proof of Lemma 1 (see Appendix B)—yields

$$\mathsf{X}(n,s,p) = \frac{(p+1)(q+1)}{(s+1)(s+2)}(n-1-s) \qquad \text{if } s < n.$$

We present here a couple of results for these tuned variants of quickselect and quicksort.

THEOREM 16. *Let $s^* = 2k^*+1$ be the optimal sample size w.r.t. the average total cost of the tuned variant of quickselect to select a random element when the median of the sample is used as pivot. Then $k^* = \sqrt{n/(4\beta - 4 - \xi)} + o\left(\sqrt{n}\,\right)$.*

THEOREM 17. *Let s^* be the optimal sample size w.r.t. the average total cost of the tuned variant of quicksort when the $\lceil \psi^*(s+1)\rceil$th element of the sample is used as the pivot. Then*

$$s^* = \sqrt{\frac{q_\xi(\psi^*)/2 - \psi^*(1-\psi^*)\cdot\xi}{\beta(\psi^*) - 1 - \psi^*(1-\psi^*)\cdot\xi}} \cdot \sqrt{n} + o\left(\sqrt{n}\,\right).$$

Appendix A. The CMT. We briefly review in this appendix the CMT; since no proofs are given here, we refer the reader to [20, 21] for a detailed treatment.

Consider a recurrence like

$$F_n = t_n + \sum_{0 \le j < n} w_{n,j} F_j, \qquad n \ge n_0, \tag{A.1}$$

where t_n is the so-called *toll function* and the quantities $w_{n,j} \ge 0$ are called *weights*. This recurrence is a *divide-and-conquer recurrence* if and only if the following conditions hold:

1. There exists the limiting value $W = \lim_{n\to\infty} \sum_{0\le j<n} w_{n,j}$.
2. The sequence $W_n = \sum_{0\le j<n} w_{n,j}$ converges to W fast enough, namely,

$$|W_n - W| = \mathcal{O}(n^{-c})$$

for some constant $c > 0$.

3. The sequence

$$Z_n = \frac{1}{W_n} \cdot \sum_{0 \le j < n} w_{n,j} \cdot \frac{j}{n}$$

is strictly bounded above by 1 for sufficiently large n.

While the weights $w_{n,j}$ represent the (average) number of recursive calls made to subproblems of size j when the given instance is of size n, W_n is the (average) number of recursive calls made for instances of size n and Z_n is the average size (as a fraction of n) of the subproblems.

THEOREM 18 (CMT). *Let F_n be the solution of* (A.1)*, where $t_n \sim K n^a \log^b n$ for some constants K, $a \ge 0$, and $b > -1$, and let $w(z)$ be a real function over $[0,1]$ such that*

$$\sum_{0 \le j < n} \left| w_{n,j} - \int_{j/n}^{(j+1)/n} w(z)\, dz \right| = \mathcal{O}(n^{-d})$$

for some constant $d > 0$. Let $\phi(x) = \int_0^1 z^x\, w(z)\, dz$ and define $\mathcal{H} = 1 - \phi(a)$. Then

1. *if $\mathcal{H} > 0$, then $F_n \sim t_n \,/\, \mathcal{H}$;*
2. *if $\mathcal{H} = 0$, then $F_n \sim t_n \ln n \,/\, \mathcal{H}'$, where $\mathcal{H}' = -(b+1) \int_0^1 z^a \ln z\, w(z)\, dz$;*
3. *if $\mathcal{H} < 0$, then $F_n = \Theta(n^\alpha)$, where α is the unique real solution of $\phi(x) = 1$.*

The proof of this theorem is based upon more basic and technical results which are also useful to solve or show particular properties of divide-and-conquer recurrences which do not satisfy the hypothesis of the CMT, e.g., for recurrences for which a shape function $w(z)$ cannot be found.

We next consider an example of the results mentioned above. These results have already been used in sections 5 and 6.

Given a function $\gamma_n > 0$ such that $\gamma_n = o(n^\sigma)$ and $\gamma_n = \omega(n^{-\sigma})$ for any $\sigma > 0$ and a toll function t_n, γ_n is a *bounding function* of t_n if and only if there exist a constant $n_\gamma \ge 1$ and a strictly positive nonincreasing function $\beta(z)$ in $[0,1]$ such that $t_n\gamma_n - t_j\gamma_j \ge \beta(j/n) \cdot t_n$ for all $n > n_\gamma$ and for all $n_\gamma \le j < n$.

Given a divide-and-conquer recurrence (A.1) for F_n and a bounding function γ_n of the toll function t_n, the *entropy* $\mathcal{H}_n^{(\gamma)}$ of F_n with respect to γ_n (or γ-entropy of F_n, for short) is

$$\mathcal{H}_n^{(\gamma)} = \gamma_n - \sum_{n_\gamma \le j < n} w_{n,j} \cdot \frac{t_j}{t_n} \cdot \gamma_j.$$

Also, we define the *limiting* γ*-entropy* as $\mathcal{H}^{(\gamma)} = \lim_{n\to\infty} \mathcal{H}_n^{(\gamma)}$ if it exists. Then it is possible to prove that

$$F_n \sim t_n \gamma_n \,/\, \mathcal{H}^{(\gamma)}\,. \tag{A.2}$$

The CMT provides an easy way to determine a bounding function and to compute the limiting value of the corresponding entropy under appropriate circumstances. Notice that $\mathcal{H} \equiv \mathcal{H}^{(1)}$ is the limiting const-entropy and $\mathcal{H}' \equiv \mathcal{H}^{(\ln)}$ is the limiting log-entropy.

Appendix B. Combinatorial identities and proofs. In this appendix we collect several standard results for Beta integrals (see, for instance, [1]) and their discrete analogues, which easily follow from basic results in discrete calculus (see, for instance, [5]). We therefore leave the first two propositions without proof.

We also prove Lemma 1 in this appendix.

PROPOSITION 19.

$$\text{(a)} \qquad \int_0^1 z^\alpha (1-z)^\beta \, dz = \frac{\alpha!\beta!}{(\alpha+\beta+1)!},$$

$$\text{(b)} \qquad -\int_0^1 z^\alpha (1-z)^\beta \, \ln z \, dz = \frac{\alpha!\beta!}{(\alpha+\beta+1)!}(H_{\alpha+\beta+1} - H_\alpha).$$

For any real x and integer $\ell \ge 0$, $x^{\underline{\ell}} = x \cdot (x-1) \cdots (x-\ell+1)$ will denote the ℓth *falling factorial* of x [5]. Many of the computations in this work involve evaluating sums of the type

$$\sum_{0\le j<n} g(j,n) \binom{j}{p} \binom{n-1-j}{q} = \frac{1}{p!\,q!} \sum_{0\le j<n} g(j,n)\, j^{\underline{p}}\, (n-1-j)^{\underline{q}}$$

for some appropriate function $g(j,n)$. For instance, to evaluate $\mathsf{X}(n,s,p)$ we use $g = j(n-1-j)$, for $\mathcal{F}(n,k)$ we use $g = j^2$, and for $\mathcal{Q}(n,\psi)$ we use $g = jH_j$.

Proposition 20 is helpful in coping with these sums. Notice the parallelism with Proposition 19.

PROPOSITION 20.

$$\text{(a)} \quad A_{\ell,m}(n) = \sum_{0\le j<n} j^{\underline{\ell}} (n-1-j)^{\underline{m}} = \ell!\, m! \binom{n}{\ell+m+1},$$

$$\text{(b)} \quad B_{\ell,m}(n) = \sum_{0\le j<n} j^{\underline{\ell}} (n-1-j)^{\underline{m}} H_j = \ell!\, m! \binom{n}{\ell+m+1} (H_n - H_{\ell+m+1} + H_\ell)\,.$$

The following intermediate result, which is used in section 5, is just an example of the usefulness of the proposition above. Similar intermediate results in the analysis of quicksort (section 6) follow also from Proposition 20(b).

PROPOSITION 21. *For all s and p such that $0 \le p < s$,*

$$(s+1)(s+2)\sum_{0\le j<n} j^2\left(\pi_{n,j}^{(s,p)} + \pi_{n,n-1-j}^{(s,p)}\right) = \big((p+1)(p+2)+(q+1)(q+2)\big)n^2 - 6(p+1)(q+1)n + (p-q)^2 + (s+1).$$

Proof. Apply Proposition 20(a) twice, using $j^2 = (j-x)(j-1-x)+(2x+1)(j-x)+x^2$, with $x=p$ the first time and with $x=q$ the second time. □

We end this appendix with a proof of Lemma 1, which gives a closed expression for $\mathsf{X}(n,s,p)$.

Proof of Lemma 1. Here we assume that the subarray to be partitioned contains the pivot in its first position. If the pivot ends at $A[j+1]$ after partitioning and there were t elements in $A[2\,..\,j+1]$ greater than the pivot, then exactly t swaps were needed to arrange the array. (We do not count the one or two additional swaps just before and after the partitioning.) The probability of this to happen is

$$\frac{\binom{j}{j-t}\binom{n-1-j}{t}}{\binom{n-1}{j}},$$

by an argument largely analogous to the one used to obtain the value of $\pi_{n,j}^{(s,p)}$ in (2.1). Thus

$$\mathsf{X}(n,s,p) = \sum_{0\le j<n} \pi_{n,j}^{(s,p)} \sum_t t\cdot \frac{\binom{j}{j-t}\binom{n-1-j}{t}}{\binom{n-1}{j}}.$$

We can now use the "derivative" of Vandermonde's convolution [5],

$$\sum_t t\binom{b}{c-t}\binom{a}{t} = a\binom{a+b-1}{c-1},$$

to get

$$\mathsf{X}(n,s,p) = \sum_{0\le j<n} \pi_{n,j}^{(s,p)} \cdot \frac{j(n-1-j)}{(n-1)}.$$

Finally, using the equality $j(n-1-j) = (j-p)(n-1-j-q)+(j-p)(q-p)+(pn-p^2-p)$ yields

$$\mathsf{X}(n,s,p) = \frac{A_{p+1,q+1}(n) + (q-p)\cdot A_{p+1,q}(n) + (pn-p^2-p)\cdot A_{p,q}(n)}{p!\,q!\,(n-1)\binom{n}{s}},$$

where $P_{p,q}(n)$ is as in Proposition 20(a). The rest of the proof is a simple matter of computation. □

Acknowledgments. We thank H.-K. Hwang for various useful comments and suggestions, especially for encouraging us to study the variance of quicksort and quickselect. We also wish to thank the anonymous referees of this paper for their comments and useful suggestions.

REFERENCES

[1] M. Abramowitz and I. Stegun, eds., *Handbook of Mathematical Functions*, Dover, New York, 1964.

[2] J. Bentley, *Programming Pearls*, Addison-Wesley, Reading, MA, 1986.

[3] J. Bentley and M. McIlroy, *Engineering a sort function*, Software—Practice and Experience, 23 (1993), pp. 1249–1265.

[4] R. Floyd and R. Rivest, *Expected time bounds for selection*, Comm. ACM, 18 (1975), pp. 165–173.

[5] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics*, 2nd ed., Addison-Wesley, Reading, MA, 1994.

[6] R. Grübel, *On the median-of-k version of Hoare's selection algorithm*, Theor. Inform. Appl., 33 (1999), pp. 177–192.

[7] C. Hoare, *Algorithm* 65: Find, Commu. ACM, 4 (1961), pp. 321–322.

[8] C. Hoare, *Quicksort*, Computer Journal, 5 (1962), pp. 10–15.

[9] P. Kirschenhofer and H. Prodinger, *Comparisons in Hoare's Find algorithm*, Combin. Probab. Comput., 7 (1998), pp. 111–120.

[10] P. Kirschenhofer, H. Prodinger, and C. Martínez, *Analysis of Hoare's* Find *algorithm with median-of-three partition*, Random Structures Algorithms, 10 (1997), pp. 143–156.

[11] D. Knuth, *Mathematical analysis of algorithms*, in Information Processing '71, North-Holland, Amsterdam, 1972, pp. 19–27.

[12] D. Knuth, *The Art of Computer Programming: Sorting and Searching*, Vol. 3, 2nd ed., Addison-Wesley, Reading, MA, 1998.

[13] H. Mahmoud, *Sorting: A Distribution Theory*, John Wiley and Sons, New York, 2000.

[14] H. M. Mahmoud, R. Modarres, and R. T. Smythe, *Analysis of quickselect: An algorithm for order statistics*, Theor. Inform. Appl., 29 (1995), pp. 255–276.

[15] C. Martínez and S. Roura, *Optimal sampling strategies in quicksort*, in Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP'98), K. Larsen, S. Skyum, and G. Winskel, eds., Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 327–338.

[16] C. Martínez and S. Roura, *Optimal Sampling Strategies in Quicksort and Quickselect*, Tech. Rep. LSI-98-1-R, LSI-UPC, 1998; also available online from www.lsi.upc.es/dept/techreps/1998.html.

[17] C. McGeoch and J. Tygar, *Optimal sampling strategies for quicksort*, Random Structures Algorithms, 7 (1995), pp. 287–300.

[18] A. Panholzer and H. Prodinger, *A generating functions approach for the analysis of grand averages for multiple QUICKSELECT*, Random Structures Algorithms, 13 (1998), pp. 189–209.

[19] H. Prodinger, *Multiple Quickselect—Hoare's Find algorithm for several elements*, Inform. Process. Lett., 56 (1995), pp. 123–129.

[20] S. Roura, *Divide-and-Conquer Algorithms and Data Structures*, Ph.D. thesis, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain, 1997.

[21] S. Roura, *Improved master theorems for divide-and-conquer recurrences*, J. ACM, 48 (2001), pp. 170–205.

[22] R. Sedgewick, *The analysis of quicksort programs*, Acta Inform., 7 (1976), pp. 327–355.

[23] R. Sedgewick, *Implementing quicksort programs*, Comm. ACM, 21 (1978), pp. 847–856.

[24] R. Sedgewick, *Quicksort*, Garland, New York, 1978.

[25] R. Singleton, *Algorithm* 347*: An efficient algorithm for sorting with minimal storage*, Comm. ACM, 12 (1969), pp. 185–187.

[26] M. van Emden, *Increasing the efficiency of quicksort*, Comm. ACM, 13 (1970), pp. 563–567.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 706–721

THE COMPACTNESS OF INTERVAL ROUTING FOR ALMOST ALL GRAPHS*

CYRIL GAVOILLE† AND DAVID PELEG‡

Abstract. Interval routing is a compact way of representing routing tables on a graph. It is based on grouping together, in each node, destination addresses that use the same outgoing edge in the routing table. Such groups of addresses are represented by some intervals of consecutive integers. We show that almost all the graphs, i.e., a fraction of at least $1 - 1/n^2$ of all the n-node graphs, support a shortest path interval routing with *three* intervals per outgoing edge, even if the addresses of the nodes are arbitrarily fixed in advance and cannot be chosen by the designer of the routing scheme. In case the addresses are initialized randomly, we show that *two* intervals per outgoing edge suffice, and, conversely, that two intervals are required for almost all graphs. Finally, if the node addresses can be chosen as desired, we show how to design in polynomial time a shortest path interval routing with a *single* interval per outgoing edge for all but at most $O(\log^3 n)$ outgoing edges in each node. It follows that almost all graphs support a shortest path routing scheme which requires at most $n + O(\log^4 n)$ bits of routing information per node, improving on the previous upper bound.

Key words. interval routing, compact routing, random graphs

AMS subject classifications. 05C85, 69Q10, 68R10, 68Q25

PII. S0097539799351717

1. Introduction.

1.1. Background. A *universal routing strategy* is an algorithm which generates a routing scheme for every given network. One type of trivial universal routing strategy is based on schemes that keep in each node a full routing table which specifies an output port for every destination. Though this strategy can guarantee routing along shortest paths, each router has to locally store $\Theta(n \log d)$ bits of information, where d is the degree of the router (i.e., the number of output ports) and n is the number of nodes in the network.

The *interval routing scheme* [9, 10] is a compact routing scheme, i.e., a routing scheme that needs to keep only a small amount of information in each node to route messages correctly through the network. The idea of this scheme is to label the n nodes of the network with unique integers from $\{1, \ldots, n\}$ and to label the outgoing arcs in every node with a set of intervals forming a partition of the name range. The routing process sends a message on the unique outgoing arc labeled by an interval that contains the destination label. While the preprocessing stage of such a routing scheme (which is performed once in the initialization of the network) might be complex, the delivery protocol consists of simple decision functions which can be implemented with $O(kd \log n)$ bits in each node of degree d, where k is the maximum number of intervals assigned to an arc. Such a routing scheme supports a compact implementation whenever k is small in comparison with n or d.

*Received by the editors February 11, 1999; accepted for publication (in revised form) May 17, 2001; published electronically October 23, 2001. A preliminary version of this paper appears in *Proceedings of the 12th International Symposium on Distributed Computing*, Andros, Greece, 1998.
http://www.siam.org/journals/sicomp/31-3/35171.html

†LaBRI, Université Bordeaux I, 351, cours de la Libération, 33405 Talence Cedex, France (gavoille@labri.fr). This author was supported by the French-Israeli cooperation "AFIRST."

‡Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel (peleg@wisdom.weizmann.ac.il). This author was supported in part by grants from the Israel Science Foundation and from the Israel Ministry of Science and Art.

In [8], it is shown that there is no universal routing strategy that can guarantee a shortest path routing scheme with less than $\Omega(n \log d)$ bits per node for all the n-node networks of maximum degree d. This result means that there is some worst-case network where for any shortest path routing function, the number of bits required to be stored in a router is not significantly smaller than the size of a routing table, whatever the node labeling (from the range $\{1, \ldots, n\}$) and the shortest paths are. Fortunately, such a problematic situation where the routing tables cannot be compressed occurs for a limited number of worst-case networks only.

In particular, in [3], it is shown that for almost all the n-node networks the size of the routing tables can be reduced to $O(n)$ bits per node. More precisely, it is shown that all labeled graphs but a $1/n^3$ fraction can be routed with a scheme that uses $3n + o(n)$ bits of information, under the assumption that nodes are randomly labeled in the range $\{1, \ldots, n\}$, and that every node knows its neighbors for "free," or that the port assignment may be changed. Moreover, if, during the initialization process of the network, nodes can be relabeled with binary string of length $c \log^2 n + o(\log^2 n)$ bits[1] (for constant $c > 3$), then $c \log^2 n$ bits per node suffice to route along the shortest paths for almost all networks.

1.2. Definitions and results. In this paper, we consider shortest path routing schemes only. An undirected graph $G = (V, E)$ represents the classic model of the underlying topology of the network. An n-node graph G with the nodes labeled by labels from the set $\{1, \ldots, n\}$ is said to *support a k-interval routing scheme* (k-IRS for short) if there exists an interval routing scheme $\mathcal{R}$ for G with the property that for every (directed) edge e, the set of node labels to which $\mathcal{R}$ routes messages via e is composed of at most k intervals. (An interval means a set of consecutive integers taken from $\{1, \ldots, n\}$, where n and 1 are considered to be consecutive.)

Our goal is to find a labeling of the nodes and a shortest path system in order to minimize the maximum number of intervals assigned to the edges of the graph. We distinguish three models depending on the freedom we have in labeling the nodes.

1. *Adversary.* Labels are fixed in advance (by an adversary) and cannot be permuted.
2. *Random.* Labels are randomly permuted.
3. *Designer.* Labels can be chosen (by the routing designer) in order to achieve the smallest possible number of intervals.

In all three models, the routing designer has the freedom of selecting the shortest paths to be used.

Corresponding to these three models, we introduce the following three parameters. We denote by $\mathrm{IRS}_A(G)$ the smallest integer k such that G supports a k-IRS in the adversary model (namely, for every arbitrary labeling of the nodes). We denote by $\mathrm{IRS}_R(G)$ the smallest k such that G supports a k-IRS in the random model (namely, given a random labeling of the nodes of G) with high probability. Finally, we denote by $\mathrm{IRS}(G)$ the smallest k such that G supports a k-IRS in the designer model (namely, under some specifically chosen node labeling of G). Clearly, $\mathrm{IRS}(G) \leqslant \mathrm{IRS}_R(G) \leqslant \mathrm{IRS}_A(G)$ for every graph G.

The parameter $\mathrm{IRS}(G)$, sometimes called the *compactness* of the scheme, has been computed for many classes of graphs (see [6] for a recent overview). Notably, in [7] it is shown that for every G, $\mathrm{IRS}_R(G) < n/4 + o(n)$, whereas there exists some worst-case G_0 such that $\mathrm{IRS}(G_0) > n/4 - o(n)$. However, as shown in this paper, the

[1] Where hereafter log denotes the logarithm in base 2.

situation is considerably better for the "average" case. Specifically, we will see that $\mathrm{IRS}(G) \leqslant 2$ for a fraction of at least $1 - 1/n^2$ of all the n-node labeled graphs.

Technically, we use random graphs instead of the Kolmogorov random graphs used in [3]. A discussion about the relationships between random and Kolmogorov random graphs can be found in [4]. The class $\mathcal{G}_{n,p}$ denotes the classic model of n-node labeled random graphs, where $0 \leqslant p \leqslant 1$ represents the probability of having an edge between any two nodes. Clearly, a random graph $G \in \mathcal{G}_{n,1/2}$ has a given property $\mathcal{P}$ with probability α if and only if $\mathcal{P}$ holds for a fraction of α of all the n-node labeled graphs. Interval routing on random graphs has been first investigated in [5], where some lower bounds are given for $\mathrm{IRS}(G)$ for $G \in \mathcal{G}_{n,p}$. More precisely, it is shown therein that for $p = n^{-1+1/s}$ for integer $s > 0$, such that there exists some $\varepsilon > 0$ satisfying $(\ln^{1+\varepsilon} n)/n < p < n^{-1/2-\varepsilon}$, a graph $G \in \mathcal{G}_{n,p}$ satisfies

$$\mathrm{IRS}(G) \;\geqslant\; \frac{1}{10}\, n^{1-6/\ln(np)-\ln(np)/\ln n} \tag{1.1}$$

with high probability. It is also shown that for some $p = n^{-1+1/\Theta(\sqrt{\log n})}$, a graph $G \in \mathcal{G}_{n,p}$ satisfies $\mathrm{IRS}(G) = \Omega(n^{1-1/\Theta(\sqrt{\log n})})$ with high probability. In this paper, we investigate the case where p is a fixed constant, e.g., $p = 1/2$, in order to establish some average results on the total space of n-node graphs. (Note that for constant p, (1.1) cannot be used since in this case p lies outside the validity range.)

The following table presents our results for each model. The results of the table are proved for a fraction of at least $1 - 1/n^2$ of all the n-node labeled graphs.

Label select	Designer	Random	Adversary
Upper bound	$\mathrm{IRS} \leqslant 2$	$\mathrm{IRS}_R \leqslant 2$	$\mathrm{IRS}_A \leqslant 3$
Lower bound	$\mathrm{IRS} \geqslant 1$	$\mathrm{IRS}_R \geqslant 2$	$\mathrm{IRS}_A \geqslant 3$

At this time, we are still unable to decide whether $\mathrm{IRS}(G) = 1$ or 2 for almost every graph G in the model where both the node labels and the shortest path system can be chosen in advance by the designer. However, we present a polynomial time algorithm to design a 2-IRS for all graphs but a $1/n$ fraction such that for every node, all its outgoing edges are labeled with a single interval, except for up to $O(\log^3 n)$ edges where two intervals are required. It follows that almost every graph supports a shortest path routing scheme that can be implemented with $n + O(\log^4 n)$ bits, improving on the best known result (cf. [3]). Note that our result is stated with the assumption that nodes can be permuted but without the assumption that nodes know their neighbors.

2. Randomly assigned node labels. In this section, we show that in the random model, almost every graph G satisfies $\mathrm{IRS}_R(G) = 2$. This implies, in particular, that almost every graph G satisfies $\mathrm{IRS}(G) \leqslant 2$. This is done by showing that, with probability at least $1 - 1/n^2$, a random graph G from $\mathcal{G}_{n,1/2}$ satisfies $\mathrm{IRS}_R(G) = 2$. Actually, we show that the result holds for the class $\mathcal{G}_{n,p}$ of random graphs for each fixed probability $0.45 < p < 1$.

2.1. Upper bound. In this subsection, we shall prove that $\mathrm{IRS}_R(G) \leqslant 2$. Assume the node labels $\{1, \ldots, n\}$ are assigned randomly for the graph G. In that case, given that G is a random graph in $\mathcal{G}_{n,p}$, we may assume that the nodes are first marked by the labels 1 through n, and only then we draw the edges randomly and uniformly with probability p.

For random graphs selected from $\mathcal{G}_{n,p}$, we have the following simple bounds.[2] We denote by $\Gamma(v)$ the set composed of v and of its neighbors.

LEMMA 2.1. *With probability at least* $1 - 1/n^3$, *and for every fixed* $0 < p < 1$, *a random graph* $G \in \mathcal{G}_{n,p}$ *is of diameter* 2, *and for every node* $v \in V$,

$$np - 3\sqrt{n \ln n} \leqslant |\Gamma(v)| \leqslant np + 3\sqrt{n \ln n}.$$

Let $\mathcal{E}_A$ denote the event that the random graph at hand does not satisfy the properties asserted in Lemma 2.1. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_A}$.

For notational convenience, we identify nodes with their labels, i.e., denote $V = \{1, \ldots, n\}$.

Consider a node $v_0 \in V$. We need to argue that with high probability, the edges of v_0 can be labeled with at most two intervals per edge so that for every possible destination $v_d \in V$, the selected edge is along a shortest path from v_0 to v_d.

Let $A = \Gamma(v_0) \setminus \{v_0\}$ and $B = V \setminus \Gamma(v_0)$. Since G satisfies the event $\overline{\mathcal{E}_A}$,

$$np - 3\sqrt{n \ln n} - 1 \leqslant |A| \leqslant np + 3\sqrt{n \ln n}, \tag{2.1}$$

$$n(1-p) - 3\sqrt{n \ln n} \leqslant |B| \leqslant n(1-p) + 3\sqrt{n \ln n}. \tag{2.2}$$

Let

$$C = \{v \in B \mid v + 1 \in A \text{ and } (v, v+1) \in E\}.$$

LEMMA 2.2. *With probability at least* $1 - 1/n^3$, *the size of the set* C *is bounded by*

$$n(1-p)p^2 - 5\sqrt{n \ln n} \leqslant |C| \leqslant n(1-p)p^2 - 5\sqrt{n \ln n}.$$

Proof. Consider a vertex $v \in B$, and let I_v denote the event that $v \in C$. This event happens precisely if $v + 1 \in A$ and $(v, v+1) \in E$. These two subevents are independent and both occur with probability p, and hence $\mathbb{P}(I_v) = p^2$. Also note that the events I_v for $v \in B$ are mutually independent. Let Z be a random variable denoting the size of $|C|$. Then $Z = \sum_{v \in B} z_v$, where z_v is the characteristic random variable of the event I_v. Hence, Z is the sum of $|B|$ mutually independent Bernoulli variables, and its expected value is $\mathbb{E}(Z) = |B|p^2$, and hence applying Chernoff's bound (cf. [1]) we get

$$\begin{aligned}\mathbb{P}\left(Z \geqslant n(1-p)p^2 + 5\sqrt{n \ln n}\right) &\leqslant \mathbb{P}\left(Z \geqslant \mathbb{E}(Z) + 2\sqrt{n \ln n}\right) \\ &\leqslant \exp\left(-\frac{\left(2\sqrt{n \ln n}\right)^2}{n}\right) \leqslant \frac{1}{n^4}\end{aligned}$$

and

$$\mathbb{P}\left(Z \leqslant n(1-p)p^2 - 5\sqrt{n \ln n}\right) \leqslant \frac{1}{n^4},$$

and the lemma follows. ∎

[2] We state a variant of the bounds suitable to our needs and make no attempt to optimize them; see [2] for sharper statements.

Let $\mathcal{E}_B$ denote the event that the random graph at hand does not satisfy the property asserted in Lemma 2.2 for *some* node v_0. Note that the probability for this event is bounded above by $1/n^3$. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_B}$.

Let us now define one interval per emanating edge of v_0 to take care of routing to the nodes in $A \cup C$. For every node $w \in A$, mark the edge (v_0, w) by the interval $[w-1, w]$ if $w - 1 \in C$ and by the interval $[w]$ if $w - 1 \notin C$.

It is thus left to show how the remaining interval per edge of v_0 can be used to route optimally towards the nodes in $X = B \setminus C$. This is done as follows. Let $X = \{x_1, \ldots, x_m\}$. Note that since G satisfies the events $\overline{\mathcal{E}_A}$ and $\overline{\mathcal{E}_B}$,

$$n(1-p)(1-p^2) - 8\sqrt{n \ln n} \;\leqslant\; m \;\leqslant\; n(1-p)(1-p^2) + 8\sqrt{n \ln n}.$$

We now describe a process for selecting a subset of A, denoted $Y = \{y_1, \ldots, y_m\} \subseteq A$, such that there is an edge $(x_i, y_i) \in E$ for every $1 \leqslant i \leqslant m$. Once this is done, we mark each edge (v_0, y_i) by the interval $[x_i]$, thus completing our task.

The selection process is a straightforward greedy one. Let $Q = A$. Having already selected $y_1, \ldots, y_{i-1}$, the ith step consists of selecting y_i to be some arbitrary neighbor of x_i in Q and discarding y_i from Q. If, at any stage, the node x_i considered by the process has no neighbors in the remaining set Q, then the process fails and we abort our attempt to provide a 2-IRS for G.

We need to argue that with very high probability, the process does not fail. Let F_i be the event that the process fails in the ith step. Note that at the beginning of step i the current set Q is of size

$$\begin{aligned} |Q| \;=\; |A| - (i-1) &\geqslant |A| + 1 - m \\ &\geqslant np - n(1-p)(1-p^2) - 11\sqrt{n \ln n} \\ &\geqslant \frac{n}{2}\left(p - (1-p)(1-p^2)\right) \end{aligned}$$

for sufficiently large n.

Let

$$f(p) \;=\; p - (1-p)(1-p^2) \;=\; -p^3 + p^2 + 2p - 1.$$

Then $f'(p) = -3p^2 + 2p + 2$, which is positive for $0 < p < 1$. Therefore, f increases on this range. Note that[3] $f(0.45) > 0.01$. Therefore, for $0.45 < p < 1$, and for sufficiently large n, $|Q| > n/200$.

Event F_i occurs only if x_i is not connected to any node of Q. This is the intersection of $|Q|$ independent events of probability $1 - p$ each, and hence

$$\mathbb{P}(F_i) \;\leqslant\; (1-p)^{n/200} \;<\; c^{-n}$$

for constant $c > 1$. Let $\mathcal{E}_F(v_0)$ denote the event that the process fails for v_0. This event occurs if for some $x_i \in X$, no remaining common neighbors of v_0 and x_i could be found, i.e., $\mathcal{E}_F(v_0) = \bigcup_i F_i$. We have $\mathbb{P}(\mathcal{E}_F(v_0)) < m\, c^{-n}$. It follows that for a sufficiently large n, the event $\mathcal{E}_F = \bigcup_{v_0} \mathcal{E}_F(v_0)$ has probability $\mathbb{P}(\mathcal{E}_F) \leqslant 1/n^3$.

Combining all possible failure events (namely, $\mathcal{E}_A \cup \mathcal{E}_B \cup \mathcal{E}_F$), we get that for sufficiently large n, the probability that our process fails to generate an interval routing

[3] Actually, the root of $f(p) = 0$, for $0 < p < 1$, is $p_0 \approx 0.445041$.

scheme for the graph with two intervals per edge is bounded from above by $1/n^2$. We remark that all the intervals considered here are *linear*, i.e., of the type $[a, b]$ with $a \leqslant b$, and are *strict*, i.e., do not include the label of the node itself.

THEOREM 2.3. *For sufficiently large n, and for every fixed $0.45 < p < 1$, a random graph $G \in \mathcal{G}_{n,p}$ satisfies $\mathrm{IRS}_R(G) \leqslant 2$ with probability at least $1 - 1/n^2$.* □

2.2. Lower bound. In this subsection, we prove that $\mathrm{IRS}_R(G) \geqslant 2$ for almost every graph G for a random assignment of node labels.

Again, we assume the node labels $\{1, \ldots, n\}$ are assigned randomly for the graph G, so given that G is a random graph in $\mathcal{G}_{n,p}$, for p fixed, we may assume that the nodes are first labeled 1 through n and the edges are randomly drawn only later. As in the previous subsection, we assume the event $\overline{\mathcal{E}_A}$.

We need to show that with high probability, a single interval per edge will not be sufficient for producing shortest paths.

Consider a node $x \in \{1, \ldots, n\}$. Suppose that x is connected to $x + 1$ and $x + 3$ and that $x + 2$ is not connected to any node from $\{x, x + 1, x + 3\}$. Let $I(x, u)$ be the interval assigned to the edge (x, u) that contains $x + 2$. Since the diameter of G is 2, it follows that $u \notin \{x + 1, x + 2, x + 3\}$. $I(x, u)$ must contain u and $x + 2$, but neither $x + 1$ nor $x + 3$, which are connected to x. This contradicts the fact that $I(x, u)$ is composed of a single interval.

Let $x_i = 4i - 3$, for every $i \in \{1, \ldots, m\}$, with $m = \lfloor n/4 \rfloor$. Let K_i denote the event x_i as in the previous configuration, and let $\mathcal{E}_K$ denote the event that there exists an event K_{i_0} that occurs. Note that by the above discussion, the probability of $\mathrm{IRS}_R(G) > 1$ (under the event $\overline{\mathcal{E}_A}$) is lower bounded by $\mathbb{P}(\mathcal{E}_K)$.

Let Z_i be the characteristic random variable of the event K_i, and $Z = \sum_{i=1}^{m} Z_i$. The events K_i are independent, and each one occurs with probability $p^2(1 - p)^3$. Therefore, $\mathbb{P}(Z = 0) = (1 - p^2(1 - p)^3)^m < 1/n^3$ for a sufficiently large n. It follows that $\mathbb{P}(\mathcal{E}_K) \geqslant 1 - 1/n^3$.

THEOREM 2.4. *For sufficiently large n, and for every fixed $0 < p < 1$, a random graph $G \in \mathcal{G}_{n,p}$ satisfies $\mathrm{IRS}_R(G) \geqslant 2$ with probability at least $1 - 1/n^2$.* □

3. Adversely assigned labels. Next we assume the adversary model, in which the assignment of the node labels $\{1, \ldots, n\}$ to nodes is done by an adversary, aiming to cause the routing scheme to use the maximum number of intervals. We show that almost every graph G satisfies $\mathrm{IRS}_A(G) = 3$.

3.1. Upper bound. We start by showing that with probability at least $1 - 1/n^2$, a random graph G from $\mathcal{G}_{n,p}$ satisfies $\mathrm{IRS}_A(G) \leqslant 3$, for every fixed probability p, $1/3 < p < 1$. More generally, we show that for each integer $k \geqslant 2$, $\mathrm{IRS}_A(G) \leqslant k$ with probability at least $1 - 1/n^2$ for $G \in \mathcal{G}_{n,p}$, for each fixed p, $1/k < p < 1$.

Once again, by Lemma 2.1, we are allowed to restrict our attention to the event $\overline{\mathcal{E}_A}$, and assume the graph $G = (V, E)$ at hand is of diameter 2, and such that for every node $v \in V$, $np - 3\sqrt{n \ln n} \leqslant |\Gamma(v)| \leqslant np + 3\sqrt{n \ln n}$.

Consider a node $v_0 \in V$ and an integer $k \geqslant 2$. We need to argue that with high probability, the edges of v_0 can be labeled with at most k intervals per edge so that for every possible destination $v_d \in V$, the selected edge is along a shortest path from v_0 to v_d.

Let $A = \Gamma(v_0) \setminus \{v_0\}$ and $B = V \setminus \Gamma(v_0)$. Let us first define one interval per emanating edge of v_0 to take care of routing to the nodes of A. Namely, for every node $w \in A$, mark the edge (v_0, w) by the interval $[w]$. It is left to show how the

remaining $k-1$ intervals per edge of v_0 can be used to route optimally towards the nodes of B.

This is done as follows. Let $B = \{b_1, \ldots, b_m\}$. Recall that A and B satisfy inequalities (2.1) and (2.2). We now describe a process for selecting an intermediate node $a_i \in A$ for every $1 \leqslant i \leqslant m$ such that the routing from v_0 to b_i will go through a_i. For this, we need to ensure that there is an edge $(a_i, b_i) \in E$ for every $1 \leqslant i \leqslant m$. Once this is done, we mark each edge (v_0, a_i) by the interval $[b_i]$, thus completing our task.

The selection process is similar to the greedy process of section 2.1. Let $Q = A$, and define a counter $C(a)$ for each node $a \in A$, initially setting all counters to zero. Having already selected $a_1, \ldots, a_{i-1}$, the ith step consists of selecting a_i to be some arbitrary neighbor of b_i in Q, increasing the counter $C(a_i)$ by one, and discarding a_i from Q if the counter has reached $k-1$. If, at any stage, the node b_i considered by the process has no neighbors in the remaining set Q, then the process fails and we abort our attempt to provide a k-IRS for G.

We need to argue that with high probability, the process does not fail. Let F_i be the event that the process fails in the ith step. Note that at the beginning of step i the counters sum up to $i-1$, and hence at most $\lfloor (i-1)/(k-1) \rfloor$ nodes were discarded from Q, so the current set Q is of size

$$\begin{aligned} |Q| \;\geqslant\; |A| - \left\lfloor \frac{i-1}{k-1} \right\rfloor &> |A| - \frac{|B|}{k-1} \\ &> np - \frac{n(1-p)}{k-1} - 6\sqrt{n \ln n} - 1 \\ &> \frac{n}{2}\left(p - \frac{1-p}{k-1}\right) \end{aligned}$$

for sufficiently large n. Since $p > 1/k$, and $k > 1$, it implies that

$$p - \frac{1-p}{k-1} \;>\; 0.$$

Therefore, there is a constant $\alpha > 0$ such that for sufficiently large n, $|Q| > \alpha n$.

Event F_i occurs only if b_i is not connected to any node of Q. This is the intersection of $|Q|$ independent events of probability $1-p$ each, and hence $\mathbb{P}(F_i) \leqslant (1-p)^{\alpha n} < c^{-n}$, for constant $c > 1$. Letting $\mathcal{E}_F(v_0)$ denote the event that the process fails for v_0, i.e., $\mathcal{E}_F(v_0) = \bigcup_i F_i$, we have $\mathbb{P}(\mathcal{E}_F(v_0)) \leqslant m\, c^{-n}$. It follows that for a sufficiently large n, the event $\mathcal{E}_F = \bigcup_{v_0} \mathcal{E}_F(v_0)$ has probability $\mathbb{P}(\mathcal{E}_F) \leqslant 1/n^3$.

Combining all possible failure events (namely, $\mathcal{E}_A \cup \mathcal{E}_F$), we get that the probability that our process fails to generate an interval routing scheme for the graph with three intervals per edge is bounded from above by $1/n^2$. We remark that all the intervals used in the scheme are linear and strict.

THEOREM 3.1. *For sufficiently large n, for every integer $k \geqslant 2$, and for every fixed p in the range $1/k < p < 1$, a random graph $G \in \mathcal{G}_{n,p}$ satisfies $\mathrm{IRS}_A(G) \leqslant k$ with probability at least $1 - 1/n^2$.* □

In particular, a graph $G \in \mathcal{G}_{n,1/2}$ satisfies $\mathrm{IRS}_A(G) \leqslant 3$ with probability at least $1 - 1/n^2$.

3.2. Lower bound. We restrict our attention to random graph $G \in \mathcal{G}_{n,1/2}$ and show that $\mathrm{IRS}_A(G) \geqslant 3$ with probability at least $1 - 1/n^2$. As in the previous sections, we assume the event $\overline{\mathcal{E}_A}$; i.e., G has diameter 2.

The main idea is the following. Consider a node v such that $|\Gamma(v)| = d+1$, for some suitable integer $d < n/2$, and let $A = \Gamma(v) \setminus \{v\}$, and $B = V \setminus \Gamma(v)$. Now suppose that $A = \{a_1, \ldots, a_d\}$, and $C = \{b_1, \ldots, b_{d+1}\} \subset B$ such that the following two assumptions hold:

(A1) For every $i \in \{1, \ldots, d\}$, a_i is connected to neither b_i nor b_{i+1}.

(A2) The adversary has labeled the nodes by $a_i = 2i$ and $b_i = 2i-1$ for all $i \in \{1, \ldots, d\}$. (The other nodes have arbitrary labels.)

Since $|A| < |C|$, for every shortest path routing there exist two nodes $b, b' \in C$, $b < b'$, that are reached from v through the same $a \in A$. Thus the set $I_{v,a}$ that labels the edge (v, a) contains a, b, b'. However, $a-1, a+1 \notin I_{v,a}$, and by assumption (A1) $b+1, b'-1 \notin I_{v,a}$. It forces at least three linear intervals for $I_{v,a}$.

However, if the labeling is allowed to be nonstrict and nonlinear, $b = 1$, $b' = n-1$, and $v = n \in I_{v,a}$, it is possible to have b and b' in a single *wraparound* interval which does not contain $b+1$ and $b'-1$. For example, $I_{v,a} = [a] \cup [b', b]$. In order to strengthen our lower bound we will show that, actually, with high probability this node labeling implies three intervals, even if the intervals used are nonstrict and nonlinear. Indeed, it suffices to show that there is a node $b'' \in B \setminus C$ that is not connected to a. From the definition of the node labeling $b < b' < b''$. Therefore, even if $b = 1$ and $v \in I_{v,a}$, the set $I_{v,a}$ cannot contain the subinterval $[b', b]$ since it would contain b''.

First, we prove[4] that with high probability there must be some node v such that $|\Gamma(v)| \leqslant n/2 - 16\log^2 n$.

LEMMA 3.2. *With probability at least* $1 - 1/n^3$*, there exists a node* v *such that* $|\Gamma(v)| \leqslant n/2 - 16\log^2 n$.

Proof. For every node v, let $\deg(v) = |\Gamma(v)| - 1$. Consider an arbitrary $v \in V$. Note that $\deg(v)$ is the sum of $n-1$ independent Bernoulli random variables each of probability $1/2$. Therefore,

$$\mathbb{P}(\deg(v) = i) \;=\; \frac{\binom{n-1}{i}}{2^{n-1}}.$$

For notational convenience, let $m = n-1$. For every integer h, $0 < h < m/2$,

$$\begin{aligned}
\mathbb{P}(\deg(v) < m/2 - h) &= \frac{1}{2^m} \sum_{i=0}^{m/2-h-1} \binom{m}{i} \\
&= \frac{1}{2^m} \left(\sum_{i=0}^{m/2} \binom{m}{i} - \sum_{i=m/2-h}^{m/2} \binom{m}{i} \right) \\
&\geqslant \frac{1}{2^m} \left(2^{m-1} - \sum_{i=m/2-h}^{m/2} \binom{m}{i} \right).
\end{aligned}$$

By the Stirling formula, $\binom{m}{m/2} \leqslant c \cdot 2^m/\sqrt{m}$ for some constant $c > 0$. Also, $\binom{m}{i} \leqslant \binom{m}{m/2}$ for every $0 \leqslant i \leqslant m$. Hence,

$$\begin{aligned}
\mathbb{P}(\deg(v) < m/2 - h) &\geqslant \frac{1}{2^m} \left(2^{m-1} - (h+1)c \cdot 2^m/\sqrt{m} \right) \\
&\geqslant \frac{1}{2} - \frac{(h+1)c}{\sqrt{m}}.
\end{aligned}$$

[4] Again, making no attempt to optimize the constants involved in the calculations.

Hence, for $h = 16\log^2 n + 1$, we have $\mathbb{P}(\deg(v) < m/2 - h) \geqslant 1/4$ for sufficiently large n. The probability that all the nodes have degree at least $m/2 - h$ is thus bounded by $(3/4)^n \leqslant 1/n^3$ for sufficiently large n. Hence, with probability at least $1 - 1/n^3$ there exists a node v such that $\deg(v) \leqslant n/2 - 16\log^2 n - 1$, and the lemma follows. □

Hence, we have $|B| - |A| \geqslant 32\log^2 n$. Define a "walk" in G such that any two consecutive nodes of the walk are nonadjacent, as follows. Start the walk at an arbitrary node b_1 of B, continuing to an arbitrary nonneighbor a_1 in A, from there back to an arbitrary nonneighbor b_2 in B, and so on. Continue in that fashion for the first $|A| - \lceil 8\log n\rceil$ "double steps" (each consisting of two substeps, from B to A and back to B) and ending at some node $a_t \in A$.

LEMMA 3.3. *With probability at least $1 - 1/n^3$, the walk does not get stuck during its first $|A| - \lceil 8\log n\rceil$ steps.*

Proof. First, consider a random v. Let $Q_B = B$ and $Q_A = A$. Having already selected $b_1, a_1, \ldots, b_{i-1}, a_{i-1}$, the ith step of the walk consists of choosing an arbitrary $b_i \in Q_B \setminus \Gamma(a_{i-1})$ and discarding b_i from Q_B. It fails if $|Q_B \setminus \Gamma(a_{i-1})| = 0$, i.e., if a_{i-1} is connected to all nodes of Q_B. Let F_B^i be this failure event. This event is the intersection of $|Q_B|$ independent events of probability $1/2$ each. Note that at any step $|Q_B| \geqslant |B| - |A| \geqslant 32\log^2 n$; thus

$$\mathbb{P}(F_B^i) \;\leqslant\; \left(\frac{1}{2}\right)^{32\log^2 n} \;<\; \frac{1}{n^{32}}.$$

Then we choose $a_i \in Q_A \setminus \Gamma(b_i)$, discarding a_i from Q_A. Let F_A^i denote the event "$|Q_A \setminus \Gamma(b_i)| = 0$." It occurs if b_i is connected to all nodes of Q_A. This is the intersection of $|Q_A|$ independent events of probability $1/2$ each. Since $|Q_A| \geqslant 8\log n$,

$$\mathbb{P}(F_A^i) \;\leqslant\; \left(\frac{1}{2}\right)^{8\log n} \;=\; \frac{1}{n^8}.$$

Thus letting $F_i = F_A^i \cup F_B^i$ be the event that the walk fails at the ith step, we have

$$\mathbb{P}(F_i) \;<\; \frac{2}{n^8}.$$

Therefore, the walk fails within the first $|A| - \lceil 8\log n\rceil < n/2$ steps with probability

$$\mathbb{P}\left(\bigcup_i F_i\right) \;<\; \frac{1}{n^7}.$$

Finally, the probability that the walks from any of the nodes v fails is bounded by

$$\mathbb{P}\left(\bigcup_v \bigcup_i F_i\right) \;<\; \frac{1}{n^3}. \qquad \square$$

Let $\mathcal{E}_W$ denote the event that G does not satisfy the property asserted in Lemma 3.3. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_W}$.

Let the first segment of the walk consist of the following sequence of nodes:

$$b_1, a_1, b_2, a_2, \ldots, b_t, a_t.$$

At the end of the first stage, there are only $k = d - t = \lceil 8 \log n \rceil$ nodes remaining in A, $a_{t+1}, \ldots, a_d$, and more than $32 \log^2 n$ nodes in B. Partition the remaining nodes of B arbitrarily into $k+1$ groups of $4 \log n$ nodes each, denoted by $B_{t+1}, \ldots, B_{d+1}$. (The last group may be larger.)

The only thing that remains to do is to pick in each set B_i a distinct node b_i that neighbors both a_{i-1} and a_i, for $i \in \{t+1, \ldots, d\}$. The resulting second segment of the walk would be

$$b_{t+1}, a_{t+1}, \ldots, b_d, a_d.$$

This can be done with high probability again. (For the last step, of choosing b_{d+1}, we need only to verify that it neighbors a_d.)

LEMMA 3.4. *With probability at least* $1 - 1/n^3$, *the second segment of the walk can be completed successfully.*

Proof. The formal proof requires some care, since it is necessary to show that the events are independent. In particular, for the second stage, we are left with some nodes in A which were not chosen completely randomly, since these are nodes that perhaps were not connected to various nodes along the first segment of the walk. However, the events we look at in the second stage are independent of the events considered earlier. In particular, for each a_i and each $b \in B_i$, the events considered are "a_i is connected to b" and "a_{i-1} is connected to b," and these events are indeed independent of any event considered in the first stage, and of each other. Moreover, the probability of each such event is exactly $1/2$. Therefore,

$$\mathbb{P}(B_i \cap \Gamma(a_i) = \varnothing) \;\leqslant\; \frac{1}{2^{4 \log n}} \;=\; \frac{1}{n^4},$$

and similarly

$$\mathbb{P}(B_i \cap \Gamma(a_{i-1}) = \varnothing) \;\leqslant\; \frac{1}{n^4}.$$

Hence, letting $\mathcal{E}_{W'}$ denote the event that the property asserted in the lemma does not hold, we have

$$\mathbb{P}(\mathcal{E}_{W'}) \;\leqslant\; \frac{2(d-t)}{n^4} \;\leqslant\; \frac{1}{n^3}. \qquad \square$$

The combined path now consists of all nodes of A and $|A| + 1$ nodes of B, and the proof follows for linear intervals. To show that the lower bound holds also for nonstrict and nonlinear intervals, it remains to show the following lemma.

LEMMA 3.5. *With probability at least* $1 - 1/n^3$, *there is no node* $a \in A$ *connected to all nodes of* $B \setminus \{b_1, \ldots, b_{d+1}\}$.

Proof. Let $C = \{b_1, \ldots, b_{d+1}\}$. The probability that all the nodes of $B \setminus C$ are connected to a random node $a \in A$ is

$$\left(\frac{1}{2}\right)^{|B \setminus C|} \;<\; \frac{1}{n^4}$$

since $|B \setminus C| > 4 \log n$. Therefore, the probability that at least one node of A is connected to all of them of $B \setminus C$ is upper bounded by $|A|/n^4 < 1/n^3$. $\square$

Combining all possible failure events (namely, $\mathcal{E}_A$, Lemma 3.2, $\mathcal{E}_W$, Lemma 3.4, and Lemma 3.5), we obtain the following theorem.

THEOREM 3.6. *For sufficiently large* n, *with probability at least* $1 - 1/n^2$, *a random graph* $G \in \mathcal{G}_{n,1/2}$ *satisfies* $\mathrm{IRS}_A(G) \geqslant 3$. $\square$

4. Designer chosen labels. We next assume the designer model, in which the assignment of the node labels $\{1,\ldots,n\}$ to nodes is done by the designer of the routing scheme, aiming to minimize the number of intervals used by the routing scheme.

In this case, the only lower bound we have at the moment is the trivial $\mathrm{IRS}(G) \geqslant 1$ for every graph G. In the opposite direction, we are also unable so far to prove an upper bound of 1 on the maximum number of intervals per edge.

However, we will show that it is possible to assign the node labels in such a way that, while some edges might still require two intervals, the number of such violations will be very small, and more specifically, bounded by $O(\log^3 n)$ with high probability. In this section, we restrict our attention to the case $p = 1/2$, so $G \in \mathcal{G}_{n,1/2}$.

The idea behind the selection process is the following. Suppose that the node set of the given random graph is partitioned into cliques $V = C_1 \cup \cdots \cup C_m$. Label the nodes of V according to this partition, so that the nodes of each clique C_i are numbered consecutively. Now use this partition to define the routing scheme as follows. Consider a sender v_0. Suppose that $v_0 \in C_J$, and consider some other clique C_I. The central property we rely upon is that if v_0 is adjacent to *some* of the nodes of C_I, then all the nodes of C_I can be provided for using a single interval on each edge going from v_0 to the nodes of C_I, as follows. Let $C_I = \{p, p+1, \ldots, q\}$. If v_0 has a unique neighbor ℓ in C_I, then mark the edge from v_0 to ℓ by the interval $[p,q]$. Otherwise, suppose v_0 has neighbors $\ell_1 < \ell_2 < \cdots < \ell_k$ in C_I. Then the edges $e_j = (v_0, \ell_j)$ leading from v_0 to these nodes can be labeled by intervals $I(e_j)$, as follows:

$$I(e_j) = \begin{cases} [p, \ell_2 - 1], & j = 1, \\ [\ell_j, \ell_{j+1} - 1], & 1 < j < k, \\ [\ell_k, q], & j = k. \end{cases}$$

Note that this choice of intervals also takes care of the special case of $C_I = C_J$ itself, where every node other than v_0 itself is a neighbor of v_0.

Thus we are left only with the need of handling the cliques C_i, none of whose nodes are adjacent to v_0. Call these cliques the "remote" cliques. The nodes of these remote cliques must be reached through nodes of other cliques, potentially using additional intervals, and at worst using a unique new interval for each node. It is thus required to bound from above the maximum number of nodes in the remote cliques. Towards this goal, we rely intuitively on the fact that large cliques are unlikely to be remote. More precisely, the probability that a clique of size k is remote is roughly $1/2^k$. It thus becomes necessary to explore the distribution of clique sizes in a clique partition of random graphs or at least generate partitions with favorable size distributions.

We make use of the following two properties of random graphs. (In the following, the function log denotes the logarithm in base 2.) First, regarding the size of the maximum clique, we have (cf. Chapt. XI.1 of [2]) the following lemma.

LEMMA 4.1. *With probability at least* $1 - 1/n^{\log\log n}$, *the maximum clique in a random graph* $G \in \mathcal{G}_{n,1/2}$ *is of size at most* $2\log n$.

Let $\mathcal{E}_C$ denote the event that the random graph at hand does not satisfy the property asserted in Lemma 4.1. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_C}$. As before, we also restrict ourselves to $\overline{\mathcal{E}_A}$.

Second, we make use of a natural technique for generating a clique partition of a given graph. This technique is the "mirror image" of the greedy algorithm often used to generate a legal coloring for a graph. This simple algorithm operates as follows. Start by ordering the nodes arbitrarily, numbering them as $1, 2, \ldots, n$. Assign the nodes to cliques $C_1, C_2, \ldots, C_n$ one by one, assigning each node to the

smallest-indexed admissible clique. Node 1 is thus assigned to C_1, node 2 is assigned to C_1 if it is a neighbor of node 1, otherwise it is assigned to C_2, and so on. It is known (cf. Chapt. XI.3 of [2]) that with high probability this process will pack the nodes of the given random graph G in fewer than $n/\log n$ cliques. Moreover, analyzing the process in more detail, we will derive bounds on the number of small cliques generated. Specifically, there will be no more than $2^k \log n$ cliques of size k with high probability. Coupled with Lemma 4.1, this can be used to show that the total number of nodes in remote cliques is bounded (with high probability) by about

$$\sum_{k=1}^{2\log n} k \cdot \frac{1}{2^k} \cdot 2^k \log n = O(\log^3 n).$$

The problem that makes formalizing this argument somewhat more difficult is that once the partition is calculated, the graph can no longer be treated as random, as the fact, say, that v_0 is not in the clique C_i bears some implications on the probability that v_0 is connected to some node of C_i and prevents us from assuming that all the events considered in the analysis are independent. Nevertheless, the dependencies can be bounded and turn out to have little effect on the resulting probabilities.

Let us fix our attention on a node v_0, belonging to the clique C_J, and on another clique C_I. We would like to bound the probability that v_0 is not connected to any node of C_I.

For every clique C_i and node $v \in V$, partition C_i into $C_i = \mathcal{B}_i(v) \cup \mathcal{A}_i(v)$, where $\mathcal{B}_i(v)$ consists of all the nodes that entered C_i *before* v was considered by the algorithm, namely, $\mathcal{B}_i(v) = \{w \in C_i \mid w < v\}$, and $\mathcal{A}_i(v) = C_i \setminus \mathcal{B}_i(v)$, the nodes added to C_i *after* v was added to some clique. Let $\beta_i(v) = |\mathcal{B}_i(v)|$ and $\alpha_i(v) = |\mathcal{A}_i(v)|$. In particular, let $\mathcal{B} = \mathcal{B}_I(v_0)$, $\mathcal{A} = \mathcal{A}_I(v_0)$, $\beta = \beta_I(v_0)$, and $\alpha = \alpha_I(v_0)$.

LEMMA 4.2. *If $I < J$, then the probability that C_I is remote from v_0 is at most $1/2^{|C_I|-1}$.*

Proof. We will actually prove the somewhat stronger claim that if $I < J$, then the probability that v_0 is not connected to any node in C_I is $\frac{1}{2^\alpha(2^\beta-1)}$ that is at most $1/2^{|C_I|-1}$ because $\beta \geqslant 1$.

Since $I < J$, when the greedy algorithm considered v_0, it had to examine (and reject) the possibility of adding it to C_I before actually adding it to C_J. The fact that v_0 was not added to C_I implies that there is some node in $\mathcal{B}$ that does not neighbor v_0. However, of all 2^β possible connection configurations between v_0 and the nodes of $\mathcal{B}$, the event $\mathcal{E}_N =$ "v_0 has a nonneighbor in $\mathcal{B}$" excludes only the possibility that v_0 neighbors all nodes of $\mathcal{B}$ and leaves us with $2^\beta - 1$ other possibilities. Hence, conditioned on $\mathcal{E}_N$, we have

$$\mathbb{P}(v_0 \text{ has no neighbors in } \mathcal{B}) \;=\; \frac{1}{2^\beta - 1}.$$

As for the nodes of $\mathcal{A}$, each such node v was added to C_I after v_0 was considered, and since $I < J$, the decision to add v into C_I was reached before considering clique C_J, and hence it was independent of the existence (or nonexistence) of the edge (v, v_0). Hence,

$$\mathbb{P}(v_0 \text{ has no neighbors in } \mathcal{A}) \;=\; \frac{1}{2^\alpha}.$$

The lemma follows. ∎

LEMMA 4.3. *If $I > J$, then the probability that C_I is remote from v_0 is at most*

$$\frac{1}{2^\beta} \cdot \prod_{v \in \mathcal{A}} \frac{2^{\beta_J(v)-1}}{2^{\beta_J(v)}-1}.$$

Proof. Since $I > J$, when the greedy algorithm considered each node v of C_I, it had to first examine (and reject) the possibility of adding it to C_J. For $v \in \mathcal{B}$, the decision not to add v to C_I was clearly independent of the edge (v, v_0). (Note that in fact $v_0 \in \mathcal{A}_J(v)$.) Hence,

$$\mathbb{P}(v_0 \text{ has no neighbors in } \mathcal{B}) \;=\; \frac{1}{2^\beta}.$$

It remains to consider nodes $v \in \mathcal{A}$.

The fact that a node $v \in \mathcal{A}$ was not added to C_J implies that there exists a node in $\mathcal{B}_J(v)$ that does not neighbor v. However, again, of all $2^{\beta_J(v)}$ possible connection configurations between v and the nodes of $\mathcal{B}_J(v)$, the event $\mathcal{E}_N(v)$ = "v has a nonneighbor in $\mathcal{B}_J(v)$" excludes only the possibility that v neighbors all nodes of $\mathcal{B}_J(v)$ and leaves us with $2^{\beta_J(v)} - 1$ other possibilities. Of those, v neighbors v_0 in exactly $2^{\beta_J(v)-1}$ possibilities. Hence, conditioned on $\mathcal{E}_N(v)$, the probability that v does not neighbor v_0 is $\frac{2^{\beta_J(v)-1}}{2^{\beta_J(v)}-1}$. Hence,

$$\mathbb{P}(v_0 \text{ has no neighbors in } \mathcal{A}) \;=\; \prod_{v \in \mathcal{A}} \frac{2^{\beta_J(v)-1}}{2^{\beta_J(v)}-1}.$$

The lemma follows. □

The product appearing in the bound of Lemma 4.3 is small only when the values $\beta_J(v)$ involved in it are sufficiently large. Fortunately, there cannot be too many nodes v with small $\beta_J(v)$ values, as we prove next.

For integer $k \geqslant 1$, let $X_J(k)$ denote the set of nodes v that were considered by the algorithm during the period when C_J contained exactly k nodes and were rejected from C_J. In particular, we are interested in the collection of such nodes for small values of k, i.e., $\hat{X} = \bigcup_{k=1}^{\log\log n} X_J(k)$.

COROLLARY 4.4. *Suppose that the clique C_I, $I > J$, contains no node from $\hat{X}$. Then the probability that v_0 is not connected to any node in C_I is at most $\gamma/2^{|C_I|}$ for some fixed constant $\gamma > 1$.*

Proof. Under the assumption of the corollary, $\beta_J(v) > \log\log n$ for every $v \in \mathcal{A}$. Therefore,

$$\frac{2^{\beta_J(v)-1}}{2^{\beta_J(v)}-1} \;=\; \frac{1}{2}\left(1 + \frac{1}{2^{\beta_J(v)}-1}\right) \;\leqslant\; \frac{1}{2}\left(1 + \frac{1}{2^{\log\log n+1}-1}\right) \;\leqslant\; \frac{1}{2}\left(1 + \frac{1}{\log n}\right).$$

The bound of Lemma 4.3 thus becomes

$$\frac{1}{2^\beta} \cdot \left(\frac{1}{2}\left(1 + \frac{1}{\log n}\right)\right)^\alpha.$$

As the size of the maximum clique in a random graph is at most $2\log n$ (with probability at least $1 - 1/n^{\log\log n}$), this bound is no greater than

$$\frac{1}{2^\beta} \cdot \frac{1}{2^\alpha}\left(1 + \frac{1}{\log n}\right)^{2\log n} \;\leqslant\; \frac{1}{2^{\beta+\alpha}} \cdot e^2,$$

and the claim follows. □

LEMMA 4.5. *With probability at least* $1-1/n^3$, *the set* $X_J(k)$ *is of size* $|X_J(k)| \leqslant 2^{k+2}\ln n$ *for every* $k \geqslant 1$.

Proof. Suppose that $|X_J(k)| > 2^{k+2}\ln n$. For every $v \in X_J(k)$, the probability for v not joining C_J (on account of a missing edge from v to some node in C_J) is $1-1/2^k$. Thus the probability of all of those nodes being rejected from C_J is

$$\left(1-\frac{1}{2^k}\right)^{|X_J(k)|} \;<\; \left(1-\frac{1}{2^k}\right)^{2^{k+2}\ln n} \;\leqslant\; e^{-4\ln n} \;=\; \frac{1}{n^4}.$$

Summing these probabilities over all k yields the desired claim. □

Let $\mathcal{E}_D$ denote the event that the random graph at hand does not satisfy the property asserted in Lemma 4.5. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_D}$. Under this restriction, the size of the set $\hat{X}$ is bounded above by

$$|\hat{X}| \leqslant \sum_{k=1}^{\log\log n} 2^{k+2}\ln n = O(\log^2 n).$$

It remains to bound the number of remote cliques C_I (that have no neighbor of v_0). Let $f(k)$ denote the number of cliques of size k.

LEMMA 4.6. *With probability at least* $1-1/n^2$, $f(k) \leqslant 2^{k+2}\ln n$ *for every* $k \geqslant 1$.

Proof. Let us bound the probability of the event that there are more than $2^{k+1}\log n$ cliques of size k, $C_{i_1},\ldots,C_{i_{f(k)}}$. Let $m = 2^{k+2}\ln n$ and consider the time when clique C_{i_m} was formed by the greedy algorithm (for the purpose of hosting the currently inspected node v'). For any node v considered after v', the probability that it could not have joined the clique C_{i_j} is

$$1-\frac{1}{2^{\beta_{i_j}(v)}} \;\leqslant\; 1-\frac{1}{2^k}.$$

Hence, the probability that v could not have joined any of those m cliques is at most

$$\left(1-\frac{1}{2^k}\right)^m \;\leqslant\; \left(1-\frac{1}{2^k}\right)^{2^{k+2}\ln n} \;\leqslant\; e^{-4\ln n} \;=\; \frac{1}{n^4}.$$

Consequently, the probability that any of the remaining nodes to be considered by the algorithm after v' could not join an existing clique, and a new clique must be formed, is at most $1/n^3$. Summing these probabilities for every k, the lemma follows. □

Let $\mathcal{E}_H$ denote the event that the random graph at hand does not satisfy the property asserted in Lemma 4.6. Henceforth, we ignore that possibility and restrict our attention to $\overline{\mathcal{E}_H}$.

LEMMA 4.7. *The number of remote cliques is at most* $O(\log^2 n)$ *with probability* $1-1/n^2$.

Proof. Assuming event $\overline{\mathcal{E}_D}$, the total number of remote cliques that contain a node of $\hat{X}$ is at most $O(\log^2 n)$. It remains to count the remote cliques among the cliques that do not contain any node of $\hat{X}$. The probability of such a clique C_I being remote is bounded, in Lemma 4.2 and Corollary 4.4, by $\delta/2^{|C_I|}$ for some constant $\delta > 1$.

For every clique C_i of size k, let R_i be the event that C_i is remote. Let R be a random variable representing the number of remote cliques of size k, and let $f_R(k)$

denote its expectation. Since R is the sum of $f(k)$ Bernoulli random variables R_i, each with probability $\delta/2^k$, $f_R(k)$ is at most $\delta f(k)/2^k$. Assuming event $\overline{\mathcal{E}_H}$, we have

$$f_R(k) \leqslant \frac{\delta 2^{k+2} \ln n}{2^k} = 4\delta \ln n.$$

Applying Chernoff's bound, we get that

$$\mathbb{P}(R \geqslant 8\delta \ln n) < \exp(-2\delta \ln n) = n^{-2\delta} < \frac{1}{n^2}.$$

Hence, for the cliques not containing any nodes from $\hat{X}$, with probability at least $1-1/n^2$, the total number of remote cliques of any size k is bounded (recalling Lemma 4.1) by $O(\log^2 n)$.

Combining both clique types together, we get the claim of the lemma. □

It follows that for every v_0, the number of "problematic" nodes (namely, those of remote cliques) that need to be assigned an individual interval is bounded by $O(\log^3 n)$ with probability $1 - 1/n$.

Combining all possible failure events (namely, $\mathcal{E}_A \cup \mathcal{E}_C \cup \mathcal{E}_D \cup \mathcal{E}_H$), we get that for a random graph in $\mathcal{G}_{n,1/2}$, with probability at least $1-1/n$, it is possible to assign node labels and design an interval routing scheme in such a way that for every node v_0, there is a single interval on every edge except at most $O(\log^3 n)$ edges with two intervals each. (Spreading the problematic nodes so that each adds an interval to a different edge is done by a greedy process similar to those of sections 2.1 and 3.) We remark that the intervals used in the scheme are linear and strict.

THEOREM 4.8. *For sufficiently large n, with probability at least $1-1/n$, a random graph $G \in \mathcal{G}_{n,1/2}$ can be given an assignment of node labels and a shortest path interval routing scheme (polynomial time constructible) using a single interval per edge, except for at most $O(\log^3 n)$ edges per node where two intervals must be used.* □

COROLLARY 4.9. *For almost every n-node graph G there exists an assignment of node labels from the set $\{1, \ldots, n\}$ and a shortest path routing scheme using at most $n + O(\log^4 n)$ bits of information per node. Moreover, the routing scheme is constructible in polynomial time.*

Proof. Assume G satisfies Theorem 4.8. The interval routing scheme on G can be implemented in each node by a table of $O(\log^3 n)$ integers and a binary vector of n bits. Indeed, every 1-IRS can be implemented in each node in $n + O(\log n)$ bits (cf. [6]). Note that the $O(\log^3 n)$ problematic nodes contribute for at most $O(\log^3 n)$ single intervals, each one composed of exactly one node label. We store in a table of $O(\log^3 n)$ entries the label of these nodes and the output port number that makes an overhead of $O(\log^4 n)$ bits in total. These nodes are treated as exceptions and checked first in the routing process. Therefore, the label of these nodes can be merged to the remaining intervals in order to simulate a 1-IRS. □

Acknowledgment. The authors thank the anonymous referee for calling their attention to some inaccuracies in the original version of the paper and making a number of suggestions that improved the presentation.

REFERENCES

[1] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.
[2] B. BOLLOBÁS, *Random Graphs*, Academic Press, New York, 1975.

[3] H. Buhrman, J.-H. Hoepman, and P. Vitányi, *Space-efficient routing tables for almost all networks and the incompressibility method*, SIAM J. Comput., 28 (1999), pp. 1414–1432.
[4] H. Buhrman, M. Li, J. Tromp, and P. Vitányi, *Kolmogorov random graphs and the incompressibility method*, SIAM J. Comput., 29 (1999), pp. 590–599.
[5] M. Flammini, J. van Leeuwen, and A. Marchetti-Spaccamela, *The complexity of interval routing on random graphs*, in Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Sciences, J. Wiederman and P. Hájek, eds., Lecture Notes in Comput. Sci. 969, Springer-Verlag, Berlin, 1995, pp. 37–49.
[6] C. Gavoille, *A survey on interval routing*, Theoret. Comput. Sci., 245 (2000), pp. 217–253.
[7] C. Gavoille and D. Peleg, *The compactness of interval routing*, SIAM J. Discrete Math., 12 (1999), pp. 459–473.
[8] C. Gavoille and S. Pérennès, *Memory requirement for routing in distributed networks*, in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, 1996, pp. 125–133.
[9] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, Comput. J., 28 (1985), pp. 5–8.
[10] J. van Leeuwen and R. B. Tan, *Interval routing*, Comput. J., 30 (1987), pp. 298–307.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 722–761

TOPOLOGICAL LOWER BOUNDS ON ALGEBRAIC RANDOM ACCESS MACHINES*

AMIR M. BEN-AMRAM† AND ZVI GALIL†

Abstract. We prove general lower bounds for set recognition on random access machines (RAMs) that operate on real numbers with algebraic operations $\{+, -, \times, /\}$, as well as RAMs that use the operations $\{+, -, \times, \lfloor\ \rfloor\}$. We do it by extending a technique formerly used with respect to algebraic computation trees. In the case of algebraic computation trees, the complexity was related to the number of connected components of the set W to be recognized. For RAMs, four similar results apply to the number of connected components of W°, the topological interior of W. Two results use $(\overline{W})^\circ$, the interior of the topological closure of W.

We present theorems that can be applied to a variety of problems and obtain lower bounds, many of them tight, for the following models:

1. A RAM which operates on real numbers, using integers to address memory and either the operations $\{+, -, \times, /\}$ or $\{+, -, \times, \lfloor\ \rfloor\}$.

2. A RAM of each of the above instruction sets, extended by allowing arbitrary real numbers to be used as memory addresses and adding a test-for-integer instruction.

3. A RAM of each of the above instruction sets which can compute with arbitrary real numbers, as well as use them for memory addressing, while the input is restricted to the integers. (For one result on this model, we require that all program constants be rational.)

Key words. algebraic computation trees, element distinctness, knapsack, component counting arguments

AMS subject classifications. 68Q25, 68P10

PII. S0097539797329397

1. Introduction.

Consider a set $W \subset \Re^n$ and the problem of recognizing W. Ben-Or proved that an algebraic computation tree recognizing W requires height that is at least a logarithm of the number of path-connected components[1] of W [2]. For the exact statement see Theorem 3.1 in section 3. Yao proved that under certain conditions, an ACT recognizing *integer* elements of W requires a height that is at least a logarithm of the number of connected components of W that have a nonempty interior [19]. For the exact statement see Theorem 3.2 in section 3. The question we ask and answer in this paper is, Can we prove similar results for random access machine models with powerful instruction sets?

A well-known example, the element distinctness problem, is defined by $W = \{x \in \Re^n |\ x_i \neq x_j \text{ for } 1 \leq i \neq j \leq n\}$. Both Ben-Or and Yao used their theorems to obtain (as an easy corollary) an $\Omega(n \log n)$ lower bound on the height of a computation tree that solves this problem. However on a random access machine (RAM), the problem can be easily solved in time $O(n)$ by storing a flag in address x_i for $i = 1, \ldots, n$. This indicates that perhaps the answer to our question above is negative. On the other hand, this solution may perhaps be ruled out: If the x_i are real numbers, can we expect to be able to use them as memory addresses? We return to this question,

*Received by the editors October 30, 1997; accepted for publication (in revised form) December 20, 2000; published electronically November 13, 2001.

http://www.siam.org/journals/sicomp/31-3/32939.html

†Department of Computer Science, Columbia University, 450 Computer Science Building, New York, NY 10027 (galil@cs.columbia.edu, amirben@mta.ac.il). The second author was partially supported by NSF grant CCR-93-16209 and CISE Institutional Infrastructure grant CDA-90-24735.

[1]It turns out that in applying this result, as well as all other theorems given in this paper, it never becomes necessary to distinguish path-connectedness from connectedness. Therefore, we neglect this distinction througout.

for the case of real-number input, later on. However, in the case of integer input, the objection does not arise, so in this setting it is clear that the applicability of computation-tree lower bounds to the RAM should be questioned. Interestingly, our results for the integer-constrained model are obtained using the techniques developed for the unconstrained case.

We start by making a couple of simple observations concerning Ben-Or's and Yao's theorems. We show that Ben-Or's theorem still holds if, instead of counting the connected components of W, we count those of W°, the interior of W. For some sets, this yields a stronger result; for others, the original version is stronger. We can also make the same modification to Yao's theorem, obtaining a result which is always at least as strong as in his formulation, and further gives these two theorems a more similar form. The importance of these observations is that they hint that one may obtain results on the complexity of recognizing W by counting the components of a *different, related set*; this twist turns out to be crucial for our results on random access machines.

Our first new results are corresponding theorems for the real-number algebraic RAM, i.e., a RAM with operations $\{+, -, \times, /\}$. We consider two main variants of this model. In the standard variant, only integers may be used to address memory locations. For this variant, we prove that recognizing W requires time that is at least a logarithm of the number of connected components of W°, as in our version of Ben-Or's theorem.

Referring again to the element distinctness problem, the proofs of the lower bounds in [2] and [19] use a set W such that both W and W° have $n!$ connected components. Thus the lower bound of $\Omega(n \log n)$ time applies to this model.

We also consider a "nonstandard" model which allows arbitrary real numbers to be used as addresses. For this model we prove that the time to recognize W is at least a logarithm of the number of connected components of $(\overline{W})^\circ$, the interior of the closure of W. The result also holds if an instruction is added to the model, which tests (at unit cost) whether a given value is an integer.

In the case of Element Distinctness, it turns out that $(\overline{W})^\circ$ consists of a single component. This explains why the lower bound for Element Distinctness cannot be obtained under this model, and indeed it supports the fast algorithm suggested above.

We also show that under conditions similar to those required by Yao, recognizing integer elements of W requires at least a logarithm of the number of connected components of $(\overline{W})^\circ$, just as in the case of real input.

We proceed to consider the truncation operation, also known as *floor*. We show that results very similar to the above three propositions also hold for corresponding models with operations $\{+, -, \times, \lfloor\ \rfloor\}$. Note that the integrality-test operation, allowed in one of the previous results, can be simulated in a straightforward way using *floor*; however, incorporating the full power of *floor* in our theorems forces us to leave *division* out. This is justified by the fact that certain problems (subject to our previous lower bounds) have faster algorithms on the model that combines *floor* with division. A well-known example is Gonzalez's algorithm for the *max gap* problem [16] which runs in linear time, contrasting an $\Omega(n \log n)$ lower bounds given by our theorems. Another example of this kind is a linear-time algorithm for the decision problem Min Gap, which is given in the appendix.

The six lower-bound theorems are given in section 3.

In section 4 we consider a dozen of examples of set-recognition problems. In some cases the lower bounds proved for algebraic computation trees also apply to random

access machines, while in some other cases they do not. In most of the latter cases, we are able to show that like in the case of element distinctness, a linear time algorithm exists on the RAM. The two exceptions are the knapsack problem and the generalized knapsack problem. We remark that in these two problems the computation-tree lower bound is larger than $O(n \log n)$. The significance of this fact will be seen later on.

Sections 5–7 are concerned with proving the above theorems. An important tool in the proofs is a technique that was introduced by Paul and Simon [15]. Informally, if a RAM recognizes a set W in time bounded by t, we are able to build a computation tree of height t that recognizes a set V, such that V approximates W in a certain sense. We develop this basic idea into six lemmas that are used in the proofs of the six theorems. An additional idea which is used in the last result, involving the truncation operation in conjunction with integer input, draws from work of Lürwer-Brüggemeier and Meyer auf der Heide [9]. This result requires that the program only use rational constants.

We now review some related results. Paul and Simon applied their technique to prove that arithmetic RAMs (later extended to algebraic RAMs [5]) do not perform better than decision trees for certain problems where decision-tree lower bounds are known (e.g., sorting). Using the same technique, Klein and Meyer auf der Heide extended in [8] the $\Omega(n^2)$ lower bound for *knapsack* (which was known for computation trees [6]) to a RAM which can only add and subtract (but not multiply or divide), and in [12] a generalized knapsack problem was considered. These papers already used component counting together with the Paul–Simon technique, but in a direct fashion (i.e., they count the components of W). This is made possible by the simple structure of sets recognized by this restricted model.

In another paper [13], Meyer auf der Heide stated a result that allows lower bounds for computation trees to be converted to RAM lower bounds, where the RAM may use all the operations that are allowed in the computation tree; hence, in our case, it may be used to infer lower bounds for algebraic RAMs. The result is given by the following lemma.

LEMMA 1.1. *Given a real-number RAM program P, we can construct an algebraic computation tree T that computes the same function, and whenever P takes t time units to process a certain input, the corresponding path in the tree is of length $O(t \log(n + t))$.*

A detailed proof is given in [1]. From this lemma it follows that a lower bound of $f(n)$ for a computation tree implies a lower bound of $\Omega(f(n)/\log(n + f(n)))$ for the RAM. Thus this theorem implies a nonlinear lower bound for the RAM only if the computation-tree lower bound is larger than $\Omega(n \log n)$, and it can never show that the complexity on the RAM is the same as in an algebraic computation tree. These gaps may be closed by our results.

The power of *floor* in computation trees has been investigated in numerous papers [3, 4, 5, 7, 9, 10, 11]; Mansour, Schieber, and Tiwari [10] also extend their lower bound technique to random access machines. It applies to a rational-number RAM with operations $\{+, -, \times, /, \lfloor\ \rfloor\}$ and an initially zero memory. This result, and most of the others, relate to problems where the time of computation depends on the size of the input value or values. In contrast, in this paper we consider *genuinely time-bounded computations* [13], namely programs whose running time is bounded by a function of n, the number of input values. Among the above papers, lower bounds for genuinely time-bounded computations appear only in [9].

All of the above results do not allow the RAM to use preinitialized tables. On the

contrary, all our lower bounds allow the program to make use of finite tables, whose size and contents may depend on n. This means, in particular, that they apply to nonuniform programs in the usual sense (i.e., the program code depends on n).

In our lower bounds for problems with real input, we allow an even stronger model: All memory cells of integer addresses may be assumed to contain initial values that depend on n. This very strong extension cannot be allowed when the input is an integer, for then it allows every problem to be solved in linear time. We sketch the way this can be done: First encode the input n-tuple in a single number, using repeated application of a standard pairing function; e.g., $\pi(x, y) = (x+y)^2 + x$. Then use this number to index an infinite table that contains all the right answers. The correctness of the procedure follows from the injectivity of the pairing function.

2. Preliminaries. In this section we define our models of computation and list some mathematical notation used in expressing the results as well as in the proofs.

2.1. Models of computation. An *algebraic computation tree* models a computational process in which each step has one of the following types: (i) an assignment $z \leftarrow u$; (ii) an arithmetic operation $z \leftarrow u \circ w$, where $\circ \in \{+, -, \times, /\}$; (iii) a comparison $u : 0$. The variables u, w stand for either input variables (denoted $x_1, \ldots, x_n$) or the results of prior computation, while z stands for a new variable name (thus every variable is only assigned once).

Computation starts at the root and proceeds down the tree as follows: Each arithmetic node has one child, which is its successor. Each comparison node has three children associated with the possibilities $>$, $<$, and $=$. Computation proceeds along the appropriate branch. Each leaf is labeled with a constant or a variable that represents the output of the computation. An algebraic computation tree is said to *recognize a set* if each output is a 0 or a 1, where the value of 1 represents acceptance. (Note that we may assume that the output value is a constant rather than a variable; otherwise, add a test, and have distinct leaves for acceptance and rejection.) The tree is said to recognize $W \subseteq \Re^n$ when an accepting leaf is reached if and only if $(x_1, \ldots, x_n) \in W$. The tree is said to recognize W for a restricted set of inputs (e.g., the integers) if the last condition is fulfilled for inputs of this set; it does not matter which leaf is reached for other inputs.

We denote by $h(T)$ the height of the tree T, which represents the worst-case time complexity of the computation described by T (more precisely, an upper bound on it; the height is precisely the worst-case time, if every leaf is reachable).

All *RAMs* share the following structure. The machine consists of a processing unit and a memory unit. The processing unit runs the program; to this end it contains a "program counter" that points to the next instruction to be executed. (The different instructions are described below.) It also makes use of a finite set of *operating registers* $r_1, \ldots, r_k$, whose number is fixed for any given program, as they can only be accessed by being named in an instruction. These registers are used for all arithmetics and tests, leaving the *memory* with the sole role of data storage.

This description is still quite general and is made specific by the choice of three parameters.

The *domain* $\mathcal{D}$ is the set of values that may be manipulated by the machine as "units of data." Every memory cell or operating register holds one element of $\mathcal{D}$. This set is the integers or natural numbers in classical RAM models and the real numbers or rational numbers in the models considered in this paper.

The *address space* $\mathcal{A}$ is the set of values that may be used as memory addresses. Thus the size of memory is $|\mathcal{A}|$, and the standard idealized model uses $\mathcal{A} = \mathbb{N}$. In

case that $\mathcal{A}$ is strictly contained in $\mathcal{D}$, a program may *fault* by attempting to use a value in $\mathcal{D} \setminus \mathcal{A}$ as an address. In this case the program may be considered invalid, or we may consider its result to be $\perp$.

The *set of primitives* $\mathcal{F}$ defines the functions that can be computed by the so-called arithmetic instructions. We call the RAM *algebraic* if $\mathcal{F}$ consists of the field operations $\{+, -, \times, /\}$. In this paper, we will also consider the nonalgebraic operations $\lfloor\ \rfloor$ (truncation to an integer) and $\chi_{\mathbb{Z}}$. (The characteristic function of $\mathbb{Z}$ can be used to test whether a given value is an integer.)

The instruction set of the RAM contains the following groups. In the notation for instructions, r_i, r_j, r_k are register names; x is a constant from $\mathcal{D}$. The notation $\langle r_i \rangle$ refers to the memory cell whose address is given by the contents of r_i.

Direct assignments:

$$r_i \leftarrow x$$
$$r_i \leftarrow r_j$$

Memory access:

$$\langle r_i \rangle \leftarrow r_j$$
$$r_j \leftarrow \langle r_i \rangle$$

Flow control instructions:

jump *label*
if $r_i \bowtie r_j$ jump *label*
$\bowtie \in \{=, <, \leq, \ldots\}$
halt

Arithmetic instructions:

$$r_i \leftarrow r_j + r_k$$
$$r_i \leftarrow r_j - r_k$$

etc., as provided by $\mathcal{F}$.

For every RAM program, the initial contents of memory cells is assumed to be zero, except for those that hold the input, and possibly another set of cells, whose contents and size may depend on n but not on the input values; thus we allow nonuniform solutions. Finally, *set recognition* by a RAM is defined as for a computation tree.

We use the notation (D, A, F)-RAM for a model where $\mathcal{D}$, $\mathcal{A}$, and $\mathcal{F}$ have the values D, A, and F, respectively.

A *real-number RAM* is a RAM where $\mathcal{D} = \Re$. Its main justification is as a convenient model for studying problems on real numbers, e.g., in computational geometry; but it is possible that a problem on integer inputs will be solved faster by a program which uses the power of the real-number RAM. Thus, we consider the real-number RAM also in conjunction with integer-constrained problems. (In fact, geometrical problems are often encountered in practice with integer-constrained input, because of the use of raster devices and digitized data.)

We consider two variants of the real-number RAM. The "standard" version has $\mathcal{A} = \mathbb{N}$, while a stronger, nonstandard version has $\mathcal{A} = \Re$. The last choice seems at first to be a far-fetched idealization; however, it may not be so far-fetched in practice. One may think of the real-number RAM as an abstraction of an actual computer

which works with a finite representation of real values, say in floating point. In this case, these representations may be used to address an ordinary random-access memory. Another justification for using an idealized model is that it is elucidates problems; and in fact, it is via our analysis of this model that some of our best results for integer-constrained input (even on the integer RAM) have been obtained.

Finally, in certain results, we are able to allow the program to initialize all integer-addressed memory cells at no cost. This is a stronger extension than ordinary use of finite tables.

2.2. Various notation and terms. For a set $W \subseteq \Re^n$, let $\beta(W)$ be the number of connected components in W. As customary, we denote by W° the topological interior of W, and by $\overline{W}$ its closure. (We use the usual Euclidean topology.) The boundary of W is $\partial W = \overline{W} \setminus W^\circ$ and is a boundary set, i.e., a set of empty interior. (So far, the definitions apply to any metric space.)

A set $W \subseteq \Re^n$ is called *scale invariant* if $\mathbf{x} \in W$ implies $\lambda\mathbf{x} \in W$ for all real $\lambda > 0$. A set $W \subseteq \Re^n$ is said to be *rationally dispersed* if, for every $\mathbf{x} \in \Re^n$ and $\varepsilon > 0$, there is a rational point $\mathbf{z}$ such that $\|\mathbf{z} - \mathbf{x}\| < \varepsilon$ and $(\mathbf{z} \in W \iff \mathbf{x} \in W)$.

We denote by $\hat{\beta}(W)$ is the number of connected components of W which have nonempty interior (called *primary*).

For real $\lambda > 0$, let $B(\lambda, \mathbf{x})$ be the open ball of radius λ centered at $\mathbf{x}$. For $W \subseteq \Re^n$, and $\mathbf{x} \in \Re^n$, we define

$$\tilde{\beta}(W, \mathbf{x}) = \limsup_{\lambda \to 0^+} \beta(W \cap B(\lambda, \mathbf{x}))$$

and

$$\tilde{\beta}(W) = \tilde{\beta}(W, \mathbf{0}),$$

where $\mathbf{0}$ denotes the origin of $\Re^n$.

The following constants appear in our theorems: $c_1 = \frac{1}{1+2\log_2 3}$ and $c_2 = 1 + \frac{\log_2 3}{1+2\log_2 3}$.

3. Lower-bound theorems. In this section we give a list of lower-bound theorems: first, we give results on computation trees, following Ben-Or [2] and Yao [19].

THEOREM 3.1 (Ben-Or). *Let T be an algebraic computation tree that recognizes $W \subseteq \Re^n$. Then*

$$h(T) \geq \log_9 \beta(W) - \frac{1}{2}n.$$

A simple modification of Ben-Or's proof (which we give below) proves the following theorem.

THEOREM 3.1′. *Let T be an algebraic computation tree that recognizes $W \subseteq \Re^n$. Then $h(T) \geq \log_9 \max(\beta(W^\circ), \beta(W)) - \frac{1}{2}n$.*

Theorem 3.1′ gives a stronger result for some sets, where $\beta(W^\circ) > \beta(W)$. (There are sets where it is smaller; this the reason for the max operator.)

THEOREM 3.2 (Yao). *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed. If T is an algebraic computation tree that recognizes W for integer input, then*

$$h(T) \geq c_1(\log_2 \hat{\beta}(W) - 1) - c_2 n.$$

A simple modification of Yao's proof (which we give below) yields the following theorem.

THEOREM 3.2′. *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed. If T is an algebraic computation tree that recognizes W for integer input, then*

$$h(T) \geq c_1(\log_2 \beta(W^\circ) - 1) - c_2 n.$$

This modification always gives a result stronger or equal to the former (see Lemma 5.7 in section 5).

The main results of this paper are theorems for various RAM variants. To clarify the picture, we give a summary of the results in Table 1. In this table, only the leading term of the lower bound is given, without constant coefficient and other details. The table also includes comments on the question, Could it be possible to obtain stronger results by allowing larger sets of addresses, operations, etc.? The precise statement of the lower bound results is given by the theorems below.

THEOREM 3.3. *Let $W \subseteq \Re^n$. If W is recognized in time $t(n)$ on $(\Re, \mathbb{N}, \{+, -, \times, /\})$-RAM, then*

$$t(n) \geq \log_9 \beta(W^\circ) - \frac{1}{2}n\,.$$

The program may specify arbitrary initial values for all memory cells.

THEOREM 3.4. *Let $W \subseteq \Re^n$. If W is recognized in time $t(n)$ on $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$-RAM, then*

$$t(n) \geq \log_9 \beta((\overline{W})^\circ) - \frac{1}{2}n\,.$$

The program may specify arbitrary initial values for all memory cells of integer addresses.

THEOREM 3.5. *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed. If W is recognized for integer input in time $t(n)$ on $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$-RAM, then*

$$t(n) \geq c_1(\log_2 \beta((\overline{W})^\circ) - 1) - c_2 n\,.$$

The program may use a finite initialized table.

THEOREM 3.6. *Let $W \subseteq \Re^n$. If W is recognized in time $t(n)$ on $(\Re, \mathbb{N}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM, then*

$$t(n) \geq \frac{1}{3}(\log_9 \max(\tilde{\beta}(W^\circ), \tilde{\beta}(W)) - 1) - \frac{5}{6}n.$$

The program may specify arbitrary initial values for all memory cells.

THEOREM 3.7. *Let $W \subseteq \Re^n$. If W is recognized in time $t(n)$ on $(\Re, \Re, \{+, -, \times, \lfloor\ \rfloor\})$-RAM, then*

$$t(n) \geq \frac{1}{3}(\log_9 \tilde{\beta}((\overline{W})^\circ) - 1) - \frac{5}{6}n\,.$$

The program may specify arbitrary initial values for all memory cells of integer addresses.

THEOREM 3.8. *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed. If W is recognized for integer input in time $t(n)$ on $(\mathbb{Q}, \mathbb{Q}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM, then*

$$t(n) \geq \frac{1}{2}c_1(\log_2 \beta((\overline{W})^\circ) - 1) - \frac{1}{2}c_2 n\,.$$

The program may use a finite initialized table.

TABLE 1
RAM lower bounds.

Problem space	$\mathcal{D}$	$\mathcal{A}$	$\mathcal{F}$	Preset cells	Lower bound
$\Re^n$	$\Re$	$\mathbb{N}^{(1)}$	$\{+,-,\times,/\}$	$\aleph_0$	$\log \beta(W^\circ)$
$\Re^n$	$\Re$	$\Re$	$\{+,-,\times,/,\chi_{\mathbb{Z}}\}$	$\aleph_0$	$\log \beta((\overline{W})^\circ)$
$\mathbb{Z}^n$	$\Re$	$\Re$	$\{+,-,\times,/,\chi_{\mathbb{Z}}\}$	finite$^{(2)}$	$\log \beta((\overline{W})^\circ)$
$\Re^n$	$\Re$	$\mathbb{N}^{(1)}$	$\{+,-,\times,\lfloor\ \rfloor\}$	$\aleph_0$	$\log \max(\tilde{\beta}(W^\circ), \tilde{\beta}(W))$
$\Re^n$	$\Re$	$\Re$	$\{+,-,\times,\lfloor\ \rfloor\}$	$\aleph_0$	$\log \tilde{\beta}((\overline{W})^\circ)$
$\mathbb{Z}^n$	$\mathbb{Q}^{(3)}$	$\mathbb{Q}$	$\{+,-,\times,\lfloor\ \rfloor\}$	finite$^{(2)}$	$\log \beta((\overline{W})^\circ)$

(1) Cannot allow $\Re$ here, as shown by the element distinctness problem.
(2) Cannot allow $\aleph_0$ here, as illustrated in the introduction.
(3) $\Re$ could possibly be allowed (open problem).

4. Applications. We give some examples of problems for which lower bounds can be proved as corollaries of our theorems. Each problem is presented as a set recognition problem, the set always denoted by W.

In applying Theorems 3.5 and 3.8, we have to make sure that the set in question is rationally dispersed as well as scale invariant. All the sets we consider will be of this type. This will also facilitate the application of Theorems 3.6 and 3.7, where $\tilde{\beta}$ is used instead of β, due to the following simple property.

FACT 4.1. *If W is scale invariant, then $\beta(W) = \tilde{\beta}(W)$.*

Interestingly, all the sets that appear in our list of examples but one belong to a special family, where the desired features can be proved to hold in general.

LEMMA 4.2. *Let W be a subset of $\Re^n$ that is defined by a Boolean combination of linear homogeneous (in)equalities with integer coefficients. Then W is scale invariant and rationally dispersed.*

Proof. According to the assumptions of the lemma, there are homogenous linear forms $l_1, \ldots, l_r$, of integer coefficients, such that W is defined by a Boolean combination of the formulae: $(l_i > 0)$, $(l_i = 0)$, and $(l_i < 0)$, where $1 \le i \le r$.

Homogenity of the l_i implies that replacing $\mathbf{x}$ with $\lambda\mathbf{x}$, for any real $\lambda > 0$, does not change the truth value of any of the (in)equalities. Hence W is scale invariant. We now prove rational dispersal. Let $\mathbf{x} \in W$ and $\varepsilon > 0$. We have to prove the existence of a rational point $\mathbf{z}$ such that $\|\mathbf{z} - \mathbf{x}\| < \varepsilon$ and $\mathbf{z} \in W$. We do not have to treat the case $\mathbf{x} \notin W$ separately, because the complement of W is also a Boolean combination of the same inequalities.

We make the following nonrestrictive assumptions on the formula defining W.

(1) There are no negations. (To achieve this, replace $\neg(l_i = 0)$ by $(l_i < 0) \vee (l_i > 0)$, etc.)

(2) All inequalities are of the form $l_i > 0$. (Otherwise use $-l_i$ instead.)

Now assume that $\mathbf{x}$ is in W because it satisfies the following conditions:

$$l_1 > 0, \ldots, l_r > 0, l_{r+1} = 0, \ldots, l_{r+s} = 0\,.$$

Because of assumption (1), every point that satisfies these conditions will also be in W. Let S be the set defined by the inequalities alone; it is a nonempty, open set. In particular, there is a neighborhood of $\mathbf{x}$, say of radius ε_1, that lies entirely in S.

Next we concentrate on the equalities. These s equations define a nonempty linear subspace L of $\Re^n$, say of dimension k. By elementary linear algebra, we can express this subspace as the graph of a linear function of k *independent* variables. Without

TABLE 2
Applications.

Problem	Computation tree	Weak RAM	Strong RAM
1. Set disjointness	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
2. Sign of resultant	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
3. Element distinctness	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
4. Min gap	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
5. Max gap	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
6. Uniform gap	$\Theta(n \log n)$	$O(n \log n)^{\dagger}$	$\Theta(n)$
7. Sign of permutation	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
8. Convex hull	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
9. Knapsack (KS)	$\Omega(n^2)$	$\Omega(n^2)$	$\Omega(\frac{n^2}{\log n})^{\ddagger}$
10. Generalized knapsack	$\Omega(n^2 \log(k+1))$	$\Omega(n^2 \log(k+1))$	$\Omega(\frac{n^2 \log(k+1)}{\log n + \log\log(k+1)})^{\ddagger}$
11. Approximate gen. KS	$\Omega(n^2 \log(k+1))$	$\Omega(n^2 \log(k+1))$	$\Omega(n^2 \log(k+1))$

†No lower bound from Theorem 3.3, but $\Omega(n \log n)$ from Theorem 3.6.

‡From Meyer auf der Heide's Lemma (section 1).

loss of generality, let these variables be $x_1, \ldots, x_k$; then L is defined by

$$\begin{pmatrix} x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = A \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix},$$

where A is an $(n-k) \times k$ matrix.

Recall that all the l_i have integer coefficients. This implies that the coefficients of A are all rational. Therefore, if we choose rational values $(z_1, \ldots, z_k)$ for the independent variables, multiplying by A to obtain the other coordinates will yield a rational point $\mathbf{z} \in L$. Furthermore, this transformation is obviously continuous, which goes to say that by choosing $(z_1, \ldots, z_k)$ sufficiently close to $(x_1, \ldots, x_k)$, we can make the point $\mathbf{z}$ arbitrarily close to the point $\mathbf{x}$. Since rational points exist arbitrarily close to $(x_1, \ldots, x_k)$, we can make $\mathbf{z}$ rational and satisfy $\|\mathbf{z} - \mathbf{x}\| < \min(\varepsilon, \varepsilon_1)$. This implies that $\mathbf{z} \in S$; hence $\mathbf{z} \in L \cap S \subseteq W$ and satisfies $\|\mathbf{z} - \mathbf{x}\| < \varepsilon$. □

The results described in this section are summarized in Table 2. For every problem we consider, this table gives three complexity results: the complexity in the algebraic computation-tree model and in two groups of RAM model. The "weak" group contains the RAMs with integer addresses that are required to handle real input. For this group the lower bounds are determined by $\log \beta(W^{\circ})$. The "strong" group contains the RAMs with real addresses that are required to handle either real or integer input. For this group, lower bounds are determined by $\log \beta((\overline{W})^{\circ})$. We remark that the upper-bound results do no require the use of noninteger addresses if the input is integer.

EXAMPLE 1 (set disjointness). *Given two sets* $A = \{x_1, \ldots, x_n\}$ *and* $B = \{y_1, \ldots, y_n\}$, *decide whether* A *and* B *are disjoint.*

Here

$$W = \{\, (\mathbf{x}, \mathbf{y}) \in \Re^{2n} : \prod_{1 \le i,j \le n} (x_i - y_j) \neq 0 \,\}.$$

It is open and has $(n!)^2$ components, so an $\Omega(n \log n)$ bound on computation trees follows from Ben-Or's theorem and similarly from Yao's for the case of integer inputs. This is tight, as the problem can be solved by sorting and merging. The same lower bounds follow from Theorems 3.3 and 3.6. On the contrary, in the other models it can be solved in linear time with a straightforward use of indirect addressing. Indeed, $\overline{W} = \Re^{2n}$ (there is a point in W arbitrarily close to every point in $\Re^{2n}$) so $(\overline{W})^\circ$ has a single component.

EXAMPLE 2 (sign of resultant). *Given $x_1, \ldots, x_n; y_1, \ldots, y_n$, decide whether*

$$RES(\mathbf{x}, \mathbf{y}) = \prod_{i,j} (x_i - y_j) > 0.$$

Ben-Or proved an $\Omega(n \log n)$ lower bound for this problem, which also holds in Yao's model. A matching upper bound can be obtained by sorting and merging.

COROLLARY 4.3. *Sign of Resultant requires $\Omega(n \log n)$ time in all the models considered in Theorems* 3.3–3.8.

Proof. Here

$$W = \left\{ (\mathbf{x}, \mathbf{y}) \in \Re^{2n} : \prod_{1 \le i,j \le n} (x_i - y_j) > 0 \right\}.$$

W is clearly open, and

$$\overline{W} = \left\{ (\mathbf{x}, \mathbf{y}) \in \Re^{2n} : \prod_{1 \le i,j \le n} (x_i - y_j) \ge 0 \right\}.$$

We now show that each point where the resultant equals 0 is a boundary point of this set. Indeed, let $(\mathbf{x}; \mathbf{y})$ be such a point; then there are indices k, l such that $x_k = y_l$. Let

$$0 < \delta < \tfrac{1}{4} \min\{|y_j - x_i| : y_j - x_i \neq 0\}$$

and replace each x_i with $x_i + \delta$. This results in a nonzero resultant. If it is negative, we obtained a point outside W. As δ tends to zero, this point gets arbitrarily close to $(\mathbf{x}; \mathbf{y})$; thus $(\mathbf{x}, \mathbf{y})$ is a boundary point. If the resultant is positive, replace x_k with $x_k + \delta$, y_l with $y_l + 2\delta$, and x_i, for all $i \neq k$, with $x_i + 3\delta$. This inverts the sign of exactly one of the differences $x_i - y_j$, namely the difference $x_k - y_l$. We obtain a point where the resultant is negative, which can be arbitrarily close to $(\mathbf{x}; \mathbf{y})$.

Removing the boundary points, we obtain $(\overline{W})^\circ = W$. The proof is completed by showing that this set has at least $(n-1)!$ connected components.

Let $\mathbf{x} = (0, 2, \ldots, 2n-2)$ and $\mathbf{y} = (1, 3, \ldots, 2n-1)$. If $RES(\mathbf{x}, \mathbf{y}) > 0$, set x_n to $2n$; this inverts the sign of just one difference $(x_n - y_n)$ and hence the sign of $RES(\mathbf{x}, \mathbf{y})$. Now, replacing $(y_1, \ldots, y_{n-1})$ by an arbitrary permutation of these values will preserve the sign of $RES(\mathbf{x}, \mathbf{y})$. We thus obtain $(n-1)!$ distinct points in W. Let $\mathbf{u}$ and $\mathbf{v}$ be two such points. Say they differ in the value of y_j for some $1 \le j \le n-1$. Then there is an i such that $x_i - y_j$ is positive at one of the points (say $\mathbf{u}$) and negative at the other ($\mathbf{v}$). On any path connecting $\mathbf{u}$ and $\mathbf{v}$ there must be a point where $x_i - y_j = 0$; this point does not belong to W. Hence $\mathbf{u}$ and $\mathbf{v}$ belong to distinct connected components, and $\beta(W) \ge (n-1)!$. □

EXAMPLE 3 (element distinctness). *Given $x_1, x_2, \ldots, x_n$, decide whether these n numbers are all distinct.*

EXAMPLE 4 (min gap). *Given $n+1$ numbers $x_1, \ldots, x_n, t$, decide if the minimum difference between a pair of these numbers is bounded above by t.*

Frequently this problem is presented in a computational form, where the minimum gap has to be reported; the above decision problem is obviously not harder, and for it we give a lower bound that matches an upper bound known for the computational problem.

Yao's theorem gives an $\Omega(n \log n)$ lower bound for Element Distinctness on computation trees, even when restricted to integer inputs. Since Element Distinctness is a special case of Min Gap ($t = 0$), the same lower bound applies to Min Gap. As there is a matching upper bound (by sorting the numbers), the complexity of both problems for the algebraic computation tree is $\Theta(n \log n)$. The lower bound for Min Gap holds for all our RAM models as well as we will shortly demonstrate.

The situation for Element Distinctness is different: We show that the $\Omega(n \log n)$ lower bound holds in the "weak" models. The "strong" models can solve the problem in linear time, with a straightforward use of indirect addressing. Thus the latter models separate the complexities of the two problems.

COROLLARY 4.4. *Min Gap requires $\Omega(n \log n)$ time in all the models considered in Theorems* 3.3–3.8. *Element Distinctness requires $\Omega(n \log n)$ time in the models with integer addresses.*

Proof. The corollary follows by showing that for Element Distinctness, W° has $n!$ components, while for Min Gap, the same is true for $(\overline{W})^\circ$.

For Element Distinctness we have

$$W = \{\, (\mathbf{x} \in \Re^n : (\forall\, 1 \le i < j \le n)\ x_i \ne x_j \,\}.$$

This set is clearly open, so $W^\circ = W$. To see that the number of components of W is at least $n!$, consider points $\mathbf{x}$ such that $(x_1, \ldots, x_n)$ is a permutation of $(1, 2, \ldots, n)$; $\mathbf{x} \in W$, and there are $n!$ such points. We show that each of them lies in a distinct component. Let $\mathbf{u}$, $\mathbf{v}$ be distinct points from this set. Let $i_1, i_2, \ldots, i_n$ be such that $u_{i_k} = k$. Since $\mathbf{v}$ is a different permutation, there will be a first k such that $v_{i_k} \ne k$. Necessarily $v_{i_k} > k$. For another position i_l, $l \ne k$, we will have $v_{i_l} = k$. Consider the function $f(\mathbf{x}) = x_{i_l} - x_{i_k}$. We have $f(\mathbf{u}) > 0$ and $f(\mathbf{v}) < 0$ and f is continuous, therefore every path connecting $\mathbf{u}$ and $\mathbf{v}$ must contain a point where $x_{i_l} = x_{i_k}$, a point not in W.

In contrast, $\overline{W} = \Re^n$, so $(\overline{W})^\circ$ has a single component. No wonder that the lower bound collapses for the strong models.

Consider now the Min Gap problem. We have

$$W = \{\, (\mathbf{x}, t) \in \Re^{n+1} : (\exists\, 1 \le i < j \le n)\, |x_i - x_j| \le t \,\}.$$

Clearly, recognizing W has the same complexity as recognizing its complement W', and adding the requirement $t \ge 0$ does not change the complexity of recognizing the set. Clearly,

$$W' = \{\, (\mathbf{x}, t) \in \Re^n : (\forall\, 1 \le i < j \le n)\, |x_i - x_j| > t \,\}$$

and

$$\overline{W' \cap \{t \ge 0\}} = \{\, (\mathbf{x}, t) \in \Re^n : t \ge 0, (\forall\, 1 \le i < j \le n)\, |x_i - x_j| \ge t \,\}.$$

Now, if $t > 0$, the points that satisfy one of these inequalities in the weak sense are boundary points. The points where $t = 0$ are all boundary points. Hence

$$(\overline{W' \cap \{t \geq 0\}})^\circ = \{\, (\mathbf{x}, t) \in \Re^n : t > 0, (\forall\, 1 \leq i < j \leq n)\, |x_i - x_j| > t \,\} = W'.$$

This set has $n!$ components. (Let $\mathbf{x}$ be a permutation of $(2t, 4t, \ldots, 2nt)$; each permutation lies in a distinct component.) □

EXAMPLE 5 (max gap). *Given $n+1$ numbers $x_1, \ldots, x_n, \varepsilon$, decide whether the x_i can be permuted into a nondecreasing sequence such that the gap between each pair of consecutive elements is bounded above by ε.*

Again, frequently we are asked to find out the maximum gap; this decision problem is not harder, and the lower bound that we give matches the upper bound for the computational version.

The theorems of Ben-Or and Yao yield lower bounds of $\Omega(n \log n)$ for Max Gap on algebraic computation trees (resp., with integer input). Clearly, Max Gap can be solved in $O(n \log n)$ time by sorting the numbers. The same lower bound holds for RAMs.

COROLLARY 4.5. *Max Gap requires $\Omega(n \log n)$ time in all the models considered in Theorems* 3.3–3.8.

Proof. We consider a restricted problem in which we have to decide whether the following conditions hold simultaneously: (1) the maximum gap is less than ε; (2) the difference between the maximum and minimum of this set is greater than $(n - 1.5)\varepsilon$. (Checking the last condition obviously takes linear time.) Let S_n be the set of permutations over $\{1, 2, \ldots, n\}$. In this problem,

$$W = \{\, (\mathbf{x}, \varepsilon) \in \Re^{n+1} : (\exists \sigma \in S_n)\, x_{\sigma(n)} - x_{\sigma(1)} > (n - 1.5)\varepsilon \\ \wedge\, (\forall j < n)\, x_{\sigma(j)} \leq x_{\sigma(j+1)} < x_{\sigma(j)} + \varepsilon \,\}.$$

Clearly,

$$\overline{W} = \{\, (\mathbf{x}, \varepsilon) \in \Re^{n+1} : (\exists \sigma \in S_n)\, x_{\sigma(n)} - x_{\sigma(1)} \geq (n - 1.5)\varepsilon \\ \wedge\, (\forall j < n)\, x_{\sigma(j)} \leq x_{\sigma(j+1)} \leq x_{\sigma(j)} + \varepsilon \,\}.$$

Each point of this set such that $\varepsilon = 0$ is a boundary point, since ε cannot be negative. For $\varepsilon > 0$, we show now that the points where at least one of the inequalities holds weakly are boundary points. Let $(\mathbf{x}, \varepsilon)$ be such a point. We show that there are points, arbitrarily close to it, not in $\overline{W}$. For simplicity, assume that the appropriate σ for this point is the identity, i.e., $x_1 \leq x_2 \leq \cdots \leq x_n$. First note that the condition $x_j = x_{j+1}$ cannot hold for any j since otherwise, $x_n - x_1 \geq (n - 1.5)\varepsilon$ implies that at least one gap exceeds ε. If $x_n - x_1 = (n - 1.5)\varepsilon$, then for all small enough $\delta > 0$, $(x_1, \ldots, x_n - \delta)$ is not in $\overline{W}$. Next, if $x_{j+1} = x_j + \varepsilon$, for all small enough $\delta > 0$ the point $(x_1, \ldots, x_j, x_{j+1} + \delta, x_{j+2}, \ldots, x_n)$ is not in $\overline{W}$.

We conclude that

$$(\overline{W})^\circ = \{\, (\mathbf{x}, \varepsilon) \in \Re^{n+1} : (\exists \sigma \in S_n)\, x_{\sigma(n)} - x_{\sigma(1)} > (n - 1.5)\varepsilon \\ \wedge (\forall j < n)\, x_{\sigma(j)} < x_{\sigma(j+1)} < x_{\sigma(j)} + \varepsilon \,\}.$$

The proof is completed by observing that this set has $n!$ components. (Each permutation of $(\varepsilon, 2\varepsilon, \ldots, n\varepsilon)$ belongs to a distinct component.) □

EXAMPLE 6 (uniform gap). *Given $n+1$ numbers $x_1, x_2, \ldots, x_n, \varepsilon$ decide whether the x_i can be permuted into a nondecreasing sequence where the gap between each pair of consecutive elements is ε.*

Here

$$W = \{\, (\mathbf{x}, \varepsilon) \in \Re^{n+1} : (\exists \sigma \in S_n)\, (\forall j < n)\, x_{\sigma(j+1)} = x_{\sigma(j)} + \varepsilon \,\}.$$

W has $n!$ components (given by permutations of $(\varepsilon, 2\varepsilon, \ldots, n\varepsilon)$), while W° is empty. Thus, Ben-Or's theorem implies an $\Omega(n \log n)$ bound for an algebraic computation tree solving Uniform Gap on real input, while Yao's theorem fails to provide a lower bound. The same happens with all our theorems but one.

COROLLARY 4.6. *Uniform Gap requires* $\Omega(n \log n)$ *time on a* $(\Re, \mathbb{N}, \{+, -, \times, \lfloor\ \rfloor\})$*-RAM.*

The result follows from the appearance of $\log \tilde{\beta}(W)$ in Theorem 3.6. We conjecture that the same is true for the model of Theorem 3.3, but our theorem does not prove it as it only refers to $\beta(W^\circ)$. In contrast, in the four "strong" models, the problem can be solved in linear time: Compute the minimum of the $\{x_i\}$, m. Then by means of indirect addressing check whether the rest of the set equals $\{m+\varepsilon, m+2\varepsilon, \ldots, m+(n-1)\varepsilon\}$.

Uniform Gap relates to Max Gap in a way that resembles the relationship of Element Distinctness to Min Gap. On one hand, a nonlinear lower bound for Uniform Gap implies the same lower bound for Max Gap, because Uniform Gap can be efficiently reduced to the latter. In fact, the answer to Uniform Gap is "yes" if and only if the answer to Max Gap on $x_1, \ldots, x_n$ is ε, and the difference between the minimum and the maximum of this set is $(n-1)\varepsilon$. Thus, when there is an $\Omega(n \log n)$ bound for Uniform Gap, the complexity of both problems coincides, and this happens for algebraic computation trees, over real input, and for the model of the above corollary. On the contrary, as with Element Distinctess and Min Gap, the "strong" models separate the complexities of the two problems.

EXAMPLE 7 (sign of permutation). *Given* $x_1, \ldots, x_n$ *decide whether there exists an even permutation* σ *such that* $x_{\sigma(1)} < x_{\sigma(2)} < \cdots < x_{\sigma(n)}$.

Here

$$W = \left\{ \mathbf{x} \in \Re^n : \prod_{i<j} (x_j - x_i) > 0 \right\}.$$

Yao showed that this problem requires $\Omega(n \log n)$ in an algebraic computation tree, even for integer input, and we show that the same holds for the RAM models.

COROLLARY 4.7. *Sign of permutation requires* $\Omega(n \log n)$ *time in all the models considered in Theorems* 3.3–3.8.

Proof. We have

$$\overline{W} = \left\{ \mathbf{x} \in \Re^n : \prod_{i<j} (x_j - x_i) \geq 0 \right\}.$$

We next prove that each point where equality holds is a boundary point of this set. Therefore, $(\overline{W})^\circ = W$ and has $\frac{n!}{2}$ components. (Each even permutation of $(1, 2, \ldots, n)$ represents a distinct component.) To complete the proof, let $(x_1, \ldots, x_n)$ be a point such that $\prod_{i<j}(x_j - x_i) = 0$; thus the coordinates are not all distinct. Without loss of generality, assume $x_1 \leq x_2 \leq \cdots \leq x_n$ and let k satisfy $x_k = x_{k+1}$. For every $\delta > 0$, consider the point

$$\mathbf{y}_\delta = (x_1 + \delta, \ldots, x_{k-1} + (k-1)\delta, x_k + (k+1)\delta, x_{k+1} + k\delta, x_{k+2} + (k+2)\delta, \ldots, x_n + n\delta)\,.$$

The difference $y_j - y_i$ is positive for all $j > i$ except $i = k, j = k+1$, where it is negative. Therefore, $\prod_{i<j}(y_j - y_i) < 0$ and $\mathbf{y} \notin \overline{W}$. However, when $\delta \to 0$, $\mathbf{y} \to \mathbf{x}$. Thus $\mathbf{x}$ is a boundary point of $\overline{W}$. □

EXAMPLE 8 (convex hull). *Given n points in the plane, decide whether they all lie on the convex hull of the set.*

Yao considered a slightly different version of this problem, where we have to decide whether the convex hull has n vertices (which means that all the points lie on the hull and are also in general position). Yao gave an $\Omega(n \log n)$ lower bound for a computation tree that solves this problem, even for integer inputs. The same lower bound for our version of the problem follows by Theorem 3.2′. In this case $W \subseteq \Re^{2n}$ is the set of all points $(x_0, y_0, \ldots, x_{n-1}, y_{n-1})$ for which there exists a permutation σ on $(0, 1, \ldots, n-1)$ such that, for all $0 \le i < n$,

$$\begin{vmatrix} 1 & x_{\sigma(i)} & y_{\sigma(i)} \\ 1 & x_{\sigma(i+1) \bmod n} & y_{\sigma(i+1) \bmod n} \\ 1 & x_{\sigma(i+2) \bmod n} & y_{\sigma(i+2) \bmod n} \end{vmatrix} \ge 0.$$

W° has $(n-1)!$ components and is scale invariant and rationally dispersed [17, 19]. W is closed, so the same holds for $(\overline{W})^\circ$, and the lower bound applies to the various RAM models as well.

EXAMPLE 9 (knapsack). *Given numbers $x_1, \ldots, x_n$ and $M > 0$ decide if there exists some subset $S \subseteq \{1, 2, \ldots, n\}$ such that*

$$\sum_{i \in S} x_i = M.$$

EXAMPLE 10 (generalized knapsack). *Given numbers $x_1, \ldots, x_n$ and $M > 0$ decide if there exists a vector $v \in \{0 \ldots k\}^n$ such that the inner product $x \cdot v = M$. (For $k = 1$ this is the original problem.)*

EXAMPLE 11 (approximate generalized knapsack). *Given numbers $x_1, \ldots, x_n$ and $M > 0$ decide if there exists a vector $v \in \{0 \ldots k\}^n$ such that*

$$|x \cdot v - M| \le \varepsilon M,$$

where $k \in \mathbb{N}$, $\varepsilon > 0$ are fixed constants.

Let $\mathcal{P}$ denote the spatial partition induced on $\Re^{n+1}$ by the hyperplanes $v \cdot x = M$, for all $v \in \{0 \ldots k\}^n$. Meyer auf der Heide [12] shows that $\mathcal{P}$ consists of at least $(k+1)^{(n(n-1)/2)}/(2^n - 1)$ connected regions. He uses this fact to give a lower bound on computation-tree complexity for the generalized knapsack problem. (For the original knapsack problem, lower bounds are given in [2, 19].) We start with the approximate problem, for which we obtain a lower bound in all of the RAM models.

COROLLARY 4.8. *On all the models considered in Theorems 3.3–3.8, the complexity of the approximate knapsack problem is $\Omega(n^2 \log(k+1))$.*

Proof. Let W' be the complement of the set to be recognized. A lower bound for recognizing W' is a lower bound for recognizing W.

$$W' = \{(x_1, \ldots, x_n, M) : |v \cdot x - M| > \varepsilon M \quad \forall v \in \{0 \ldots k\}^n\}.$$

It is easy to show that $(\overline{W'})^\circ = W'$, and that if ε is small enough, then a component of W' exists within each region of $\mathcal{P}$. Thus, $\beta((\overline{W'})^\circ) \ge (k+1)^{(n(n-1)/2)}/(2^n - 1)$ and the lower bound follows. □

The exact problems differ from the approximate ones by setting $\varepsilon = 0$. This change invalidates the results using $(\overline{W'})^\circ$, because it becomes $\Re^n$. However, $(W')^\circ = W'$, so the above lower bound stays valid for the RAMs with integer addresses. We remark that for the stronger models, it is possible to obtain (weaker) lower bounds using a technique Meyer auf der Heide's lemma (section 1). The results are $\Omega(n^2/\log n)$ for Exact Knapsack and $\Omega(n^2 \log(k+1)/(\log n + \log\log(k+1)))$ for Exact Generalized Knapsack.

5. Background for Proofs. In this section we group a few definitions, facts and lemmas that are useful for the forthcoming proofs. We start with some point-set topology.

FACT 5.1. *Every connected component of an open set is open.*

FACT 5.2. *If $U \subseteq V$, then every connected component of U is contained within a component of V.*

LEMMA 5.3. *Let W, A be subsets of some metric space such that A is closed and has empty interior. Then*

$$(\overline{W \setminus A})^\circ = (\overline{W})^\circ = (\overline{W \cup A})^\circ.$$

Proof. The first equality follows from the second by substituting $W \setminus A$ for W. We now prove the second one. We obviously have $\overline{W} \subseteq \overline{W \cup A}$. Therefore also $(\overline{W})^\circ \subseteq (\overline{W \cup A})^\circ$. To prove inclusion in the other direction, note that

$$\overline{(W \cup A)} = \overline{W} \cup \overline{A} = \overline{W} \cup A\,.$$

Let $x \in (\overline{W \cup A})^\circ = (\overline{W} \cup A)^\circ$. Since $x \in (\overline{W} \cup A)^\circ$, there exists a ball $B \subseteq \overline{W} \cup A$ that includes x. We distinguish two cases.

Case 1: $x \notin A$. A is closed, so $B \setminus A$ is an open set, contains x, and is contained in $\overline{W}$. This shows that $x \in (\overline{W})^\circ$.

Case 2: $x \in A$. Since A has empty interior, $B \subseteq \overline{B \setminus A}$. Since B is open, this implies $B \subseteq (\overline{B \setminus A})^\circ$. Thus $x \in (\overline{B \setminus A})^\circ$. But $B \setminus A \subseteq \overline{W}$, so $(\overline{B \setminus A})^\circ \subseteq (\overline{W})^\circ$. □

LEMMA 5.4. *Let $X \subseteq \Re^n$, B an open ball in $\Re^n$. Then*

$$(\overline{X})^\circ \cap B = (\overline{X \cap B})^\circ.$$

Proof. Because B is open, we have

$$\overline{X} \cap B \subseteq \overline{X \cap B}.$$

Hence,

$$(\overline{X})^\circ \cap B = (\overline{X})^\circ \cap B^\circ = (\overline{X} \cap B)^\circ \subseteq (\overline{X \cap B})^\circ.$$

On the other hand, because B is a ball, $(\overline{B})^\circ = B$; hence

$$(\overline{X \cap B})^\circ \subseteq (\overline{X} \cap \overline{B})^\circ = (\overline{X})^\circ \cap (\overline{B})^\circ = (\overline{X})^\circ \cap B.$$

Combining the two inclusions, we have

$$(\overline{X})^\circ \cap B = (\overline{X \cap B})^\circ\,. \qquad \square$$

LEMMA 5.5. *Let W be an open set. Then*

$$\beta((\overline{W})^\circ) \leq \beta(W).$$

Proof. We show that each component of $(\overline{W})^\circ$ contains at least one component of W. In fact, since $W \subseteq \overline{W}$, we also have $W \subseteq (\overline{W})^\circ$. Thus a component of W must be contained within a component of $(\overline{W})^\circ$. It remains to show that each component C of $(\overline{W})^\circ$ contains at least one point of W. To this end, observe that $\overline{W} = W \cup \partial W$, so $W \supseteq \overline{W} \setminus \partial W$; hence

$$C \cap W \supseteq C \cap (\overline{W} \setminus \partial W) = (C \cap \overline{W}) \setminus (C \cap \partial W) = C \setminus \partial W$$

and this is not empty since a boundary set cannot contain C. □

LEMMA 5.6. *Let W be an open set and A a boundary set. Then*

$$\beta(W \setminus A) \geq \beta(W).$$

Proof. We show that each component of W contains at least one component of $W \setminus A$. In fact, since $W \setminus A \subseteq W$, every component of $W \setminus A$ must be contained within a component of W. It remains to show that for each component C of W contains at least one point of $W \setminus A$. This follows since otherwise, we would have $C \subseteq A$, but C is open and A is a boundary set, a contradiction. □

LEMMA 5.7. *For any set W, $\hat{\beta}(W) \leq \beta(W^\circ)$.*

Proof. Recall that $\hat{\beta}(W)$ is the number of components of W that have a nonempty interior. Let C be such a component; we show that it contains at least one component of W°. This establishes the desired inequality. In fact, it suffices to prove that $C \cap W^\circ$ is not empty, because once C contains one point of W°, it will contain the whole component of this point (Fact 5.2).

Now, $W^\circ = W \setminus \partial W$. Therefore $C \cap W^\circ = C \setminus \partial W$. Since C has a nonempty interior, it cannot be contained in ∂W; hence the last set is not empty. □

We now move on to properties of polynomials and rational functions on $\Re^n$. We will be particularly interested in the existence of integer points on the graphs of such functions.

DEFINITION 5.8. *An algebraic variety in $\Re^n$ is the set of common zeros of a finite set of polynomials; in other words, it is the set of points satisfying a finite set of algebraic equations.*

The adjective *algebraic* is often omitted, since we do not consider other types of "varieties." A variety is called *nontrivial* if it is different from $\Re^n$. Here are some important properties.

FACT 5.9. *Every finite union or intersection of algebraic varieties is itself a variety. Every nontrivial algebraic variety is a closed boundary set.*

DEFINITION 5.10. *A function of the form P/Q, where P and Q are (multivariate) polynomials, is called rational.*

LEMMA 5.11. *Any function that can be computed from $x_1, \ldots, x_n$ using the operations $\{+, -, \times, /\}$ is rational.*

Proof. The coordinate functions $I_k(\mathbf{x}) = x_k$ are rational. The result follows by induction on the number of operations, using the following identities:

$$\begin{aligned} \tfrac{P}{Q} \pm \tfrac{R}{T} &= \tfrac{PT \pm RQ}{QT}, \\ \tfrac{P}{Q} \cdot \tfrac{R}{T} &= \tfrac{PR}{QT}, \\ \tfrac{P}{Q} \Big/ \tfrac{R}{T} &= \tfrac{PT}{QR}. \end{aligned}$$ □

Standard notation is $\Re[\mathbf{x}]$ for the ring of polynomials in $\mathbf{x}$ with coefficients from $\Re$, and $\Re(\mathbf{x})$ for the corresponding field of rational functions.

LEMMA 5.12. *Let $f : \Re^n \to \Re$ be rational. Assume that there is an open nonempty subset of $\Re^n$ on which f only assumes integer values. Then f is constant throughout its domain of definition D_f.*

Proof. Note first that f is continuous on every path-connected component of D_f. Assume that $O \subseteq \Re^n$ is an open subset as described. Let $\mathbf{x}_0 \in O$, and $y = f(\mathbf{x}_0)$. By the assumption, $y \in \mathbb{Z}$. Let $O_1 \subseteq O$ be the path-connected component of O that contains $\mathbf{x}_0$. By Fact 5.1, O_1 is an open set. If for some $\mathbf{x}_1 \in O_1$, $f(\mathbf{x}_1) \neq f(\mathbf{x}_0)$, it follows from continuity of f that there is $\mathbf{x}_2 \in O$ such that $f(\mathbf{x}_2) \notin \mathbb{Z}$, contradicting the assumption. Therefore, $f \equiv f(\mathbf{x}_0)$ throughout O_1.

Express f as P/Q where P,Q are polynomials. We obtain that for all $\mathbf{x} \in O_1$,

$$P(\mathbf{x})/Q(\mathbf{x}) = f(\mathbf{x}_0), \tag{1}$$

i.e.,

$$P(\mathbf{x}) - f(\mathbf{x}_0)Q(\mathbf{x}) = 0\,. \tag{2}$$

In the last equation, the left-hand side is a polynomial; a polynomial that is zero on an open set must be the zero polynomial. Therefore, (1) is an identity, and the same goes for (2) (within D_f). □

DEFINITION 5.13. *A set $H \subseteq \Re^n$ is a bale if there is a finite set of polynomials $\{ P_i, Q_i,\ i \in I \}$ such that*

$$H = \bigcup_{i \in I} \left\{ \mathbf{x} \in \Re^n : \left(\frac{P_i(\mathbf{x})}{Q_i(\mathbf{x})} \in \mathbb{Z} \right) \vee (Q_i(\mathbf{x}) = 0) \right\}.$$

The above bale is denoted by $\mathcal{B}(P_i, Q_i, i \in I)$. If there is only one pair of polynomials P, Q we write $\mathcal{B}(P, Q)$. A bale is called *nontrivial* if it is different from $\Re^n$. The following observations are important for our proofs.

FACT 5.14. *Every algebraic variety in $\Re^n$ is a bale. A finite union of bales is a bale. Every nontrivial bale is a closed boundary set.*

Proof. For the first claim, note that an algebraic variety in $\Re^n$ can be described as the zero set of a single polynomial (the sum of squares of the defining polynomials). Call this polynomial Q. The zero-set of Q is exactly the bale $\mathcal{B}(Q/2, Q)$. The second claim is immediate from the definition. For the third, let $H = \mathcal{B}(P, Q)$. It is enough to prove the claim for such a bale because a finite union of closed boundary sets is a closed boundary set. H is, by definition, a countable union of varieties (namely, $\{Q_i(\mathbf{x}) = 0\}$, and $\{P_i(\mathbf{x}) = kQ_i(\mathbf{x})\}$ for all $k \in \mathbb{Z}$). Each of them is a boundary set so, by Baire's category theorem, so is their union. To show that H is closed, we prove that $\Re^n \setminus H$ is open: let $\mathbf{y} \in \Re^n \setminus H$. Then $Q(\mathbf{y}) \neq 0$. By continuity of Q, there is a neighborhood $\mathcal{N}$ of $\mathbf{y}$ in which $Q \neq 0$. Also $\frac{P(\mathbf{y})}{Q(\mathbf{y})} \notin \mathbb{Z}$. By continuity of P/Q in $\mathcal{N}$, there is a neighborhood of $\mathbf{y}$ in which $\frac{P(\mathbf{y})}{Q(\mathbf{y})} \notin \mathbb{Z}$. Thus there exists a neighborhood of $\mathbf{y}$ contained in $\Re^n \setminus H$. □

DEFINITION 5.15. *For two functions f, g we write $f \prec g$ if f is defined at least where g is and $f_{|domg} = g$.*

DEFINITION 5.16. *A subset of $\Re^n$ is called large if it is not contained in any nontrivial algebraic variety.*

Note that a large set is necessarily infinite, since every finite set of points is a nontrivial variety. Furthermore, every finite union of nontrivial varieties is a nontrivial variety; hence no large set can be covered by finitely many nontrivial varieties.

THEOREM 5.17. *Let $r \in \Re(\mathbf{x})$ be a rational function. If there is a large set $B \subseteq \mathbb{Z}^n$ such that $r(B) \subseteq \mathbb{Z}$, then there are $P, Q \in \mathbb{Z}[\mathbf{x}]$ such that $P/Q \prec r$.*

Informally, if a rational function (with coefficients from $\Re$) has an integer value in "many" integer points, then the coefficients may be safely assumed to be from $\mathbb{Z}$.

Proof. Every rational function has a *reduced* form, in which the nominator and denominator are relatively prime polynomials. Every other form has larger degree in both polynomials. The reduced denominator is a divisor of any other possible denominator, and hence the reduced fraction will be defined whenever the nonreduced form is as follows: symbolically, if R_1/R_2 is the reduced form of r, then $R_1/R_2 \prec r$. Let $d_i = \deg R_i$. Then R_1/R_2 is completely specified by $d = d_1+d_2+2$ real coefficients, $(a_1, \ldots, a_d)$. The relation $r(\mathbf{x}) = y$ can be rewritten as $R_1(\mathbf{x}) = yR_2(\mathbf{x})$, an equation linear in $(a_1, \ldots, a_d)$. Every point of B gives rise to such an equation; we obtain an infinite set E of linear equations which have a common solution, hence a set of solutions which is a linear subspace of $\Re^d$. Let e be the dimension of this subspace; then we may select $d - e$ equations from E that suffice to determine the required subspace.

The equations in question have integer coefficients; therefore, they can be solved over the rationals. We obtain a rational vector of coefficients $(a_1, \ldots, a_d)$. Multiplying by a common denominator we can assume the a_i to be integer. (Note that the equations are homogeneous.) We thus obtain a function P/Q, with $P, Q \in \mathbb{Z}[\mathbf{x}]$, that coincides with R on B. Thus, $R_1Q - R_2P = 0$ on B, but by the assumption on B, the polynomial $R_1Q - R_2P$ must be the zero polynomial. Hence $P/Q = R_1/R_2$ wherever both are defined. Moreover, since R_1/R_2 is reduced, and P, Q are constrained by the same degrees as R_1, R_2, we must have $(P, Q) = c(R_1, R_2)$ for a scalar c. Thus P/Q coincides precisely with R_1/R_2. □

In what follows, we use the letter t to denote a scalar variable (in contrast with the boldface letters, such as $\mathbf{x}$, that designate vector variables). By $\mathbb{Z}^+$ we denote, as usual, the set of positive integers.

LEMMA 5.18. *Let $P \in \mathbb{Q}[t]$. Then*

(i) *there is a rational $0 < q \leq 1$ such that $P(\mathbb{Z}) \subseteq q\mathbb{Z}$;*

(ii) *there is an integer $a > 0$ such that either $P(a\mathbb{Z}) \subseteq \mathbb{Z}$ or $P(a\mathbb{Z}) \cap \mathbb{Z} = \emptyset$.*

Proof. Assume $P \not\equiv 0$ (otherwise the statements are trivial). The nonzero coefficients of P are all rational so they have a common denominator $d > 0$. We choose $q = 1/d$ and $a = d$. We leave it to the reader to verify the claims. □

DEFINITION 5.19. *A ray in $\Re^n$ is a set of the form $\{t\mathbf{c} : t > 0\}$ for $\mathbf{c} = (c_1, c_2, \ldots, c_n) \neq \mathbf{0}$.*

The vector $(c_1, \ldots, c_n)$ is *a coordinate vector* for the ray. In fact, every point on the ray can serve as coordinate vector.

Let $f : \Re^n \to \Re^n$, and let K be a ray. We denote by $f_{|K}$ the function of a single variable obtained by restricting f to K; thus $f_{|K}(t) = f(t\mathbf{c})$, where $\mathbf{c}$ is a chosen coordinate vector for K. When we want to be specific about the choice of $\mathbf{c}$, we write $f_{\mathbf{c}}(t)$ for $f(t\mathbf{c})$. It is easy to see that if f is a polynomial, so is $f_{\mathbf{c}}$ for any $\mathbf{c} \in K$.

The following is almost too obvious to state.

FACT 5.20. *A set in $\Re^n$ is scale invariant if and only if it is a union of rays (plus possibly the origin).*

LEMMA 5.21. *Let H be an algebraic variety in $\Re^n$ and $K \subseteq \Re^n$ a ray. Then $H \cap K$ either equals K or is finite.*

Proof. Let $\mathbf{c}$ be a coordinate vector for K; substitute $\mathbf{x} = t\mathbf{c}$ in all the polynomials defining H. We obtain a set of monovariate polynomials that define $H \cap K$. However,

a monovariate polynomial either is identically zero or has finitely many zeros. The lemma follows. □

A ray is called *rational* if it has a rational coordinate vector (which amounts to saying that it contains a rational point). We denote the set of rational rays in $\Re^n$ by $\mathcal{Q}^{(n-1)}$. (The superscript reminds us that this set is an $(n-1)$-dimensional vector space, but this is not important for our discussion.) Subsets of $\mathcal{Q}^{(n-1)}$ are often denoted by Γ.

LEMMA 5.22. *Let $\Gamma \subseteq \mathcal{Q}^{(n-1)}$, and for all $K \in \Gamma$ let $S(K) \subseteq K \cap \mathbb{Z}^n$ be infinite. If $\bigcup_{K\in\Gamma} K$ is large, then $\bigcup_{K\in\Gamma} S(K)$ is also large.*

Proof. We prove the contrapositive: Assume that there is a nontrivial variety H such that $\bigcup_{K\in\Gamma} S(K) \subseteq H$. Thus, every ray $K \in \Gamma$ intersects with H infinitely often. By Lemma 5.21, $K \subseteq H$. Hence $\bigcup_{K\in\Gamma} K \subseteq H$. □

DEFINITION 5.23. *The function $f : \Re \to \Re$ is periodically integral (p.i.) if there is $a \in \mathbb{Z}^+$ such that $f(a\mathbb{Z}^+) \subseteq \mathbb{Z}$. We say that f is periodically nonintegral (p.n.i.) if there $a \in \mathbb{Z}^+$, such that $f(a\mathbb{Z}^+) \subseteq \Re \setminus \mathbb{Z}$.*

LEMMA 5.24. *If f is not periodically (non)integral, then there is an infinite set $S \subseteq \mathbb{Z}^+$ such that $f(S)$ lies outside (inside) of $\mathbb{Z}$.*

We omit the easy proof. It is also easy to see that, if $f(t)$ is p.i. (p.n.i.), then $f(qt)$ is p.i. (p.n.i.) for any $q \in \mathbb{Q}^+$. The following is an immediate consequence.

LEMMA 5.25. *Let $K \in \mathcal{Q}^{(n-1)}$ and $f : \Re^n \to \Re$. Then $f_{|\mathbf{y}}$ is p.i. (p.n.i.) for some $\mathbf{y} \in K \cap \mathbb{Q}^n$ if and only if it is p.i. (p.n.i.) for all such $\mathbf{y}$.*

In view of this lemma, we can use the expression "$f_{|K}$ is p.i. (p.n.i)" without specifying the coordinates used for K. Another easy observation is that a function may be either p.i. or p.n.i. but not both. There are also functions that are neither; as the next lemma shows, this does not occur for polynomials from $\mathbb{Q}[t]$.

LEMMA 5.26. *A polynomial in $\mathbb{Q}[t]$ is p.i. if and only if its constant term is an integer. It is p.n.i. otherwise.*

Proof. Let $Q(t) = c_0 + c_1 t + \cdots + c_d t^d \in \mathbb{Q}[t]$. Assume first that c_0 is an integer; let d be the common denominator of $c_1, \ldots, c_d$. Then $Q(d\mathbb{Z}^+) \subseteq \mathbb{Z}$. On the other hand, if $c_0 \notin \mathbb{Z}$, then $Q(dm) \notin \mathbb{Z}$ for all $m \in \mathbb{Z}$. □

DEFINITION 5.27. *Let $S \subseteq \Re$ and $x \in \Re$. The distance $d(x, S)$ is defined as* $\inf_{s\in S} |x - s|$.

We are now prepared to prove a useful theorem about rational functions and their behavior on rational rays.

THEOREM 5.28. *For $r \in \Re(\mathbf{x})$ let Γ (resp., Δ) be the set of rays $K \in \mathcal{Q}^{(n-1)}$ such that $r_{|K}$ is periodically (non)integral. Then either $\bigcup_{K\notin\Gamma} K$ or $\bigcup_{K\notin\Delta} K$ is contained in a nontrivial, scale-invariant bale.*

Proof. Fix r, and hence Γ and Δ, as in the theorem. To prove the theorem, we assume

(∗) $\bigcup_{K\notin\Delta} K$ is not contained in any nontrivial, scale-invariant bale.

We will prove that $\bigcup_{K\notin\Gamma} K$ is contained in a nontrivial scale-invariant variety.

For every $K \notin \Delta$, let $S(K)$ be the following set, implied by Lemma 5.24: $S(K) \subseteq K \cap \mathbb{Z}^n$, it is infinite, and $f(\mathbf{x}) \in \mathbb{Z}$ for all $\mathbf{x} \in S(K)$.

From our assumption (∗) and Lemma 5.22, $\bigcup_{K\notin\Delta}(S(K))$ is a large set. Note that this is a set of integer points where r assumes an integer value. Its existence implies, by Theorem 5.17, that we may assume r to be of the form P/Q, where P and Q have integer coefficients.

We now consider the restrictions of r, P, and Q to a single rational ray $K \notin \Delta$,

using for coordinates an arbitrary point $\mathbf{c} \in K \cap \mathbb{Q}^n$. We have

$$r_{\mathbf{c}} = \frac{P_{\mathbf{c}}}{Q_{\mathbf{c}}},$$

where $P_{\mathbf{c}}$, $Q_{\mathbf{c}}$ are univariate polynomials. Applying polynomial division to divide $P_{\mathbf{c}}$ by $Q_{\mathbf{c}}$, we obtain a quotient $S^{(\mathbf{c})}(t)$ and a remainder $R^{(\mathbf{c})}$. The remainder is a polynomial of degree less than $\deg Q_{\mathbf{c}}$, and

$$r_{\mathbf{c}}(t) = S^{(\mathbf{c})}(t) + \frac{R^{(\mathbf{c})}(t)}{Q_{\mathbf{c}}(t)}\,. \tag{3}$$

Since $\deg R^{(\mathbf{c})} < \deg Q_{\mathbf{c}}$, the right-hand term converges to zero as $t \to \infty$. Applying Lemma 5.18 (part i), we obtain a rational $0 < q \leq 1$ such that for integer t, $S^{(\mathbf{c})}(t) \in q\mathbb{Z}$. Combining with (3), we have (for integer t large enough)

$$d(r_{\mathbf{c}}(t), q\mathbb{Z}) = \left| \frac{R^{(\mathbf{c})}(t)}{Q_{\mathbf{c}}(t)} \right|.$$

Since $r_{|K}$ is not p.n.i., there are infinitely many $t \in \mathbb{Z}^+$ such that $r_{\mathbf{c}}(t) \in \mathbb{Z}$, i.e., $d(r_{\mathbf{c}}(t), q\mathbb{Z}) = 0$. Thus we must have

$$\frac{R^{(\mathbf{c})}(t)}{Q_{\mathbf{c}}(t)} = 0$$

for such t, but then $R^{(\mathbf{c})}$ must be the zero polynomial. In other words, the polynomial $Q_{\mathbf{c}}$ is a divisor of $P_{\mathbf{c}}$.

Denote by $\sigma_i^{(\mathbf{c})}$ the coefficient of t^i in $S^{(\mathbf{c})}(t)$. We know that the latter polynomial is not p.n.i.; by Lemma 5.26, $\sigma_0^{(\mathbf{c})}$ is an integer. Recall that

$$P_{\mathbf{c}}(t) = P(c_1 t, c_2 t, \ldots, c_n t) = \sum_i p_i(c_1, \ldots, c_n) t^i,$$

where each p_i is a polynomial in $c_1, \ldots, c_n$ (in fact a homogenous polynomial of degree i). Similarly let us write

$$Q_{\mathbf{c}}(t) = \sum_i q_i(c_1, \ldots, c_n) t^i.$$

Note that the $S^{(\mathbf{c})}$ is defined by the equation

$$P_{\mathbf{c}} = Q_{\mathbf{c}} \cdot S^{(\mathbf{c})}\,.$$

By comparing corresponding coefficients, we convert this equation to the linear system (in the unknowns $\sigma_j^{(\mathbf{c})}$)

$$\begin{aligned}
p_0(\mathbf{c}) &= q_0(\mathbf{c})\sigma_0^{(\mathbf{c})}, \\
p_1(\mathbf{c}) &= q_1(\mathbf{c})\sigma_0^{(\mathbf{c})} + q_0(\mathbf{c})\sigma_1^{(\mathbf{c})}, \\
p_2(\mathbf{c}) &= q_2(\mathbf{c})\sigma_0^{(\mathbf{c})} + q_1(\mathbf{c})\sigma_1^{(\mathbf{c})} + q_0(\mathbf{c})\sigma_2^{(\mathbf{c})}, \\
&\;\;\vdots \\
&\;\;\vdots \\
&\;\;\vdots
\end{aligned} \tag{4}$$

If $q_0 \neq 0$, these equations are easy to solve:

$$\begin{aligned} \sigma_0^{(\mathbf{c})} &= \frac{p_0(\mathbf{c})}{q_0(\mathbf{c})}, \\ \sigma_1^{(\mathbf{c})} &= \frac{p_1(\mathbf{c})}{q_0(\mathbf{c})} - \frac{q_1(\mathbf{c})}{q_0(\mathbf{c})}\sigma_0^{(\mathbf{c})}, \\ &\vdots \end{aligned} \tag{5}$$

If $q_0(\mathbf{c}) = 0$, the first equation degenerates, so we proceed to the following equations. If $q_1(c) \neq 0$, we obtain

$$\sigma_0^{(\mathbf{c})} = \frac{p_1(\mathbf{c})}{q_1(\mathbf{c})}$$

and so on. To sum up, the form of the solution depends on the first polynomial of $q_0, q_1, \ldots$ which does not vanish at $\mathbf{c}$. The first q_i which has nonzero value is used as the denominator in a solution similar to (5). In particular, the solution for $\sigma_0^{(\mathbf{c})}$ is $\frac{p_i(\mathbf{c})}{q_i(\mathbf{c})}$.

To add some formal notation to the above discussion, let $\mathcal{V}(q)$ be the zero set of the polynomial q. We define

$$H_i = \left(\bigcap_{j<i} \mathcal{V}(q_j) \right) \setminus \mathcal{V}(q_i)$$

(with obvious modifications for the first and last i). H_i is the set of points $\mathbf{c}$ for which q_i is the denominator in the solution. It is easy to verify, that there is at most one index k such that H_k is a large set. In fact, $H_k = \Re^n - \mathcal{V}(q_k)$.

Note that each of the polynomials q_i is homogenous of degree i. It follows, that $\mathcal{V}(q_k)$ is scale invariant, and so is its complement H_k.

One must not forget that in the system (4) there are more equations than unknowns. It is possible that such a system will not have any solution. This occurs when the solution obtained from some of the equations does not fulfill the other equations. Note, however, that once the solutions for σ_i are substituted in the other equations, they become algebraic equations in $(c_1, \ldots, c_n)$. Thus, to every H_i we have an associated variety G_i that restricts the solution.

We argued before that every $\mathbf{c} \in \bigcup_{K \notin \Delta}(S(K))$ gives rise to divisibility of $P_{\mathbf{c}}$ by $Q_{\mathbf{c}}$, i.e., enables a solution of (4). Hence

$$\bigcup_{K \notin \Delta} (S(K)) \subseteq \bigcup_i (H_i \cap G_i) .$$

The set on the left-hand side is known to be large, so the set on the right-hand side must also be large. We conclude that there is a k such that $H_k \cap G_k$ is large. Then H_k is the unique large set mentioned above, and G_k is trivial, i.e., $H_k \cap G_k = H_k$.

We now get rid of the specific ray K we chose before, and consider the relationship of a general ray K to the set H_k. Since H_k is scale invariant, a ray is either contained in this set or disjoint from it. Let $K \subseteq H_k \, (= H_k \cap G_k)$. Pick a point $\mathbf{c} \in K \cap \mathbb{Q}^n$.

By definition of H_k and G_k, (2) can be solved for $\mathbf{c}$ and produce a polynomial $S^{(\mathbf{c})}$ such that $r_{\mathbf{c}} = S^{(\mathbf{c})}$.

The set of points on which $\sigma_0^{(\mathbf{c})}$ obtains integer values is contained in $\mathcal{B}(p_k, q_k)$. (Note that this bale also includes the case $q_k = 0$.) We argued above that for $K \notin \Delta$ and $\mathbf{c} \in S(K)$, $\sigma_0^{(\mathbf{c})}$ is an integer. Thus

$$\bigcup_{K \notin \Delta} S(K) \subseteq \mathcal{B}(p_k, q_k).$$

Since p_i and q_i are both homogenous, $\mathcal{B}(p_k, q_k)$ is scale invariant and contains all of $\bigcup_{K \notin \Delta} K$. This is a contradiction to our opening assumption $(*)$, unless the bale is trivial. A trivial bale satisfies $p_k(\mathbf{c})/q_k(\mathbf{c}) \in \mathbb{Z}$ for every $\mathbf{c}$ such that $q_k(\mathbf{c}) \neq 0$, i.e., for $\mathbf{c} \in H_k$. We conclude that $r_{|K}$ is p.i. for every ray contained in H_k. In other words, $\bigcup_{K \notin \Gamma} K \subseteq \mathcal{V}(q_k)$, a nontrivial scale-invariant variety. □

DEFINITION 5.29. *Let $P(\mathbf{x}) \in \Re[\mathbf{x}]$. We define $\psi(P)(\mathbf{x}) = \lim_{\lambda \to 0} \lambda^s P(\mathbf{x}/\lambda)$ where $s = \deg P$. That is, $\psi(P)$ is the sum of monomials in P of leading degree. For any rational function $r = P/Q$ where $P, Q \in \Re[\mathbf{x}]$, $\psi(r)(\mathbf{x}) \stackrel{\text{def}}{=} \psi(P)(\mathbf{x})/\psi(Q)(\mathbf{x})$.*

ψ is a "homogenizing operator" on polynomials and rational functions.

LEMMA 5.30. *If a set of rays in $\Re^n$ is contained in some nontrivial algebraic variety, then it is contained in a nontrivial, scale-invariant one.*

Proof. Let Γ be the set of rays and V a variety that contains it. Let $p_1, \ldots, p_k$ be the polynomials defining V. Let $\psi(V)$ be defined by $\psi(p_1) = 0, \ldots, \psi(p_k) = 0$. Clearly, $\psi(V)$ is scale invariant. We first claim that it is nontrivial: for $\psi(V)$ to be trivial, at least one of its defining polynomials must be identically zero. But $\psi(p_i) \equiv 0$ if and only if $p_i \equiv 0$. It rests to show that $\Gamma \subseteq \psi(V)$. To see this, let $K \in \Gamma$; then $K \subseteq V$, so $p_{i|K} \equiv 0$ for every i. In particular,

$$\lim_{\lambda \to 0} \lambda^s p_{i|K}(t/\lambda) = 0$$

for every t; i.e., $\psi(p_i)_{|K} \equiv 0$. Thus $K \subseteq \psi(V)$. □

6. Translating RAM programs into computation trees. The foundation for the proof of all our lower-bound theorems for RAMs is a way of translating a RAM program into a computation tree that computes the same function for almost all inputs (or almost all inputs within a certain neighborhood). The precise statement varies from theorem to theorem, and accordingly we present six different lemmas, that prepare the ground for our six theorems.[2] The starting point for these proofs is the technique of Paul and Simon [15]. The first lemma below is "almost" proved in their work.

LEMMA 6.1. *Let P be a $(\Re, \Re, \{+, -, \times, /\})$-RAM program that recognizes a set $W \subseteq \Re^n$ in time bounded by t. The initial contents of the random access memory are assumed to be zero except for a finite set of cells. Then P can be transformed into an algebraic computation tree, recognizing a set V, such that the following hold:*

(1) *$h(T) \leq t$;*

(2) *V is an open set;*

(3) *there exists a nontrivial algebraic variety H such that $W \cup H = V \cup H$.*

Proof. We make the following nonrestrictive assumptions on the program. The input variables, $x_1, \ldots, x_n$, reside in addresses $1, 2, \ldots, n$ of the random access memory. The program has access to a fixed set of *registers* $r_1, \ldots, r_k$ which are not in

[2]Note that the order of the lemmas does not correspond to the order of the theorems; it was chosen to facilitate the presentation of the proofs.

the random access memory and on which arithmetics and tests are performed, while main memory is accessed explicitly only by instructions that transfer the contents of a register into memory, or vice versa. These instructions can use *direct addressing* (the address is a program constant) or *indirect addressing* (the address is given in a specified register). The registers must be explicitly initialized, while the memory is preset to the program-determined values. Finally, we assume that the program ends with an output instruction that specifies the result of the computation (say, by naming a register).

We begin by representing P in tree form in the standard way, which involves unrolling loops, so that control only flows from root to leaves. This tree is potentially infinite; however, since we assume that the time complexity of the program is bounded by t, we may prune the tree at height t, without interfering with its correctness. The nodes of this tree are labeled with instructions of the program. To make it a computation tree, we have to replace all references to registers and memory locations with named variables. The variable names will be u_1, u_2, etc.

We start by replacing each left-hand occurrence of a register name by a unique variable name u_i. Next, we replace all right-hand occurrences in accordance with the closest preceding left-hand replacement. (Since we assume that registers must be initialized explicitly, there will always be one.)

Our main concern is handling memory references. We will use additional new variables to represent values stored in memory (except for the initial values). In order to determine the target of a memory reference, we have to find out whether the value used for addressing (itself the result of prior computation) coincides with the address used in a preceding memory reference. Since in our model $\mathcal{F} = \{+, -, \times, /\}$, all functions that can be computed by a straight-line sequence of instructions are rational functions from $\Re(\mathbf{x})$. Therefore, in order to analyze the memory references of the program, we represent each value computed as a rational function.

Recall that register names have already been replaced with variable names; thus it rests to handle memory access instructions, which can take one of four possible forms (here a represents a real constant):

(i) $\langle a \rangle \leftarrow u_i$
(ii) $\langle u_j \rangle \leftarrow u_i$
(iii) $u_i \leftarrow \langle a \rangle$
(iv) $u_i \leftarrow \langle u_j \rangle$

In order to analyze the memory references we apply the following analysis procedure to the whole tree. The procedure handles the nodes of the tree in a top-down order. With each node ν that sets a variable u_i we associate a function $f_i \in \Re(\mathbf{x})$ that represents the value assigned. With nodes representing STORE instructions (types (i) and (ii)), we also associate a function g_i that represents the address accessed. Later, we will change a STORE instruction into an assignment to a unique variable. This variable will be used in subsequent LOAD instructions only if the functions that describe the addresses accessed in the STORE and the subsequent LOAD are identical. This is a "rough" analysis, because if nonidentical functions yield the same address, our analysis will be incorrect; the inputs where such a failure happens will create the difference between V (the set recognized by the computation tree) and W (the set recognized by P).

The following rules are applied to each node, according to the type of instruction it represents.

(1) A decision node: no special handling.

(2) An assignment

$$u_i \leftarrow a, \qquad \text{where } a \in \Re.$$

We define $f_i = a$.
(3) An assignment

$$u_i \leftarrow u_j.$$

We define $f_i = f_j$. Note that f_j must have already been defined.
(4) A computational instruction

$$u_i \leftarrow u_j \circ u_k, \qquad \text{where } \circ \in \{+, -, \times, /\}.$$

We define $f_i = f_j \circ f_k$. Note that f_j and f_k must have been already defined.

(5) For a STORE instruction of type (i), we replace the left-hand side with a new variable u_k; we define $f_k = f_i$ and $g_k = a$.

(6) For a STORE instruction of type (ii), we replace the left-hand side with a new variable u_k; we define $f_k = f_i$ and $g_k = f_j$.

(7) For a LOAD instruction of type (iii), we look for the closest ancestor of the current node that represents an instruction of type (i) or (ii) and has $g_k = a$. If we find such a node, we replace the right-hand side of the assignment with u_k. If we do not, and $a \in \{1, \ldots, n\}$, we replace the right-hand side with x_a. Otherwise, we replace it with a constant that is the initial value of memory address a (either specified explicitly by the program or the default value zero).

(8) For a LOAD instruction of type (iv), we look for the closest ancestor of the current node that represents an instruction of type (i) or (ii) and has $g_k = f_j$. If we find such a node, we replace the right-hand side of the assignment with u_k. If we do not, and $f_j \in \{1, \ldots, n\}$, we replace the right-hand side with x_{f_j}. Otherwise, if $f_j = a$ where a is one of the addresses initialized to a nonzero value, we replace the right-hand side with this value. In all other cases we replace the right-hand side with a zero.

In cases (1) through (6), it is clear that the function f_i represents correctly the value of u_i (assuming ancestors were handled correctly). Only cases (7)–(8) require finer consideration. The analysis of case (7) is correct, provided that the input vector $\mathbf{x}$ does not mislead us by satisfying an equation $g_k(\mathbf{x}) = a$ which is not an identity. Similarly, the analysis of case (8) is correct, provided that the input vector $\mathbf{x}$ does not mislead us by satisfying a nontrivial equation among $g_k(\mathbf{x}) = f_j(\mathbf{x})$, $f_j(\mathbf{x}) = 1, \ldots, f_j(\mathbf{x}) = n$, and $f_j(\mathbf{x}) = a$, where a is one of the addresses initialized to a nonzero value. Note that the set of misleading $\mathbf{x}$ values is defined by a finite number of equations in rational functions. Each equation of rational functions, say

$$\frac{P(\mathbf{x})}{Q(\mathbf{x})} = \frac{R(\mathbf{x})}{S(\mathbf{x})}$$

(P, Q, R, and S are polynomials), can be rewritten as

$$PS - RQ = 0 \tag{6}$$

(possibly adding solutions where one of the denominators is zero). This way we obtain a set H_1, defined by a finite number of polynomial equations, i.e., an algebraic variety, that contains all the misleading $\mathbf{x}$ values. H_1 is a nontrivial variety, for the defining

equations are never identities. For every $\mathbf{x} \in \Re \setminus H_1$, the computation tree obtained simulates the RAM program faithfully.

For every decision node $u_i : u_j$, we can now write the comparison in terms of rational functions $f_i : f_j$. If this comparison happens to be trivial (has only one possible outcome), we delete the decision node, keeping the only child that is reached. For every remaining decision node, we remove the branch of equality. Namely, we use the form of decision nodes with three children ($<$, $=$, and $>$) and replace every subtree rooted at $=$ with a rejecting leaf. Let H_2 be the set of inputs that satisfy one or more of the equations $f_i = f_j$ (rewritten as in (6)); H_2 is a nontrivial algebraic variety.

Following the last change, we obtain a tree T that simulates P faithfully if the input is not in $H = H_1 \cup H_2$.

Let V be the set recognized by T. Now, V is an open set because it is defined by strict polynomial inequalities. If $\mathbf{x} \in W \cup H$, then either $\mathbf{x} \in H$ or $\mathbf{x} \in W \setminus H$. In the latter case, $\mathbf{x}$ is accepted by the program P; moreover the tree simulates P faithfully on $\mathbf{x}$ and therefore accepts it. Thus $\mathbf{x} \in V$. Similarly, if $\mathbf{x} \in V \setminus H$, then $\mathbf{x} \in W$. We have proved the lemma. □

LEMMA 6.2. *Let P be a $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$-RAM program that recognizes a set $W \subseteq \Re^n$ in time bounded by t. The initial contents of the memory cells of integer addresses may be arbitrarily specified by P. Then P can be transformed into an algebraic computation tree recognizing a set V, such that the following hold:*

(1) *$h(T) \leq t$;*

(2) *V is an open set;*

(3) *there exists a nontrivial bale H such that $V \cup H = W \cup H$.*

Proof. We point out only the differences from the proof of Lemma 6.1. The extensions allowed by our RAM are integrality testing and free initialization of all integer-addressed cells. We consider the latter extension first. It affects the proof of Lemma 6.1 by introducing infinitely many equations for misleading inputs (only for case 8). These are the equations:

$$\{f_j(\mathbf{x}) = a : a \text{ is an initialized address}\}.$$

Since f_j is a rational function, the solutions of these equations belong to the bale $\mathcal{B}(P, Q)$ where $P/Q = f_j$. The other prohibited equations define algebraic varieties, which are a special case of bale. Thus the total set of misleading inputs is contained in a finite union of bales, i.e., in a bale. We initially define H_1 to be this bale.

We proceed to handle tests of the two kinds: ordinary comparisons are handled as before, giving rise to the set H_2. Now we also have integrality tests. Here we distinguish two cases. For a given integrality-testing node, it is possible that the test passes for all input that arrives there (i.e., only integers are ever tested). If this is the case, the node is simply eliminated, and only the "yes" branch kept. Otherwise, we eliminate the node as well but keep the "no" branch instead; to maintain correctness, we forbid all input that passes the test. Since the variable tested has already been associated with a rational function, the set of inputs that pass the test forms a bale (and not the trivial one, because some inputs do not pass). Call this bale H_3. We let $H = H_1 \cup H_2 \cup H_3$. □

LEMMA 6.3. *Let P be a $(\Re, \mathbb{N}, \{+, -, \times, /\})$-RAM program that recognizes a set $W \subseteq \Re^n$ in time bounded by t. The initial contents of the random access memory may be arbitrarily specified by P. Then P can be transformed into an algebraic computation tree recognizing a set V, such that the following hold:*

(1) $h(T) \leq t$;
(2) V *is an open set;*
(3) *there exists a boundary set* A *such that* $V \cup A = W^\circ$.

Proof. T is obtained from P much as in the proof of Lemma 6.1, with a difference only in the handling of indirect addressing operations. Here, we treat every memory access as a direct addressing one. For the constant address, we use the value of the function g_k at an arbitrary point of its domain of definition.

We now elaborate the effects of this change. Recall that in the process that derives T from the program tree, certain parts of the tree have been removed. First, we remove all branching nodes where there is only one possible outcome (with no effect on the computation). Next, every node, that is reached on a nontrivial equality, is replaced by a rejecting leaf. (This change gives rise to the set H_2.)

Therefore, it suffices to consider indirect addressing nodes in those parts of the tree that are not eliminated. Let ν be such a node and D_ν be the set of inputs $\mathbf{x} \in \Re^n$ that arrive at ν (more precisely, the computation on $\mathbf{x}$ arrives at ν). This set is defined by the branching conditions on the path to ν, which are all strict inequalities among polynomials; it is therefore an open set. Let f be the function, that yields the address for the memory access at ν. It is a rational function, and its value must be integer for all $\mathbf{x} \in D_\nu$; otherwise the program would not be valid. By Lemma 5.12, f is constant throughout its domain of definition, justifying the change described above.

After replacing register names with variables, we proceed to replace memory access operations with assignments. In the proof of Lemma 6.1, we argued that this substitution is safe for input $\mathbf{x}$ unless $\mathbf{x}$ satisfies one of a certain set of nontrivial equations, giving rise to a variety of "misleading input" H_1. The equations test whether the memory address coincides with an address accessed in some prior node or with a preset location. In the current construction we have changed all address expressions into constants: thus the equations in question become relations between constants. They can hold for $\mathbf{x}$ if and only if they are identities, but identities were not included in the definition of H_1. The set corrseponding to H_1 for the current construction is thus empty.

We conclude that T simulates P faithfully if the input is not in H_2. All inputs in H_2 are rejected; therefore, every input accepted by T is indeed in W, and we obtain

$$\begin{aligned} V \subseteq & W^\circ \subseteq W \subseteq V \cup H_2 \\ \Rightarrow \quad & W^\circ = V \cup (W^\circ \cap H_2)\,. \end{aligned}$$

We thus let $A = W^\circ \cap H_2$. Since H_2 is a boundary set, so is A. □

LEMMA 6.4. *Let* P *be a* $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$*-RAM program that recognizes a set* $U \subseteq \Re^n$ *in time bounded by* t. *The initial contents of the random access memory are assumed to be zero except for a finite set of cells. Then* P *can be transformed into an algebraic computation tree recognizing a set* V, *such that the following hold:*

(1) $h(T) \leq t$;
(2) V *is an open set;*
(3) *there exists a nontrivial, scale-invariant bale* B *such that for all* $K \in \mathcal{Q}^{(n-1)}$, *where* $K \not\subseteq B$, $\exists a_K \in \mathbb{Z}^+$ *such that for all* $\mathbf{x} \in (K \cap a_K \mathbb{Z}^n), \mathbf{x} \in U \Leftrightarrow \mathbf{x} \in V$.

Proof. We extend the construction of Lemma 6.1 to handle the integrality predicate. This extension will introduce the bale mentioned in the lemma. We consider first the case where the integrality predicate is not used by the program; then we can apply Lemma 6.1 and obtain a variety H such that $U \cup H = V \cup H$. Let Γ be the set of rays K wholly contained in H. By Lemma 5.30, Γ may be completed to a nontrivial,

scale-invariant variety which we now call B. Consider a rational ray $K \not\subseteq B$. Only a finite number of points from K may be contained in H (Lemma 5.21). The conclusion of our lemma is immediate.

We proceed to programs with integrality tests $\chi_{\mathbb{Z}}$. This operation is represented in the program tree by a branching node, where the branch labeled *yes* is taken if and only if the register tested contains an integer. Note that the construction of Lemma 6.1 gives no special treatment to decision nodes of any kind, so it can be carried out in the extended model with no change. We deal with integrality tests after performing the original construction as described for Lemma 6.1. Consider now a $\chi_{\mathbb{Z}}$-node ν, the first such node on its path. The register name it used has been replaced by a variable u_i, and a rational function f_i has been found that represents the value of u_i on "safe" input. Because there are no $\chi_{\mathbb{Z}}$ nodes before ν, the safety requirement applying to f_i is the same as in Lemma 6.1, i.e., input $\notin H$.

Let

$$\Gamma_i = \{K \in \mathcal{Q}^{(n-1)} : f_{i|K} \text{ is p.i.}\}$$

and similarly

$$\Delta_i = \{K \in \mathcal{Q}^{(n-1)} : f_{i|K} \text{ is p.n.i.}\}.$$

Theorem 5.28 shows that either $\bigcup_{K \notin \Gamma_i} K$ or $\bigcup_{K \notin \Delta_i} K$ is contained in a nontrivial, scale-invariant bale. We add this bale to B. (A union of two nontrivial, scale-invariant bales is once again such a bale.)

First assume that $\bigcup_{K \notin \Gamma_i} K \subseteq B$. Then we remove the decision done and keep only the *yes* branch. The change is safe on input $\mathbf{x}$, provided that $f_i(\mathbf{x}) \in \mathbb{Z}$. Now if $K \not\subseteq B$ then $K \in \Gamma_i$, so f_i is p.i. on K. Thus there is an $a_i^K \in \mathbb{Z}^+$, such that the modified tree handles $\mathbf{x}$ correctly if $\mathbf{x} \in K \cap a_i^K \mathbb{Z}^n$.

In the other case we have $\bigcup_{K \notin \Delta_i} K \subseteq B$. Then we remove the decision done and keep only the *no* branch. The safety argument is similar.

We complete the construction by setting a_K, for every $K \not\subseteq B$, to be a common multiple of the a_i^K associated with all the nodes eliminated. □

LEMMA 6.5. *Let P be a $(\Re, \Re, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program that recognizes a set $W \subseteq \Re^n$ in time bounded by t. The initial contents of the random access memory cells of integer addresses may be arbitrarily specified by P. Then there exists a constant $\lambda_0 > 0$ such that for all $0 < \lambda < \lambda_0$, P can be transformed into an algebraic computation $T = T_\lambda$ such that the following conditions hold:*

(1) *$h(T) \leq 3t + 2n + 1$;*

(2) *T recognizes an open set V;*

(3) *there exists a boundary set A such that $V \cup A = (W \cup A) \cap B(\lambda, \mathbf{0})$.*

Proof. We start by constructing a computation tree, T_0, from P, much as in the proofs of Lemmas 6.1 and 6.2. As in the former constructions, we associate with each variable u_i a function f_i that represents the value assigned to that variable. However, this time, the function will be a polynomial. In fact, the computation tree that we derive only includes the operations $\{+, -, \times\}$. Floor operations are eliminated in course of the construction, and it is their elimination that necessitates restricting the input to the neighborhood $B(\lambda, \mathbf{0})$. The constant λ_0 is derived in the construction.

The first stage, as in the former lemmas, involves unfolding the program to tree form and replacing register names by variables, obtaining the program tree T_P of height t. This tree may now include operations $u_i \leftarrow \lfloor u_j \rfloor$. These nodes are handled

specially as described below. With each such node, a constant λ_i is associated, in addition to the function f_i. The construction proceeds top-down, applying the following rules to each node according to its type.

(1) A decision node: no special handling.

(2) An assignment

$$u_i \leftarrow a, \qquad \text{where } a \in \Re.$$

We define $f_i = a$.

(3) An assignment

$$u_i \leftarrow u_j.$$

We define $f_i = f_j$. Note that f_j must have already been defined.

(4) An arithmetic instruction

$$u_i \leftarrow u_j \circ u_k, \qquad \text{where } \circ \in \{+, -, \times\}.$$

We define $f_i = f_j \circ f_k$. Note that f_j and f_k must have been already defined, and that the resulting function is a polynomial.

(5) A truncation instruction

$$u_i \leftarrow \lfloor u_j \rfloor.$$

Let $c_j = f_j(\mathbf{0})$. We distinguish two cases.

(5.1) $c_j \notin \mathbb{Z}$, i.e., $\lfloor c_j \rfloor < c_j < \lceil c_j \rceil$. By continuity of f_j, we choose $\lambda_i > 0$ such that for all $\mathbf{x} \in B(\lambda_i, \mathbf{0})$ we have $\lfloor c_j \rfloor < f_j(\mathbf{x}) < \lceil c_j \rceil$. Hence, $\lfloor f_j(\mathbf{x}) \rfloor = \lfloor c_j \rfloor$ for all such $\mathbf{x}$. We relabel the current node with the instruction

$$u_i \leftarrow \lfloor c_j \rfloor$$

and set $f_i = \lfloor c_j \rfloor$.

(5.2) $c_j \in \mathbb{Z}$. By continuity of f_j, we choose $\lambda_i > 0$ such that for all $\mathbf{x} \in B(\lambda_i, \mathbf{0})$ we have $c_j - 1 < f_j(\mathbf{x}) < c_j + 1$. Hence, $\lfloor f_j(\mathbf{x}) \rfloor \in \{c_j - 1, c_j\}$ for all such $\mathbf{x}$. We replace the current node with a branching structure, which tests the value of $f_j(\mathbf{x})$ to determine which of the two possibilities is appropriate, and proceed accordingly. This transformation is shown in Figure 1, where T_1' (resp., T_1'') represents a copy of T_1 with new (primed; resp., double-primed) variable names; u_k is an additional variable not in the original tree. The functions f_k, f_i and $f_{i'}$ are now defined as in case (2).

The rest of the nodes, representing LOAD and STORE operations, are replaced with assignment operations just as in the proof of Lemma 6.2. This results in the definition of a "forbidden bale" H. This set contains all input values for which memory references were not resolved correctly. The resulting tree T_0 simulates the RAM program faithfully provided that the input is not in H, *and* the replacement of floor operations is correct. The latter condition will be fulfilled if $\mathbf{x} \in B(\lambda_0, \mathbf{0})$ where λ_0 is the minimum over the λ_i defined. (If none were defined, which means that there were no floor operations, λ_0 can be chosen arbitrarily.)

We conclude the construction by eliminating redundant comparisons and then replacing every $=$-branch (in the comparison nodes that remained) with a rejecting leaf. These equality branches define another variety, H_2, which contains input which we may unjustly reject. The effect of the last change is that the set V_1 accepted by the resulting tree T_1 is open. It also satisfies, for $H = H_1 \cup H_2$, and for every $0 < \lambda < \lambda_0$,

$$(V_0 \cup H) \cap B(\lambda, \mathbf{0}) = (W \cup H) \cap B(\lambda, \mathbf{0}).$$

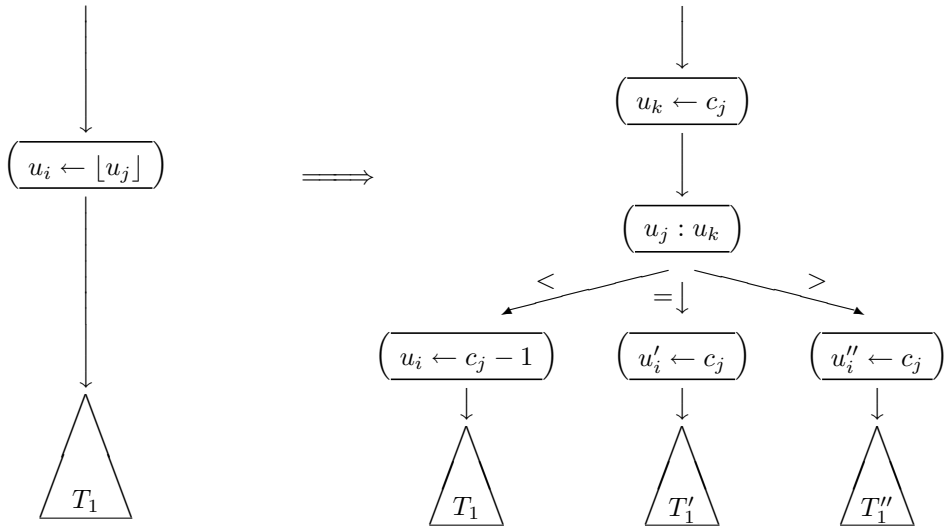


FIG. 1. *Transformation of truncation nodes.*

It is easy to see that $h(T_1) \leq 3t$. We now extend T_1 to the desired tree T by inserting, before the root of T_1, a series of nodes that compute

$$\|\mathbf{x}\| = \left(\sum_{i=1}^{n} x_i^2 \right),$$

and finally compare this value with λ. The branch for $\|\mathbf{x}\| < \lambda$ is connected to the root of T_1; the other branches reject. It is clear that the resulting tree recognizes the set $V = V_1 \cap B(\lambda, \mathbf{0})$. Let $A = H \cap B(\lambda, \mathbf{0})$. It is easy to verify that parts (2) and (3) of the lemma is satisfied. Verifying part (1) is also straightforward. □

LEMMA 6.6. *Let P be a $(\Re, \mathbb{N}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program that recognizes a set $W \subseteq \Re^n$ in time bounded by t. The initial contents of the random access memory may be arbitrarily specified by P. Then there exists a constant $\lambda_0 > 0$ such that for all $0 < \lambda < \lambda_0$, P can be transformed into an algebraic computation tree $T = T_\lambda$ such that the following hold:*

(1) *$h(T) \leq 3t + 2n + 1$;*
(2) *T recognizes an open set V;*
(3) *there exists a boundary set H such that $V \cup H = W^\circ \cap B(\lambda, \mathbf{0})$.*

We omit details of the proof, which is a straightforward combination of the proofs of Lemmas 6.5 and 6.3.

LEMMA 6.7. *Let P be a $(\mathbb{Q}, \mathbb{Q}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program that recognizes a set $U \subseteq \mathbb{Q}^n$ in time bounded by t. The initial contents of the random access memory are assumed to be zero except for a finite set of cells. Then P can be transformed into an algebraic computation tree T recognizing a set $V \subseteq \Re^n$, such that the following hold:*

(1) *$h(T) \leq 2t$;*
(2) *V is an open set;*
(3) *there exist a nontrivial algebraic variety H and a constant $a \in \mathbb{Z}^+$ such that*

$$V \cap (a\mathbb{Z}^n \setminus H) = U \cap (a\mathbb{Z}^n \setminus H).$$

Proof. We construct T from P in a way similar to the construction of Lemma 6.1. As in the previous constructions, we associate with each variable u_i a function f_i that represents the value assigned to that variable. This time, this function will not be a rational function but a polynomial. Thus only the operations $\{+, -, \times\}$ take part in these functions; floor operations will be eliminated. We will define the constant a and show that these functions represent the computation precisely enough for input in $a\mathbb{Z}^n$.

As usual we start by representing the program as a tree, renaming registers to unique variables. In this tree, there still are operations of the form $u_i \leftarrow \lfloor u_j \rfloor$. These operations have to be replaced by algebraic operations. The idea that allows this is as follows. If only the operations $\{+, -, \times\}$ had been used, all functions computed by the program would have been polynomials. We study the effect of applying the truncation operation to a polynomial. Consider for simplicity a univariate polynomial $c_k x^k + \cdots + c_1 x + c_0$. All c_j are rational. Let d be the common denominator of $\{c_1, \ldots, c_k\}$. Then, if x is divisible by d, the fractional part of this polynomial is determined by the constant c_0. Thus, computing the "floor" of this expression is achieved by subtracting a fixed constant. Hence, for input suitably restricted, we can replace this truncation operation with subtraction. In this way, we eliminate one by one all the truncation operations in the tree.

More precisely, we now associate with each variable u_i an integer a_i (in addition to the function f_i which is used as before). The construction proceeds top-down, applying the following rules to each node according to its type.

(1) A decision node: no special handling.

(2) An assignment

$$u_i \leftarrow c, \qquad \text{where } c \in \mathbb{Q}.$$

We define $f_i = c$ and $a_i = 1$.

(3) An assignment

$$u_i \leftarrow u_j.$$

We define $f_i = f_j$ and $a_i = 1$. Note that f_j must have already been defined.

(4) An instruction

$$u_i \leftarrow u_j \circ u_k, \qquad \text{where } \circ \in \{+, -\}.$$

We define $f_i = f_j \circ f_k$, and $a_i = 1$. Note that f_j and f_k must have been already defined and that the resulting function is a polynomial.

(5) A multiplication instruction

$$u_i \leftarrow u_j \cdot u_k.$$

We define $f_i = f_j \cdot f_k$. Note that f_j and f_k must have been already defined and that the resulting function is a polynomial. We define a_i to be the least common denominator of the constant terms of the polynomials f_j, f_k.

(6) A truncation instruction

$$u_i \leftarrow \lfloor u_j \rfloor.$$

Let c_j be the constant term of f_j. We replace this node by two nodes, labeled with the instructions

$$u_k \leftarrow c_j - \lfloor c_j \rfloor$$
$$u_i \leftarrow u_j - u_k$$

(u_k is a new variable). Note that this is a correct replacement for the truncation instruction, provided that f_j represents correctly the value of u_j and that the non-constant part of f_j is integer. We define f_i, f_k and a_i, a_k accordingly.

The rest of the nodes, representing LOAD and STORE operations, are handled exactly as in the Paul–Simon construction. a_i is defined as 1 for all of these instructions. This results in the definition of a "forbidden variety" H_1 in the usual way, which contains all input values for which memory references were not resolved correctly. We obtain that the tree simulates the RAM program faithfully, provided that the input is not in H_1 *and* the replacement of floor operations is correct. To this end we define a to be the product of all the a_i. We claim that if the input belongs to $a\mathbb{Z}^n$, then these replacements are correct. As above, we denote by c_j the constant term of f_j. We assume for simplicity that the variables are numbered so that the variables defined before u_i (that is, in ancestors thereof) have lower indices.

CLAIM 6.8. *Let $\mathbf{x} \in a\mathbb{Z}^n \setminus H_1$. On input $\mathbf{x}$, the value assigned to u_i is $f_i(\mathbf{x})$, for every u_i in T. The value of $f_i(\mathbf{x}) - c_i$ is an integer divisible by the $d_i = a/(a_1 \cdot \cdots \cdot a_i)$.*

Hence the replacement of all floor operations is correct for such $\mathbf{x}$.

The claim is proved by induction, starting at the root of the tree. For variables defined by case (2) it is immediate. For variables defined by cases (3)–(4) the claim follows from the inductive assumption regarding u_j and u_k. Consider case (5). It is clear that if the values assigned to u_j and u_k are, respectively, $f_j(\mathbf{x})$ and $f_k(\mathbf{x})$, then the value assigned to u_i is $f_i(\mathbf{x})$. Now

$$f_i(\mathbf{x}) - c_i = f_j(\mathbf{x}) \cdot f_k(\mathbf{x}) - c_j c_k = (f_j(\mathbf{x}) - c_j) f_k(\mathbf{x}) + c_j (f_k(\mathbf{x}) - c_k).$$

The inductive assumption shows that $f_j(\mathbf{x}) - c_j$ is a multiple of $d_j = a/(a_1 \cdots a_j)$, and hence of $d_{i-1} = a/(a_1 \cdots a_{i-1})$. By the definition of a_i, it is the common denominator of the rational numbers $f_k(\mathbf{x})$ and $f_j(\mathbf{x})$. Thus $(f_j(\mathbf{x}) - c_j) f_k(\mathbf{x})$ is a multiple of $d_{i-1}/a_i = d_i$. The same holds for $c_j(f_k(\mathbf{x}) - c_k)$.

In case (6) the inductive assumption shows that the value of u_j is $f_j(\mathbf{x})$ and that $f_j(\mathbf{x}) - c_j$ is a whole number; thus the fractional part of u_j is indeed $c_j - \lfloor c_j \rfloor$. It follows that the new instructions compute u_i correctly (for this input).

STORE and LOAD instructions were replaced by regular assignments; the claim thus holds, as it does for regular assignments, as long as the replacement is valid. We know (see Lemma 6.1) that it is valid for input not in H_1. This concludes the proof of the claim.

Our construction is concluded, as in Lemma 6.1, by deleting trivial comparisons and replacing equality branches that remain by a rejecting leaf. This gives rise to the nontrivial variety H_2. As usual, we define $H = H_1 \cup H_2$. Now, the set V recognized by the resulting tree is an open set. If $\mathbf{x} \in U \cap (a\mathbb{Z}^n \setminus H)$, Claim 6.8 applies to $\mathbf{x}$, showing that the tree is faithful to the program. From $\mathbf{x} \in U$ it follows that $\mathbf{x} \in V$. We have thus shown

$$U \cap (a\mathbb{Z}^n \setminus H) \subseteq V \cap (a\mathbb{Z}^n \setminus H).$$

The reverse containment is similarly proved. □

7. Proofs of lower bound theorems.

7.1. Extending Ben-Or's theorem. Theorem 3.1 is a direct consequence of the following lemma [2].

LEMMA 7.1. *Let T be an algebraic computation tree recognizing $W \subseteq \Re^n$. Then $\beta(W) \leq 9^{h(T)} 3^n$.*

Ben-Or proves this lemma by applying, in an elegant way, results of Milnor [14] and Thom [18] on the topology of varieties. Using this lemma we can also prove Theorem 3.1′. Let T be a computation tree that recognizes W. Let T' be obtained from T by rejecting all nontrivial equalities (as in the proof of Lemma 6.4). Note that $h(T') \leq h(T)$. Let V be the set recognized by T'; obviously $V \subseteq W$ and is open. Thus $V \subseteq W^\circ$. In fact, $V = W^\circ \setminus H$ where H is the union of all (nontrivial) varieties defined by the equalities affected. Combining Lemma 7.1 with Lemma 5.6 we have

$$\beta(W^\circ) \leq \beta(V) \leq 9^{h(T')}3^n \leq 9^{h(T)}3^n,$$

whereby

$$h(T) \geq \log_9 \beta(W^\circ) - \frac{1}{2}n.$$

Theorem 3.1′ follows.

7.2. Proof of Theorem 3.3. Consider a $(\Re, \mathbb{N}, \{+, -, \times, /\})$-RAM program P recognizing W in time t. Let T and V be given by Lemma 6.3. By the last lemma,

$$\beta(V) \leq 9^t 3^n.$$

Using Lemmas 6.3 and 5.6,

$$\beta(V) \geq \beta(W^\circ).$$

Combining the two inequalities we have

$$9^t 3^n \geq \beta(W^\circ).$$

Theorem 3.3 follows.

7.3. Proof of Theorem 3.4. Consider an $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$-RAM program P recognizing W in time t. Let T, V, and H be given by Lemma 6.2. By Lemma 7.1,

$$\beta(V) \leq 9^t 3^n.$$

Using Lemmas 6.2 and 5.3,

$$(\overline{W})^\circ = (\overline{W \cup H})^\circ = (\overline{V \cup H})^\circ = (\overline{V})^\circ,$$

and by Lemma 5.5,

$$\beta((\overline{W})^\circ) = \beta((\overline{V})^\circ) \leq \beta(V) \leq 9^t 3^n.$$

Theorem 3.4 follows.

7.4. Proof of Theorem 3.6. For simplicity we assume $\tilde{\beta}(W^\circ)$ and $\tilde{\beta}(W)$ to be finite. If one of them is infinite the theorem claims that there is no genuinely time-bounded program for recognizing W, and the proof is a simple extension of the finite case.

Consider a $(\Re, \mathbb{N}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program P recognizing W in time t. Let λ_0 be given by Lemma 6.6. Since the limit $\tilde{\beta}(W^\circ)$ is finite, there is a number $0 < \lambda < \lambda_0$

such that $\beta(W^\circ \cap B(\lambda, \mathbf{0})) = \tilde{\beta}(W^\circ)$. Pick $T = T_\lambda$ as in Lemma 6.6, and let V be the set T recognizes. By Lemma 7.1,

$$\beta(V) \leq 9^{3t+2n+1}3^n = 9^{3t+1}3^{5n}.$$

Using Lemmas 6.6 and 5.6,

$$\beta(V) \geq \beta(W^\circ \cap B(\lambda, \mathbf{0})) = \tilde{\beta}(W^\circ).$$

Combining the two inequalities we have

$$9^{3t+1}3^{5n} \geq \tilde{\beta}(W^\circ)$$

or

$$t \geq \frac{1}{3}(\log_9 \tilde{\beta}(W^\circ) - 1) - \frac{5}{6}n.$$

A similar reasoning gives the same inequality with W instead of W°, and together they prove Theorem 3.6.

7.5. Proof of Theorem 3.7. As above, we assume $\tilde{\beta}((\overline{W})^\circ)$ to be finite. Consider a $(\Re, \Re, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program P recognizing W in time t. Let λ_0 be given by Lemma 6.5. Since the limit $\tilde{\beta}(W^\circ)$ is finite, there is a number $0 < \lambda < \lambda_0$ such that $\beta(W^\circ \cap B(\lambda, \mathbf{0})) = \tilde{\beta}(W^\circ)$. Pick T, V and A as in Lemma 6.5. By Lemmas 5.5 and 7.1,

$$\beta((\overline{V})^\circ) \leq \beta(V) \leq 9^{3t+2n+1}3^n = 9^{3t+1}3^{5n}\,.$$

On the other hand,

$$\begin{aligned}
(\overline{W})^\circ \cap B(\lambda, \mathbf{0}) &= (\overline{W \cup A})^\circ \cap B(\lambda, \mathbf{0}) && \text{(Lemma 5.3)} \\
&= (\overline{(W \cup A) \cap B(\lambda, \mathbf{0})})^\circ && \text{(Lemma 5.4)} \\
&= (\overline{V \cup A})^\circ && \text{(Lemma 6.5)} \\
&= (\overline{V})^\circ.
\end{aligned}$$

Combining the last results we have

$$9^{3t+1}3^{5n} \geq \tilde{\beta}((\overline{W})^\circ).$$

Theorem 3.7 follows.

7.6. Extending Yao's lower bound. We start by reviewing Yao's proof of Theorem 3.2, and extend it to prove both Theorem 3.2 and Theorem 3.2′. We denote by $\mathcal{A}_{\mathbb{Z}^n, W}$ the class of algebraic computation trees that recognize the set $W \subseteq \Re^n$ when restricted to input in $\mathbb{Z}^n$.

Let $T \in \mathcal{A}_{\mathbb{Z}^n, W}$. With any node ν in T that represents an assignment or a computation, we associate a rational function r_ν that represents the value assigned or computed. Namely, if a constant if assigned, r_ν is this constant. If a previous variable is assigned, which was itself defined at node μ, then $r_\nu = r_\mu$. A computation node is handled as follows. Consider a node labeled with $u \leftarrow z \circ w$, where z, w are defined at nodes μ, τ, respectively. Then $r_\nu = r_\mu \circ r_\tau$ $(\circ \in \{+, -, \times, /\})$.

We define p_ν and q_ν to be polynomials such that $r_\nu = p_\nu/q_\nu$. These polynomials are defined in the natural way, e.g., in case $r_\nu = r_\mu \times r_\tau$ we have $p_\nu = p_\mu p_\tau$ and $q_\nu = q_\mu q_\tau$.

For any leaf $l \in T$, we denote by $\xi_{T,l}$ the path from the root of T to l. For $\alpha \in \{0,1\}$, let L_α be the set of leaves l such that the output of l is α and such that $\xi_{T,l}$ traverses no arcs labeled "=". Let $S_{T,l}$ be the set of $\mathbf{x} \in \Re^n$ that will trace the path $\xi_{T,l}$ in T.

DEFINITION 7.2. *T is called* normal *if, for every leaf l, the sequence of operations from the root to l has the form*

$$\begin{aligned} &z_1 \leftarrow u_1\ op_1\ w_1,\\ &z_1 : 0,\\ &z_2 \leftarrow u_2\ op_2\ w_2,\\ &z_2 : 0,\\ &\quad\cdot\\ &\quad\cdot\\ &\quad\cdot\\ &z_m \leftarrow u_m\ op_m\ w_m,\\ &z_m : 0, \end{aligned}$$

where $m \geq n$; for $1 \leq i \leq m$, $op_i \in \{+,-,\times,/\}$, every u_i and w_i is either a constant or an element of $\{z_1, z_2, \ldots, z_{i-1}\} \cup \{x_1, \ldots, x_n\}$, and for $1 \leq j \leq n$, $u_j = x_j$, $w_j = 1$ and $op_j = \times$.

DEFINITION 7.3. *T is* irredundant *if every node u can be traversed by some input $\mathbf{x} \in \mathbb{Z}^n$, and every computation node $\nu \in T$ computes a function r_ν which is not constant.*

DEFINITION 7.4. *A* regular *computation tree is one that is normal and irredundant.*

LEMMA 7.5 (Yao). *For every $T \in \mathcal{A}_{\mathbb{Z}^n,W}$ there exists $T' \in \mathcal{A}_{\mathbb{Z}^n,W}$ such that T' is regular and $h(T') \leq 2h(T) + 2n$.*

Proof. Add the sequence of instructions $z_i \leftarrow x_i \times 1$, $i = 1, \ldots, n$, at the beginning; expand each internal node into a computation node $z_i \leftarrow \cdots$ followed immediately by a branching node $z_i : 0$. Find the function r_ν computed at every node. For functions that are constant, remove this node (and the succeeding test), leaving only the correct branch of the test, and replace future references to the variable computed by the corresponding constant. Now prune away all the branches in the tree that are not traversed by any input $\mathbf{x} \in \mathbb{Z}^n$. □

We now define an operator ϕ, that translates every regular computation tree into a related decision tree. The technique developed by Yao gives a lower bound on the height of $\phi(T)$ from the topology of the set it recognizes. We further relate this set to the set recognized by T in order to derive a lower bound for the latter.

The following definition is repeated here for convenience.

DEFINITION 5.29. *Let $P(\mathbf{x}) \in \Re[\mathbf{x}]$. We define $\psi(P)(\mathbf{x}) = \lim_{\lambda \to 0} \lambda^s P(\mathbf{x}/\lambda)$ where $s = \deg P$. That is, $\psi(P)$ is the sum of monomials in P of leading degree. For any rational function $r = P/Q$ where $P, Q \in \Re[\mathbf{x}]$, let $\psi(r)(\mathbf{x}) = \psi(P)(\mathbf{x})/\psi(Q)(\mathbf{x})$.*

DEFINITION 7.6. *For any regular T, $\phi(T)$ is the decision tree obtained from T as follows. For each branching node ν of T, group the pair $(\mu = parent[\nu], \nu)$ into one node $\phi(\nu)$. Associate it with the test $\psi(p_\mu)(\mathbf{x}) \cdot \psi(q_\mu)(\mathbf{x}) : 0$. The three children of this node (labeled <,=,>) are obtained from the corresponding children in T.*

LEMMA 7.7 (Yao). *Let T be regular. For any leaf $l \in L_0(T) \cup L_1(T)$, we have $\beta(S_{\phi(T),\phi(l)}) \leq 2 \cdot 3^{n+h(T)}$.*

Note that $L_0(T) \cup L_1(T)$ is the set of leaves l such that the path to l traverses no arc labeled "=".

LEMMA 7.8 (Yao). *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed, and let $T' \in \mathcal{A}_{\mathbb{Z}^n,W}$. Then there is a boundary set A such that*

$$\bigcup_{l \in L_1(T)} S_{\phi(T'),\phi(l)} = W \setminus A\,.$$

For proofs of both lemmas see [19].

We now prove Theorems 3.2 and 3.2′.

Let $T \in \mathcal{A}_{\mathbb{Z}^n,W}$. Let T' be the regular tree provided by Lemma 7.5. By definition of $L_1(T')$, the set $\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}$ is open. Thus Lemma 7.8 can be strengthened to

$$\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)} = W^\circ \setminus A.$$

Therefore Lemma 5.6 shows that

$$\beta\left(\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}\right) \geq \beta(W^\circ).$$

By the definition of a normal tree, together with Lemma 7.5, we have

$$|L_1(T')| \leq 2^{h(T')/2} \leq 2^{h(T)+n}.$$

We now combine the last two inequalities with Lemma 7.7:

$$\beta(W^\circ) \leq |L_1(T')| \cdot 2 \cdot 3^{n+h(T')} \leq 2^{h(T)+n+1} \cdot 3^{3n+2h(T)},$$

hence

$$h(T) \geq c_1(\log_2 \beta(W^\circ) - 1) - c_2 n.$$

This proves Theorem 3.2′ directly and Theorem 3.2 as a consequence (since $\beta(W^\circ) \geq \hat{\beta}(W)$).

7.7. Proof of Theorem 3.5. Consider a $(\Re, \Re, \{+, -, \times, /, \chi_{\mathbb{Z}}\})$-RAM program P that recognizes W in time t for integer inputs. W is assumed to be scale invariant and rationally dispersed. Let U be the set of real-valued inputs accepted by P. Correctness of the program means $U \cap \mathbb{Z}^n = W \cap \mathbb{Z}^n$. Let T, V and B be given by Lemma 6.4. Let T' be a regular computation tree obtained from T (and recognizing the same set V) such that $h(T') \leq 2t + 2n$.

LEMMA 7.9. *There is a closed boundary set A such that*

$$\left(\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}\right) \setminus A = W \setminus A.$$

Proof. Let H be the set of $\mathbf{x} \in \Re^n$ that satisfy $\psi(p_\nu)(\mathbf{x}) \cdot \psi(q_\nu)(\mathbf{x}) = 0$ for some branching node ν in T'. H is an algebraic variety and, by the regularity of T', is nontrivial. Since the operator ψ yields homogenous polynomials, H is scale invariant. Let

$A = B \cup H \cup \{\mathbf{0}\}$. Then A is a scale-invariant, closed boundary set. Let $\mathbf{x} \in \Re^n \setminus A$. Then $\mathbf{x} \in S_{\phi(T'),\phi(l)}$ for $l \in L_1(T') \cup L_0(T')$. We wish to prove that $\mathbf{x} \in W$ if and only if $l \in L_1(T')$.

We consider first the case $\mathbf{x} \in W$. The set $S_{\phi(T'),\phi(l)}$ is open (it is defined by strict inequalities), so $S_{\phi(T'),\phi(l)} \setminus A$ is open too, and $\mathbf{x}$ belongs in it. Hence we can choose $\varepsilon > 0$ such that the ε-neighborhood of $\mathbf{x}$ will be contained within this set. Since W is rationally dispersed, we can pick $\mathbf{y} \in \mathbb{Q}^n$ such that $\|\mathbf{y} - \mathbf{x}\| < \varepsilon$ and $\mathbf{y} \in W$. Let $N_0 > 0$ be an integer such that $N_0\mathbf{y} \in \mathbb{Z}^n$. Since W is scale invariant, we have, for all integer $k > 0$, $kN_0\mathbf{y} \in W$; hence also $kN_0\mathbf{y} \in U$. Since $\mathbf{y} \notin A$, in particular $\mathbf{y} \notin B$, and by Lemma 6.4 there is an integer $a > 0$ such that, for all $k \in \mathbb{Z}^+, kaN_0\mathbf{y} \in V$. We now choose an integer $N = kaN_0$ with k large enough to make, for all $\nu \in \xi_{T,l}$,

$$\operatorname{sign}(r_\nu(N\mathbf{y})) = \operatorname{sign}\left(\frac{1}{N^d} r_\nu(N\mathbf{y})\right) = \operatorname{sign}(\psi(r_\nu)(\mathbf{y}))$$

(such k exists by the definition of ψ). The above equation implies that $N\mathbf{y}$ will follow the same path in T' as $\mathbf{y}$ does in $\phi(T')$. Since $\mathbf{y}$ reaches leaf $\phi(l)$ in $\phi(T')$, $N\mathbf{y}$ will reach leaf l in T'. Since $N\mathbf{y} \in V \cap \mathbb{Z}^n$, we have $l \in L_1(T')$.

For $\mathbf{x} \notin W$, similar arguments show that $l \in L_0(T')$, hence $\mathbf{x} \notin \bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}$. The conclusion of the lemma folows. □

Combining the last result with Lemmas 5.3 and 5.5, we have

$$\beta\left(\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}\right) \geq \beta\left(\left(\overline{\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}}\right)^\circ\right) = \beta((\overline{W})^\circ)$$

and by applying Lemma 7.7, as in the preceding proof, we obtain

$$\beta((\overline{W})^\circ) \leq 2^{t+n+1} \cdot 3^{3n+2t},$$

completing the proof of Theorem 3.5.

7.8. Proof of Theorem 3.8. The proof of Theorem 3.8 is very similar to the last proof. However, since there are some technical differences, we give the proof in whole.

LEMMA 7.10. *Let $W \subseteq \Re^n$ be scale invariant and rationally dispersed, H a nontrivial algebraic variety in $\Re^n$. For any $\mathbf{x} \in \Re^n \setminus (H \cup \{\mathbf{0}\})$ and $\varepsilon > 0$, there exists a rational point $\mathbf{y} \neq \mathbf{0}$ and an integer $N_0 > 0$, such that $\|\mathbf{y} - \mathbf{x}\| < \varepsilon$, $N_0\mathbf{y}$ has integer coordinates, and for every positive multiple kN_0 we have $kN_0\mathbf{y} \notin H$ and $kN_0\mathbf{y} \in W \iff \mathbf{x} \in W$.*

Proof. Without loss of generality, let us assume that H contains the origin. Let $\mathbf{x} \in \Re^n \setminus H$. The latter is an open set, so there is a neighborhood $\mathcal{N}$ of $\mathbf{x}$ that is disjoint from H. Assume that $\mathbf{x} \in W$ (the other case is treated analogously). Since W is rationally dispersed, W contains rational points arbitrarily close to $\mathbf{x}$. Choose $\mathbf{y}$ as such a point that belongs to $\mathcal{N}$, and satisfies $\|\mathbf{y} - \mathbf{x}\| < \varepsilon$. Then $\mathbf{y} \in W \setminus H$. Let L be the line $\{\lambda\mathbf{y}, \lambda \in \Re\}$. Consider the set $N_H = \{\, N > 0 : N\mathbf{y} \in H \,\}$. We have $N_H\mathbf{y} \subseteq L \cap H$. Since $\mathbf{y} \notin H$, L cannot be contained in H. Hence by Lemma 5.21, $L \cap H$ is finite; hence also $N_H\mathbf{y}$ is. Note that since $\mathbf{y}$ is rational, there are infinitely many $N > 0$ such that $N\mathbf{y} \in \mathbb{Z}^n$. Choose N_0 to be such a number, greater than the maximum of N_H. The statement of the lemma is fulfilled. □

Consider a $(\mathbb{Q}, \mathbb{Q}, \{+, -, \times, \lfloor\ \rfloor\})$-RAM program P that recognizes W in time t for integer inputs. W is assumed to be scale invariant and rationally dispersed. Let

U be the set of rational input vectors accepted by P. Correctness of the program means $U \cap \mathbb{Z}^n = W \cap \mathbb{Z}^n$. Let T, V, H, and a be given by Lemma 6.7. Let T' be a regular computation tree obtained from T (and recognizing the same set V) such that $h(T') \leq 2t + 2n$.

LEMMA 7.11. *There is a closed boundary set A such that*

$$\left(\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)} \right) \setminus A = W \setminus A\,.$$

Proof. Let H_1 be the set of $\mathbf{x} \in \Re^n$ that satisfy $\psi(p_\nu)(\mathbf{x}) \cdot \psi(q_\nu)(\mathbf{x}) = 0$ for some branching node ν in T'. H_1 is an algebraic variety and, by the regularity of T', is nontrivial. Let $A = H \cup H_1$. Then A is a closed boundary set. Let $\mathbf{x} \in \Re^n \setminus A$. Then $\mathbf{x} \in S_{\phi(T'),\phi(l)}$ for $l \in L_1(T') \cup L_0(T')$. We wish to prove that $\mathbf{x} \in W$ if and only if $l \in L_1(T')$.

First, we assume $\mathbf{x} \in W$. The set $S_{\phi(T'),\phi(l)} \setminus A$ is open and contains $\mathbf{x}$, so we can choose $\varepsilon > 0$ such that the ε-neighborhood of $\mathbf{x}$ will be contained within this set. Pick $\mathbf{y}$ and N_0 as provided by Lemma 7.10. Then $\mathbf{y} \in S_{\phi(T'),\phi(l)}$, and for all $k > 0$, $kN_0\mathbf{y} \in W \setminus H$; this implies, by Lemma 6.7, that

$$\forall k \in \mathbb{Z}^+ \quad kaN_0\mathbf{y} \in U \iff kaN_0\mathbf{y} \in V\,. \tag{7}$$

We now choose an integer $N = kaN_0$ with k large enough to make, for all $\nu \in \xi_{T,l}$,

$$sign(r_\nu(N\mathbf{y})) = sign\left(\frac{1}{N^d} r_\nu(N\mathbf{y})\right) = sign(\psi(r_\nu)(\mathbf{y})).$$

This implies that $N\mathbf{y}$ will follow the same path in T' as $\mathbf{y}$ does in $\phi(T')$. Since $\mathbf{y}$ reaches leaf $\phi(l)$ in $\phi(T')$, $N\mathbf{y}$ will reach leaf l in T'. Since $N\mathbf{y} \in W \cap \mathbb{Z}^n$, we also have $N\mathbf{y} \in U \cap \mathbb{Z}^n$, and (7) shows that $N\mathbf{y} \in V$. Thus $l \in L_1(T')$.

For $\mathbf{x} \notin W$, similar arguments show that $l \in L_0(T')$; hence $\mathbf{x} \notin \bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}$. The conclusion of the lemma follows. ∎

Combining the last result with Lemmas 5.3 and 5.5, we have

$$\beta\left(\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)} \right) \geq \beta\left(\left(\overline{\bigcup_{l \in L_1(T')} S_{\phi(T'),\phi(l)}} \right)^\circ \right) = \beta((\overline{W})^\circ),$$

and by applying Lemma 7.7, as in the preceding proofs, we obtain

$$\beta((\overline{W})^\circ) \leq 2^{h(t)+n+1} \cdot 3^{3n+2h(T)} \leq 2^{2t+n+1} \cdot 2^{3n+4t},$$

completing the proof of Theorem 3.8.

8. Conclusion. In this paper we showed how topological arguments, previously used to achieve lower bounds for algebraic computation trees, can be applied to random access machines. We considered machines with standard memory access (using integer addresses) and nonstandard models that may use arbitrary real addresses. We considered the case of real-number input as well as that of integer input, and instruction sets $\{+, -, \times, /\}$ and $\{+, -, \times, \lfloor\ \rfloor\}$. Our first lower bounds for these models (Theorems 3.3 and 3.6, resp.) show that when problems with real input are considered, it is hard to make effective use of integer addresses within these models. This is one motivation for studying the nonstandard model.

If the model is extended with the integrality predicate (which is not an algebraic function), some problems may be solved faster without resorting to noninteger addresses. For example, Uniform Gap can be solved in linear time. Theorem 3.4 gives a class of problems that cannot benefit this way. An even stronger extension is adding the truncation ("floor") operation. Together with division, it allows a linear-time solution to Max Gap and to Min Gap. Obtaining a better characterization of the power of this instruction set is a challenging open problem. Another open problem is to close the gap that remains between even the algebraic RAM and computation trees regarding bounds for the knapsack and generalized knapsack problems. (In fact, even the computation-tree bound is not known to be tight.)

Finally, another extension of this research is to use other topological properties of the set, besides the number of connected components. Yao [20] obtained new lower bounds for algebraic computation trees using higher Betti numbers of W (the number of path-connected components is the 0th Betti number). In particular he obtained an $\Omega(n \log(n/k) - cn)$ lower bound for the following.

EXAMPLE 12 (generalized element distinctness). *Given $x_1, x_2, \ldots, x_n$, decide whether some k of these n numbers are equal.*

This result can be easily applied to the models of Theorems 3.3 and 3.6, since their proofs give a direct reduction of the RAM program to an algebraic computation tree. However, for the other models, making use of this result for RAM programs is still an open problem.

9. Appendix. An algorithm using truncated division. The Min Gap decision problem calls for deciding whether, in an array of input numbers, there are two whose difference is bounded by an input value t. In section 3, we have shown that the algebraic RAM, as well as the RAM with floor but without division, require $\Omega(n \log n)$ time to solve this problem even with $\mathcal{A} = \Re$. This appendix gives a simple linear-time algorithm to solve this problem on a $(\Re, \mathbb{N}, \{+, -, \times, /, \lfloor\ \rfloor\})$-RAM with an initially zero memory.

ALGORITHM 1. MIN GAP DECISION.
Input: Real numbers $x_1, \ldots, x_n$ and $t > 0$.
Output: If there is a pair $i \neq j$ such that $|x_i - x_j| \leq t$, the algorithm outputs such a pair; otherwise it asserts that there is none.
Method: We partition the real line into intervals of length t and use the following fact: If there is a pair of points from the given set whose difference is bounded by t, then these points occupy either the same interval or two consecutive ones. Furthermore, if no pair satisfies the first condition, detecting the second is easy since each interval contains at most a single point.

A detailed program appears in Figure 2. In this program, we assume that uses of indirect addressing in the algorithm do not collide with any variables; assumptions which can be removed with standard programming techniques. Finally, we assume

the points x_i have been translated, if necessary, so that their minimum is zero. The notation $\langle a \rangle$ is used for the memory cell whose address is a.

```
for i = 1, ..., n do
   a_i ← ⌊x_i / t⌋ + 1
   if ⟨a_i⟩ ≠ 0 then
      output i, ⟨a_i⟩ and halt
   else
      if ⟨a_i − 1⟩ ≠ 0 then
         d ← |x_i − x_⟨a_i − 1⟩|
         if d ≤ t then
            output i, ⟨a_i − 1⟩ and halt
         end if
      end if
      if ⟨a_i + 1⟩ ≠ 0 then
         d ← |x_i − x_⟨a_i + 1⟩|
         if d ≤ t then
            output i, ⟨a_i + 1⟩ and halt
         end if
      end if
      ⟨a_i⟩ ← i
   end if
end for
reject
```

FIG. 2.

REFERENCES

[1] A. M. BEN-AMRAM, *On the Power of Random Access Machines*, Ph.D. thesis, Tel-Aviv University, Tel-Aviv, Israel, 1994, 1995.

[2] M. BEN-OR, *Lower bounds on algebraic computation trees*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 80–86.

[3] N. H. BSHOUTY, *Lower bounds for the complexity of functions in a realistic RAM model*, in Proceedings of the Israel Symposium on the Theory of Computing and Systems, Haifa, Israel, 1992, pp. 12–23.

[4] N. H. BSHOUTY, Y. MANSOUR, B. SCHIEBER, AND P. TIWARI, *Fast exponentiation with the floor operation*, Comput. Complexity, 2 (1992), pp. 244–255.

[5] E. DITTERT AND M. J. O'DONNELL, *Lower bounds for sorting with realistic instruction sets*, IEEE Trans. Comput., C-34 (1985), pp. 311–317.

[6] D. DOBKIN AND R. J. LIPTON, *A lower bound of $\frac{1}{2}n^2$ on linear search programs for the knapsack problem*, J. Comput. System Sci., 16 (1978), pp. 413–417.

[7] B. JUST, F. MEYER AUF DER HEIDE, AND A. WIGDERSON, *On computations with integer division*, RAIRO Informatique Théorique, 23 (1989), pp. 101–111.

[8] P. KLEIN AND F. MEYER AUF DER HEIDE, *A lower time bound for the knapsack problem on random access machines*, Acta Inform., 19 (1983), pp. 385–395.

[9] K. LÜRWER-BRÜGGEMEIER AND F. MEYER AUF DER HEIDE, *Capabilities and complexity of computations with integer division*, in Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 665, Springer-Verlag, New York, 1993, pp. 463–472.

[10] Y. MANSOUR, B. SCHIEBER, AND P. TIWARI, *Lower bounds for computations with the floor operation*, SIAM J. Comput., 20 (1991), pp. 315–327.

[11] Y. MANSOUR, B. SCHIEBER, AND P. TIWARI, *A lower bound for integer greatest common divisor computations*, J. ACM, 38 (1991), pp. 453–471.

[12] F. Meyer auf der Heide, *Lower bounds for solving linear diophantine equations on random access machines*, J. ACM, 32 (1985), pp. 929–937.

[13] F. Meyer auf der Heide, *On genuinely time bounded computations*, in Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 349, Springer-Verlag, New York, 1989, pp. 1–16.

[14] J. Milnor, *On the Betti numbers of real varieties*, Proc. Amer. Math. Soc., 15 (1964), pp. 275–280.

[15] W. Paul and J. Simon, *Decision trees and random access machines*, in Logic and Algorithmic, Monograph. Enseign. Math. 30, Université de Genève, Geneva, 1982, pp. 331–340.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction,* Springer-Verlag, New York, 1985, corrected 5th printing 1993.

[17] M. Steele and A. Yao, *Lower bounds for algebraic decision trees*, J. Algorithms, 3 (1982), pp. 1–8.

[18] R. Thom, *Sur l'homologie des variétés algébriques réelles*, in Differential and Combinatorial Topology, S. S. Cairns, ed., Princeton University Press, Princeton, NJ, 1965, pp. 255–265.

[19] A. C-C. Yao, *Lower bounds for algebraic computation trees with integer inputs*, SIAM J. Comput., 20 (1991), pp. 655–668.

[20] A. C. Yao, *Decision tree complexity and Betti numbers*, J. Comput. System Sci., 55 (1997), pp. 36–43.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 762–776

SUCCINCT REPRESENTATION OF BALANCED PARENTHESES AND STATIC TREES*

J. IAN MUNRO† AND VENKATESH RAMAN‡

Abstract. We consider the implementation of abstract data types for the static objects: binary tree, rooted ordered tree, and a balanced sequence of parentheses. Our representations use an amount of space within a lower order term of the information theoretic minimum and support, in constant time, a richer set of navigational operations than has previously been considered in similar work. In the case of binary trees, for instance, we can move from a node to its left or right child or to the parent in constant time while retaining knowledge of the size of the subtree at which we are positioned. The approach is applied to produce a succinct representation of planar graphs in which one can test adjacency in constant time.

Key words. abstract data type, succinct representation, binary trees, balanced parenthesis, rooted ordered trees, planar graphs

AMS subject classifications. 68P05, 68Q65

PII. S0097539799364092

1. Introduction.

The binary tree is among the most fundamental of data structures. While it is often the case that substantial amounts of data are associated with each node, there are also important cases in which there is little enough so that the representation of the tree with two $\lg n$ [1] bit pointers per node is a substantial, or even dominant, component of the total storage cost of a system. As there are $C_n \equiv \binom{2n}{n}/(n+1)$ or about $2^{2n}/n^{3/2}$ binary trees on n nodes, $2n$ bits are necessary (ignoring logarithmic terms) to encode an arbitrary binary tree. It is also easy to come up with a $2n$ bit representation of an n-node binary tree using a balanced nested parenthesis sequence. (Do a preorder traversal of the tree outputting an open parenthesis when visiting a node for the first time and a matching closing parenthesis when visiting it again after visiting the nodes of its subtree. Note that in this representation it is not possible to distinguish a node having a left child but not a right child from one having a right child but not a left child, but we will deal with this in section 3.2). A greater challenge is to come up with an optimal representation where the navigation around the tree, or for that matter the parentheses, can be done efficiently. Jacobson [13] proposed a method achieving the information theoretically optimal bound to within a lower order term. His interest, like ours, was in representing static data structures succinctly while still permitting the natural operations of navigating around the structure to be performed quickly. His encoding permitted the operations to move to the parent or either child of a given node, upon inspection of $\Theta(\lg n)$ bits. There are two shortcomings with this approach. First, the bits inspected are by no means localized. At one point a "binary search" of $\Theta(\lg^2 n)$ bits is per-

*Received by the editors November 18, 1999; accepted for publication (in revised form) July 9, 2001; published electronically November 13, 2001. Some of the results of this paper appeared in preliminary form in *Proceedings of the* 38*th Annual Symposium on Foundations of Computer Science (FOCS)*, Miami Beach, FL, 1997, pp. 118–126.

http://www.siam.org/journals/sicomp/31-3/36409.html

†Department of Computer Science, University of Waterloo, Waterloo, Canada N2L 3G1 (imunro@uwaterloo.ca).

‡The Institute of Mathematical Sciences, C.I.T. Campus, Chennai, India 600113 (vraman@imsc.ernet.in).

[1] We use lg to denote the ceiling of the logarithm base 2 and $\lg^2$ to denote its square.

formed. Hence, the runtime is certainly not constant, even if a single operation were to permit the inspection of $\lg n$ consecutive bits. While this difficulty can be overcome (by an approach of the first author appearing only in [6]), the second shortcoming is really the one which inspired our work. In a number of applications, the retention of the size of the subtree, rooted at the current node, is essential. For example, in using a binary trie as an index for full text search, one is given a query phrase and traces down the index trie following the bit pattern of the query. Where one runs out of query bits, the entire subtree rooted at the current node corresponds to the set of matches of the query in the text. When this subtree is large, it is crucial that the system be able to announce the number of matches without enumerating them all. For example, the word "the" occurs 2,744,347 times in the Oxford English Dictionary.

Clark and Munro [7] used a $3n$ bit encoding of a binary tree to demonstrate the viability of succinct tree representations for indexing large text files. The encoding they used, however, had three other important features: two good and one bad. First, it was based on explicitly giving the size of one subtree at each node; hence, keeping track of the size of a subtree was a by-product of moving down the tree. Second, the structural information needed at any time was, more or less, together. This meant that, if one considered a model under which a word is a sequence of $\lg n$ consecutive bits, the basic (arithmetic and bitwise logical) operations could be performed in constant time. The shortcoming of the technique was that it did not support efficient determination of the parent of a given node.

In this paper, we give an encoding for arbitrary binary trees on n nodes, using $2n + o(n)$ bits under this $\lg n$ bit word RAM model. As in [3], for convenience we count the space in terms of bits instead of words, and we assume that all standard operations on words can be performed in constant time. Our representation for binary trees supports all four operations: *left child*, *right child*, *parent*, and *size*. This method, along with all others mentioned, maps the nodes of the tree onto the integers $1, \ldots, n$, and hence all are appropriate for applications in which data is to be associated with nodes or leaves.

Despite the motivation for our work coming initially from the binary tree application, our technique focuses first on the problem of an optimal representation of balanced sequences of parentheses. As the class of such strings of length $2n$ is isomorphic to the class of binary trees on n nodes, it is convenient to let n denote the number of open parentheses, or half the string length. Jacobson [13] also considered this problem and proposed what seems to be a $10n + o(n)$ bit solution, a factor of 5 times the optimal. His solution supports the following operations:

- *findclose(i)*. Find the position of the closing parenthesis that matches the open parenthesis in position i.
- *findopen(i)*. Find the position of the open parenthesis that matches the closing parenthesis in position i.
- *excess(i)*. Find the difference of the number of open parentheses and the number of closing parentheses from the beginning to position i.

As with his solution to the tree representation problem, these operations require the inspection of $\Theta(\lg n)$ bits. Our solution uses space which is within a lower order term of the information theoretic $2n$ bits, and all operations can be performed in constant time in our model. One other operation is also necessary for our tree representation and so, again in constant time, we support the following:

- *enclose(i)*. Given a parenthesis pair whose open parenthesis is in position i, find its closest enclosing matching parenthesis pair, if it exists.

Throughout the paper, by the term "position" of a parenthesis, we mean the index to the location (in the given parenthesis sequence) of the parenthesis from the beginning. Given a bitstring, $rank(x)$, the rank of the bit at position x is the number of 1's up to and including the position x, and $select(i)$ is the position of the ith 1 in the bitstring. These two operations can be supported in constant time for an n bit string using $o(n)$ bits of extra space (see [13, 15]). When referring to elements stored in an array (or a sequence), we will speak of the array (sequence) as going from left to right.

The next section discusses our implementation of a sequence of balanced parentheses, and section 3 discusses our representations for static trees. Section 4 gives an efficient representation of bounded page number graphs, including planar graphs. Section 5 presents some concluding remarks and some recent work based on the preliminary version of this work.

2. Balanced parentheses. Here we are given a balanced sequence of parenthesis of length $2n$, so there are n open and n closing parentheses. We want to use an auxiliary storage of $o(n)$ bits to answer standard operations in constant time. Jacobson [13] first breaks the given sequence into blocks of length $\lg n$. Therefore, simply keeping $\Theta(\lg n)$ bits of information with each block contributes heavily to the lead term ($\Theta(n)$ term) of the amount of storage required. We get around the problem by using three levels of blocking. First, we break the sequence into "big blocks" of size $b = \lg^2 n$. This permits a constant number of references to be stored with each big block. It does, however, preclude scanning the entire block in constant time and thus forces the notion of a small block. Each big block is divided into "small blocks" of size $\lg^2 b = 4(\lg \lg n)^2$. As a *local* reference to a parenthesis in a big block requires only $2 \lg \lg n$ bits, such a reference to a single parenthesis can be stored for each small block. Finally, small blocks are further subdivided into "tiny blocks" of length $2 \lg \lg n$. These are so small that we can afford a single table with an entry for every possible distinct tiny block.

To help answer queries, we associate some auxiliary storage with each of these blocks. A similar but two level storage has been used to succinctly represent sets of a finite universe to support membership [3] and to represent bitstrings to support the operations rank and select [13, 15].

We gear our description of our auxiliary storage to answer the operation *findclose(i)* which finds the matching closing parenthesis for the open parenthesis in position i. Some of the other operations, such as *excess(i)*, are required along the way, while others, such as *enclose(i)*, can be implemented using the *findclose(i)* operation. We will specifically address these operations later.

Using the terminology of Jacobson, we call a parenthesis *far* if its matching parenthesis lies outside its block. A far open (closing) parenthesis is a *pioneer* if its matching parenthesis lies in a different block than that of the immediately previous (next) far open (closing) parenthesis in the sequence from the beginning. (Note that the matching parenthesis of a pioneer may or may not be a "pioneer" itself.) These two definitions are general enough that we can, and we will, use them with respect to any of the block sizes: big, small, or tiny. However, until otherwise noted the terms will be used in the context of big blocks. For example, the matching parenthesis of a far parenthesis lies outside its "big block." With each pioneer parenthesis, we keep the position of its matching parenthesis. For any other parenthesis, we will keep sufficient information to find

- its immediately preceding pioneer parenthesis in the sequence; and
- its matching parenthesis, given the matching parenthesis of its pioneer.

In our auxiliary storage structure, there are two types of information stored. In the first type, we keep some information with each block (big, small, or tiny). This is stored in three arrays, one indexed by the big blocks, one by the small blocks, and one by the tiny blocks. Each cell of each array contains all information stored with the corresponding block in an arbitrary but fixed sequence. The second type of information is stored only for specific parentheses or blocks. Each such information is stored in an efficient hash table (see [8, 19]), each cell of which consists of a position of the specific parentheses or the block, along with all the information stored with the corresponding parenthesis or the block. Given a position (of a parenthesis or a block), its membership can be found in constant time in such a hash table. For example, the position of the matching parenthesis for every pioneer parenthesis is stored in such a hash table, each cell of which consists of a position of a pioneer parenthesis along with the position of its matching parenthesis. We will call that table the pioneer table p. Given the position of a parenthesis, it can be checked in constant time, whether or not it is a pioneer, by checking for its membership in this hash table.

Now we begin the complete description of our auxiliary storage which consists of the following:

- With every pioneer parenthesis, we keep the position of its matching parenthesis in the table p.
- With every big block, we keep
  - the position of the immediately preceding pioneer (i.e., the pioneer immediately to the left of the block) and
  - the excess, namely, the number of open parentheses minus the number of closing parentheses up to the position of the first parenthesis, in the block.

  We will call the array containing this information B.
- With every small block within a big block, we keep
  - the position of the immediately preceding pioneer inside the big block (and a string of 0's if there is no pioneer to its left in the big block) and
  - the excess at that position starting from the beginning of its big block. (This value could be negative.)

  We will call the array containing this information b.
- With every small block containing a pioneer, we will keep a bitstring whose length equals the size of the small block. The ith bit in the bitstring is 1 if the corresponding parenthesis is a pioneer in the small block, and 0 otherwise. The hash table containing this information is called bp.

In addition, for every possible bitstring of length $4(\lg \lg n)^2$, and for every position in that string, we create a table entry. There are two values in each table entry: one giving the position of the preceding 1 in that string (a string of 0's if there is no 1 to the left of that position) and the other giving the excess (the number of 0's minus the number of 1's) in binary at the given position inside that string. (Again this value could be negative.) We will call this table st.

Note that we have not yet associated any storage with individual tiny blocks. We will do this later.

We claim that the storage used so far is $o(n)$ and is sufficient to find, for every far parenthesis, its target big block and the excess at its position in constant time.

2.1. Finding the target big block of the matching parenthesis. As Jacobson [13] noted, if there are b blocks, there are at most $2b-3$ pioneers, and so we have $O(n/\lg^2 n)$ pioneer parentheses. Hence, the hash table p that stores the location of

the matching parenthesis of every pioneer parenthesis takes $O(n/\lg n)$ bits. It is also clear that the total space occupied by the table B is $O(n/\lg n)$ bits. As the address of the previous pioneer inside the big block, and the excess inside the big block at that position, for every small block, take $O(\lg\lg n)$ bits, and the number of small blocks is $O(n/(\lg\lg n)^2)$, the total space to store these values in the table b is $O(n/\lg\lg n)$ bits. The number of small blocks with pioneers is at most $O(n/\lg^2 n)$, and so the hash table bp storing the bitstring of pioneers take at most $O(n/\lg n)$ bits. The total number of entries in the table st is $O((\lg n)^{4\lg\lg n}(\lg\lg n)^2)$, and each table entry takes $O(\lg\lg\lg n)$ bits for a total of $O((\lg n)^{4\lg\lg n}(\lg\lg n)^2\lg\lg\lg n)$ bits of storage for the table st.

Now we can find the preceding pioneer parenthesis for a given far parenthesis in position i in constant time as follows:

- Given the position i, first check, by consulting the hash table bp containing the (addresses of the small blocks having pioneers and the) bitstring of pioneers, whether the small block to which i belongs has a pioneer.
- If there is a pioneer in the small block, then a table lookup in the st table, at the position given by the bitstring of pioneers in that small block and its position i, gives the position of the previous pioneer if there is one to its left in the small block.
- If there is no pioneer in the small block to the left of i, then the appropriate position in the array b gives the position of the previous pioneer in that big block. If that is simply a string of 0's, then there is no pioneer in the big block to the left of this small block. In that case, the appropriate index in array B gives the position of the pioneer previous to the position i.

A completely analogous strategy finds the excess at the given position. Here we don't need to consult a hash table first, and we have to treat the open parenthesis as a 0 and a closing parenthesis as a 1 to index into the table st for the table look up. Three words (an entry in table st and appropriate locations in arrays b and B) have to be inspected and the resulting values summed up to get the answer. Once we find the preceding pioneer parenthesis of a given far parenthesis, we find the matching parenthesis of the pioneer from the pioneer hash table p. From that, we know the target big block of the matching parenthesis of the given far parenthesis. Thus we have the following lemma.

LEMMA 1. *Given a sequence of balanced parentheses of length $2n$, and a far parenthesis (i.e., one whose matching parenthesis lies outside its block), the target block of its matching parenthesis and the excess at its position can be found in constant time using $o(n)$ bits of auxiliary information.*

2.2. Finding the matching parenthesis of a far parenthesis. Let us call the "excess" value to which we have been referring the *left excess* and retain, also in $o(n)$ auxiliary storage, similar information regarding the "excess" of the reversal of the given sequence. Then the *right excess* at a position is defined and determined in the analogous way. More specifically, the right excess at a given position i is the number of closing parentheses minus the number of open parentheses starting from the end of the string to the position i. Now, given a far open parenthesis with left excess e at its position, its matching parenthesis in the target big block is the first (from the left) closing parenthesis with right excess e. (Note that there may be several parentheses with the same excess and the first one will necessarily be far.) To compute this information, we need something similar to the inverse of the excess operation: given an excess value and a block, we want to find the first far parenthesis with that excess value in that block.

We can first subtract from e, the right excess (from the end of the string until the beginning of the next block) stored with the next block. What we now require is the first far closing parenthesis with the residual right excess in the target block. Though arrays B and b have the excess information for each of the big and small blocks, it is not enough to simply consult them and/or scan a small block to find the answer. To handle this problem, we create another auxiliary storage. Note that the excess within a big block is at most $\lg^2 n$. With each big block, we keep an array f of answers (the first far closing parenthesis with that excess) for every $\sqrt{\lg n}$ excesses inside the block. This requires $\lg \lg n$ bits for each answer and thus $O(n \lg \lg n / \sqrt{\lg n})$ bits in total. Now to find the parenthesis with a specified excess not a multiple of $\sqrt{\lg n}$, one has to scan the portion between the answers corresponding to the closest multiples of $\sqrt{\lg n}$ to the given excess. If this portion between the two consecutive answers is of size at most $(\lg n)/2$, then we can solve this problem by keeping a table f_1 of answers for every such string for every position. Since there are $\sqrt{n}$ strings of size $(\lg n)/2$, this takes $O(\sqrt{n} \lg n \lg \lg n)$ bits in total. If, on the other hand, the portion between two consecutive answers is larger than $(\lg n)/2$, then we keep the answer for *every* intermediate excess (between the two closest $\sqrt{\lg n}$ multiples) for each such portion. There are $O(\sqrt{\lg n})$ answers for every such portion requiring a total of $O(\sqrt{\lg n} \lg \lg n)$ bits for each such portion. Since there can be at most $2n/\lg n$ such portions, the total space requirement for these values is at most $O(n \lg \lg n / \sqrt{\lg n})$ bits. This information can be stored in an efficient hash table f_3, one for each block, where each cell of the hash table contains the position of the "large" portion (portion of length more than $(\lg n)/2$) between two consecutive answers, along with $O(\sqrt{\lg n} \lg \lg n)$ bits for each such portion. Since there are at most $\lg^{1.5} n$ portions (between two consecutive answers) for each block, the position information for each portion takes only $1.5 \lg \lg n$ bits. Note that at most $2 \lg n$ such portions are large in each block.

Thus in constant time, we can find the matching closing parenthesis for a far open parenthesis using $o(n)$ auxiliary bits of space. A completely analogous strategy finds the matching open parenthesis for a far closing parenthesis. Along the way, we also saw that we can find the excess value at any position in constant time. Thus we have the following lemma.

LEMMA 2. *Given a sequence of balanced parentheses of length $2n$, and a far parenthesis (i.e., one whose matching parenthesis lies outside its block), the position of its matching parenthesis can be found in constant time using $o(n)$ bits of auxiliary information.*

Now we need only to explain how to implement the enclose operation and to deal with parentheses which are not far.

2.3. The enclose operation. Here we outline a method of finding the closest enclosing parentheses for a given matching parenthesis pair at positions i and j. Suppose the parenthesis at position i has left excess e at that position. If e is 0, then there is no enclosing parenthesis. Otherwise, the closing parenthesis of the answer (the closest enclosing parenthesis) is the first far closing parenthesis in the block of j (or its next block having a far parenthesis) with right excess $e - 1$. This can be found, by the method described earlier, to find the first far closing parenthesis with a given excess and a target block. Sometimes the excess $e - 1$ may not be in the block containing j. For this, with every block, we keep a pointer to the far parenthesis that has right excess one less than the minimum in that block. This requires an additional $O(n/\lg n)$ bits. Once we find the closing parenthesis in the answer, its matching parenthesis can be found in constant time using the *findopen* operation. Thus we

have the following lemma.

LEMMA 3. *Given a sequence of balanced parentheses of length* $2n$*, the "enclose(i)" operation, that finds the closest enclosing parentheses for the given matching parenthesis pair whose open parenthesis is at position* i*, can be supported in constant time using* $o(n)$ *bits of auxiliary information.*

2.4. Checking if a parenthesis is far. To check whether a given parenthesis has its matching pair inside a big block, i.e., to check whether it is far (with respect to a big block), we run exactly the procedure outlined in sections 2.1 and 2.2 within a big block with the "small blocks" playing the role of "big blocks" and the "tiny blocks" playing the role of "small blocks." There is one difference though, and that is that the initial entire sequence is a balanced sequence, whereas the sequence inside the big block may not be. This can be easily taken care of by imagining two blocks, one before and one after the big block, with the required number of open and closing parentheses, respectively, to make the augmented sequence balanced. There is no need to keep any auxiliary storage with these imaginary blocks. Furthermore, whenever a parenthesis is found to have its matching parenthesis in the imaginary blocks, then we know that it is far. The rest of the argument goes through as before. For completion, we give a complete description and the auxiliary storage count for this below, omitting just the names of individual auxiliary tables and arrays.

We first fix a big block (the big block in which the given parenthesis resides) and speak of "far" and "pioneer" parentheses with respect to the small blocks within the big block. Note that the sequence inside the big block will, in general, not be balanced; and so we *imagine* augmenting it with a block preceding it, and one succeeding it, to make it balanced. Therefore, the parentheses whose matching pairs lie in the imaginary blocks are the far ones in which we are interested. In fact, some of these will be pioneers with respect to the small blocks. In addition to the storage described earlier, our auxiliary storage is analogous to that previously described for the full structure and consists of the following. (Note that in the discussion below 'far' and 'pioneer' are with respect to the small blocks within the big block.)

- With every pioneer parenthesis, store the position of its matching parenthesis. For those pioneer parentheses whose matching pairs are not in the big block, i.e., for those that are far, we simply indicate it by a special bit. If we hit such a parenthesis, then we immediately know that it is far.
- With every small block, we keep
  - the position of the immediately preceding pioneer in the big block (i.e., the immediate pioneer to the left of the block) and
  - the excess, namely, the number of open parentheses minus the number of closing parentheses up to the position of the first parenthesis, in the big block.
- With every tiny block within a small block, we keep
  - the excess at that position starting from the beginning of the small block it belongs to and
  - the position of the immediately preceding pioneer inside the small block (and 0 if there is no pioneer to its left in the small block)
- With every tiny block containing a pioneer, we will keep a bitstring whose length is the size of the tiny block. The ith bit in the bitstring is 1 if the corresponding parenthesis is a pioneer in the small block, and 0 otherwise.

Furthermore, for every possible bitstring of length $2(\lg \lg n)$, and for every position in that string, we create a table entry. There are two values in each table entry: one

giving the position in that string of the preceding 1 in that string (0 if there is no 1 to the left of that position) and the other giving the excess (the number of 0's minus the number of 1's) at the given position inside that string. (This value could be negative.)

We show that the storage used is $o(n)$, and it suffices to identify in constant time whether a given parenthesis is far with respect to its big block.

As there are $O(\lg^2 n/(\lg\lg n)^2)$ small blocks inside a big block, there are $O(n/(\lg\lg n)^2)$ pioneer parentheses in total. The space to store the matching parenthesis of each pioneer parenthesis is $O(\lg\lg n)$ bits, thus requiring a total of $O(n/\lg\lg n)$ bits for the hash table storing the addresses of these pioneers. It is also clear that the total space for the values associated with every small block is also $O(n/\lg\lg n)$. As the values (address of the previous pioneer inside the small block and the excess inside the small block) at the tiny blocks take $O(\lg\lg\lg n)$ bits each, and the number of tiny blocks is $O(n/\lg\lg n)$, the total space to store these values is $O(n\lg\lg\lg n/\lg\lg n)$ bits. The number of tiny blocks with pioneers is at most $O(n/(\lg\lg n)^2)$, and so the bitstring of pioneers take at most $O(n/\lg\lg n)$ bits in the hash table storing these bitstrings. The total number of table entries is $O(\lg^2 n\lg\lg n)$, and each table entry takes $O(\lg\lg\lg n)$ bits for a total of $O(\lg^2 n\lg\lg n\lg\lg\lg n)$ bits.

A completely analogous procedure as described earlier can find the target small block, if it exists, for a far (with respect to small blocks) open parenthesis in constant time. If the target block happens to be one of those imaginary blocks, we are through. Otherwise, given the excess and the target small block, to find the answer within the small block, a similar strategy used for the big blocks require another auxiliary storage of $o(n)$ bits. The size of each big block is $b = \lg^2 n$ and the size of each small block is $\lg^2 b$. Therefore, if we keep an auxiliary storage for answers analogous to the one described earlier, by replacing n by b, then the required storage will be $o(b)$ bits for each big block, thus resulting in $o(n)$ bits overall. (This turns out to be $O(n\lg\lg\lg n/\sqrt{\lg\lg n})$.)

To check whether a given parenthesis has its matching pair inside a small block (of size $4(\lg\lg n)^2$), we can have a table of all such parenthesis strings and answers for each position in that string. This requires only $O((\lg n)^{4\lg\lg n}(\lg\lg n)^2\lg\lg\lg n)$ bits. Thus we have the following lemma.

LEMMA 4. *Given a sequence of balanced parentheses of length $2n$, and an open or closing parenthesis, it can be checked in constant time using $o(n)$ bits of auxiliary information whether the parenthesis is far, i.e., whether its matching parenthesis is outside its block.*

2.5. Putting things together. To summarize, given an open parenthesis, we first check whether it has a matching pair in its small block using a table of answers. If not, we check whether it has a matching pair in its big block using the procedure described in section 2.4 using the small and tiny block divisions. Otherwise, it is far and so we use the procedure outlined in the proof of Lemma 2 using the big and small block divisions to find its matching parenthesis. A completely analogous strategy works to find the excess value at each position. Thus we have the following theorem.

THEOREM 1. *Given a sequence of balanced parentheses of length $2n$, using $o(n)$ auxiliary bits, we can perform the operations findclose(i), findopen(i), excess(i), and enclose(i) in constant time.*

3. Static trees.

3.1. Rooted ordered trees. To represent a general rooted ordered tree (where each node could have an unbounded number of children), we use the isomorphism

between general ordered trees and balanced parenthesis expressions. Here, in performing a preorder traversal of the general tree starting from the root, we write an open parenthesis when the node is first encountered, going down the tree, and then a closing parenthesis while going up after traversing its subtree. For example, the rooted ordered tree of Figure 3.1(b) is represented by the parenthesis sequence in Figure 3.1(c). With this representation, we make the following observations. We represent a node by its corresponding open parenthesis.

- The parent of a given node is given by the closest parenthesis pair enclosing the parenthesis pair corresponding to the given node.
- The subtree of a given node x consists of all nodes represented by parenthesis pairs in the sequence starting from the open parenthesis corresponding to x to its matching closing parenthesis. Hence, the subtree size is simply half the difference between the indices of x's closing parenthesis and open parenthesis (both end points included).
- The ith child of a node x can be obtained as follows. If the parenthesis next to x's open parenthesis is a closing parenthesis, then x has no children. Otherwise, that open parenthesis next to x's and its matching closing parenthesis form the first child of x, the next parenthesis (if it is open) and its matching parenthesis form the second child of x, and so on. In this process, using the $findclose$ operation at most $i-1$ times starting from the open parenthesis next to that of x, we find the open parenthesis of the ith child or find that x has no ith child.

  In a similar fashion, given a node x, we can find its left or right sibling in constant time as follows. First apply $findclose(x)$ operation to find the matching parenthesis of the open parenthesis corresponding to x. If the parenthesis next to that is an open parenthesis, then it is the parenthesis corresponding to x's right sibling. Similarly, if the parenthesis before the open parenthesis corresponding to x is a closing parenthesis, its matching parenthesis gives the parenthesis corresponding to x's left sibling.

Note that we require that a node be given by the position of its open parenthesis in the sequence. Given an open parenthesis at position p in the sequence we can find, in constant time, the (preorder) index of the node corresponding to it as follows. The preorder index of the corresponding node is simply the number of open parentheses up to but not including position p. (The root has preorder index 0.) This can be found in one of two ways: we can simply use the rank function (previously defined as the number of 1's up to a given position, and detailed in [13, 6]) on the given sequence and find the rank of the current open parenthesis. The other option is to use the excess operation and find the excess e at p. If there are i open parentheses and j closing parentheses up to position p, then $i+j=p$, and $i-j=e$. Therefore, we can compute i if we know p and e. Similarly, the position of the open parenthesis corresponding to a node whose preorder index is i can be obtained by a select operation on the string representing the tree. More precisely, it is the $(i+1)$st open parenthesis in the parenthesis representation of the tree.

Thus we have the following theorem.

THEOREM 2. *Given a static rooted ordered tree on n nodes, we can represent it using $2n + o(n)$ bits in such a way that given a node x (by its preorder index), we can find its parent and its subtree size in constant time and find its ith child in $O(i)$ time.*

3.2. Binary trees. A binary tree is not just a special case of an ordered rooted tree, as there is a distinction between a node having a left child but no right child

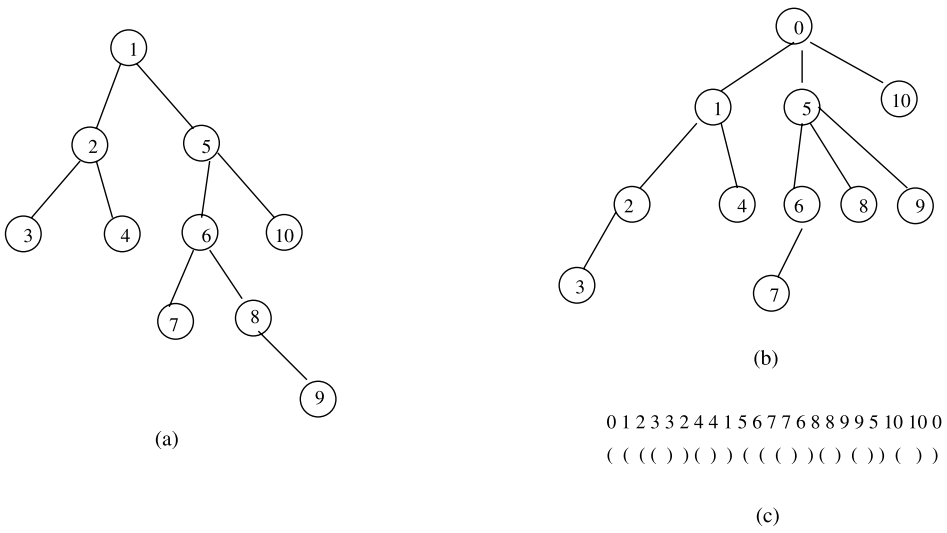


FIG. 3.1. (a) *The given binary tree on* 10 *nodes.* (b) *Equivalent rooted ordered tree.* (c) *The parenthesis representation.*

and one with a right child but no left child. We can, however, take advantage of a well-known isomorphism between these classes. Here the ordered tree has a root which does not correspond to any node in the binary tree. Beyond this, the left child of a node in the binary tree corresponds to the leftmost child in the ordered tree of the corresponding node, and the right child in the binary tree corresponds to the next sibling to the right in the ordered tree. As a consequence, the root of the binary tree corresponds to the leftmost child of the root of the ordered tree. (See, for example, Figures 3.1(a) and 3.1(b).) This isomorphism is often used to map the ordered tree to a binary tree and thus permit the representation of the ordered tree in nodes of a fixed size (i.e., two pointers). We use the isomorphism in the opposite direction as, together with the representation for rooted ordered trees, it enables us to store subtrees in contiguous segments and thus infer the size of a subtree.

We can therefore represent the given binary tree by this sequence of balanced parentheses of length $2n+2$. In fact, $2n$ bits are sufficient as the first open parenthesis along with its matching parenthesis is redundant. For example, the binary tree in Figure 3.1(a) is represented by the parenthesis sequence given in Figure 3.1(c).

By convention, we will denote a given node by the corresponding open parenthesis in the sequence. Then the following facts follow immediately for the binary tree representation by the parenthesis sequence:

- If the symbol following the open parenthesis corresponding to a given node is an open parenthesis, then it corresponds to the left child of the node. Otherwise, the left child is absent.
- To find the right child, if it exists, find the matching parenthesis of the open parenthesis corresponding to the given node. If the following symbol is an open parenthesis, then that is the parenthesis corresponding to the right child of the given node. Otherwise, there is no right child.
- To find the parent of a given node, if the symbol before the parenthesis corresponding to the node is an open parenthesis, then that is its parent. If there is no symbol before, then it is the root. Otherwise, the symbol before

is the closing parenthesis of its left sibling in the general ordered tree. Find its matching open parenthesis, that is, the parenthesis corresponding to its parent in the binary tree.

- The subtree rooted at a node x in the binary tree consists of all nodes in the subtrees rooted at x and at its right siblings in the general tree. In the parenthesis notation, this corresponds to all parentheses pairs from x's open parenthesis to (but not including) the closing parenthesis of x's parent in the general tree. However, x's parent parentheses are the closest enclosing parentheses of x's parentheses. Therefore, given x's open parenthesis, we can find the closest enclosing parentheses using the enclose operation, and thus we can obtain the parentheses corresponding to the nodes in the entire subtree rooted at x. The size of the subtree rooted at x is simply half the difference between the indices of the closing parenthesis of x's parent (excluding it) and the open parenthesis of x.

Thus we have the following theorem.

THEOREM 3. *Given a static binary tree on n nodes, we can represent it using $2n + o(n)$ bits such that given a node in the tree, we can find the left child, right child, and parent of the node if they exist and the size of the subtree rooted at that node in constant time.*

As before, here also a node must be given by the position of its open parenthesis in the sequence, or by its preorder index in the binary tree. It can be verified that we can obtain one from the other in constant time using rank and select operations.

4. Planar graphs. To represent a planar graph efficiently, it is tempting to use the fact [18] that any planar graph can be partitioned into at most three edge-disjoint spanning forests and use our succinct representation to represent the trees of each forest. However, note that our representation for trees is for unlabeled trees and it gives an implicit labeling (based on the preorder traversal of the tree). It is not clear that there is a partition of a planar graph into spanning forests which maintains consistency of the vertex labeling when the vertices of a forest are labeled by our representation. Hence, we resort to the approach of Jacobson [13] and use an edge partition of a planar graph which does maintain the same vertex labeling in each part of the partition. Such a partition is a partition of the edges into *pages* (defined below). Since in linear time planar graphs can be embedded in four pages [22], this leads to a succinct and quickly found representation for planar graphs.

Jacobson first gives a representation for one-page graphs and generalizes to graphs with bounded number of pages. His representation for one-page graphs uses his parentheses representation and hence is fairly inefficient in terms of space. Our parenthesis representation gives an immediate reduction in the higher order term for the number of bits used. Furthermore, as we have expanded the types of operations that can be handled in our implementation, we represent one page graphs differently, resulting in further reduction in space. We give details after defining the term "page." We use the terminology of Bernhart and Kainen [2], as did Jacobson [13].

A *k-page book embedding* of a graph $G = (V, E)$ is a printing order of V (a permutation specifying the ordering of the nodes along the spine of a book), plus a partition of E into k *pages*. The edges on a given page must not intersect, and all pages share the same printing order of the nodes. The *page number* of a graph G is the minimum number of pages in any book embedding of G.

4.1. One-page graphs. When the book is drawn so that the spine is horizontal, then the edges form a nesting structure as in a balanced string of parentheses (see Figure 4.1).

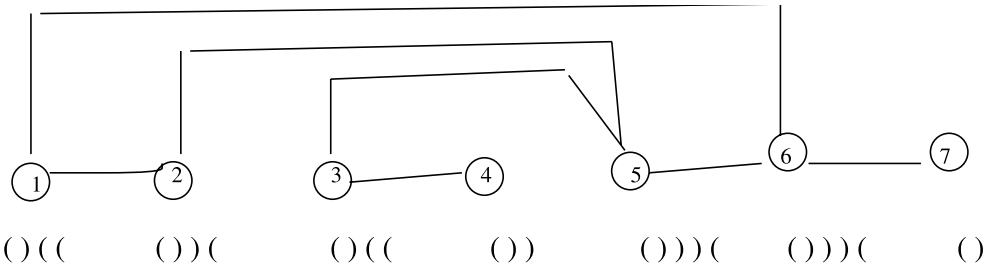


FIG. 4.1. *A one-page graph and its parenthesis representation. (The graph has seven vertices and seven edges.)*

Our representation by a balanced parentheses sequence is as follows. We represent every node and every edge by a matching pair of open and closing parentheses. We associate with every node j its representative matching pair of parentheses followed by a sequence of (single) parentheses for every edge incident on j. The open parenthesis corresponding to an edge $(i, j), i < j$, is in the sequence of parentheses associated with the node i, and the matching closing parenthesis is with the sequence associated with the node j. Thus with respect to a vertex i, we have the following: a matching pair of parentheses to represent node i, a sequence of closing parentheses in decreasing order of their other end points for edges incident on i whose other end points are less than i, followed by a sequence of open parentheses in increasing order of their other end points for edges incident on i whose other end points are greater than i. We concatenate the sequences with respect to every vertex in the order of vertices in the spine (see Figure 4.1). In our representation, a pair of adjacent symbols forms a matching pair of parentheses if and only if the pair represents a node in the one-page graph.

Clearly, this representation uses $2m + 2n$ bits, where m is the number of edges (which is $O(n)$) and n is the number of vertices in the graph. In fact, $2m + 2n - 2$ bits are sufficient as the first matching pair representing the first vertex is redundant.

In this representation, given the label of a node, its matching parenthesis pair in the sequence can be found using a select operation by denoting the adjacent parenthesis pairs (both open and closing) by a single 1 and the other (either open or closing) parenthesis by a 0. Conversely, given a parenthesis pair corresponding to a node, we can find its label by using a rank operation (which counts the number of 1s up to a position in a bit string) in a similar way. Furthermore, the neighbors of a vertex are immediately after its corresponding matching parentheses pair. Thus in time proportional to the degree of the vertex, we can find all the neighbors of a vertex using our parentheses sequence operations ($findopen()$ and $findclose()$). If we just want the degree of the vertex, we can use the rank and select operation to first find the parenthesis pair corresponding to the next vertex in the spine; the difference in the indices between the pairs corresponding to the two vertices gives the degree of the vertex.

To find whether nodes i and j ($i < j$ in the spine order) are adjacent in the graph, we can adapt the method of Jacobson [12] given for his representation. Simply find the matching parenthesis of the first open parenthesis after the pair corresponding to vertex i. If its position is before the position of the pair corresponding to vertex j, then i and j are not adjacent. If it corresponds to an edge incident on j, then i and j are adjacent. Otherwise, find the matching open parenthesis of the last closing

parenthesis after the pair corresponding to vertex j. If it corresponds to an edge incident on i, then i and j are adjacent. Otherwise, they are not. It can be verified that a constant number of rank and select computations are sufficient to perform these operations. Thus we have the following theorem.

THEOREM 4. *A one-page multigraph on n vertices and m edges can be represented using $2n + 2m + o(n + m)$ bits in such a way that the adjacency of a pair of vertices, and the degree of any given vertex, can be found in constant time, and the neighbors of a vertex can be produced in time proportional to the degree of the vertex.*

4.2. Graphs with more than one page. The generalization to graphs with more than one page is direct. We simply represent each page (which is a one-page graph) separately using the representation of section 4.1 and concatenate them together. As all pages share the same printing order of the vertices, the matching pair corresponding to each vertex appears in all pages. Therefore, if the graph has k pages, a vertex requires $2k$ bits, whereas an edge requires only 2 bits. Thus the parenthesis representation requires $2kn + 2m$ bits. The address of the beginning of each page can be stored separately in $k \lg n$ bits. The adjacency of a pair can be checked in $O(k)$ time by checking it in each page. Similarly, the degree of a vertex can be found in $O(k)$ time. The neighbors of a vertex can be listed out by following through the adjacency list in each page, in $O(k + d)$ time, where d is the degree of the vertex.

In particular for planar graphs (which can be embedded in four pages [22]), our representation requires $8n + 2m$ bits (ignoring lower order terms), requiring only a constant factor increase in the time spent. Thus we have the following theorem.

THEOREM 5. *A k page graph on n vertices and m edges can be represented using $2kn + 2m + o(nk + m)$ bits in such a way that adjacency between a pair of vertices and the degree of a vertex can be found in $O(k)$ time, and the neighbors of a vertex x can be listed out in $O(d(x)+k)$ time where $d(x)$ is the degree of the vertex x. In particular, a planar graph on n vertices and m edges can be represented using $8n + 2m + o(n)$ bits so as to test adjacency of a pair in constant time.*

5. Concluding remarks and further work. We have given the first representation for static binary trees on n nodes that uses the information theoretically optimum $2n + o(n)$ bits and still supports the *left child, right child, parent*, and the *subtree size* operations in constant time. Our representation simply uses the natural preorder traversal sequence. We also gave the first information theoretically optimum representation for a balanced sequence of parenthesis supporting several natural operations in constant time.

A related, but restricted, scheme for static binary trees has been implemented by Chupa [5]. The operations of moving up or down the tree are amazingly fast, though that scheme does not support finding the size of a subtree. We also note that a tree must have more than 10 million nodes before the total number of bits used, including those noteworthy "lower order terms," is less than three times the size of the tree. We feel the method discussed here will be comparable in implementation, while supporting a full range of motion in the tree.

Based on the preliminary report of our work, a number of related advances have been made. Our representation actually supports more operations than we have described here. For example, given a node, suppose we want to find the leftmost leaf in the subtree rooted at that node. In the parenthesis representation of the ordered tree corresponding to the binary tree, this is simply the node corresponding to the matching parenthesis pair in which the matching parentheses are adjacent and is the first such pair after the open parenthesis corresponding to the node. By considering

the adjacent matching parenthesis pairs as representing 1's and the other parentheses as representing 0's, this operation is simply a computation of rank and select in the parenthesis sequence. In a very similar way, we can find the rightmost leaf in the subtree rooted at any given node. Using these operations and our representation of static trees, Munro, Raman, and Rao [16] have demonstrated very efficient representations of suffix trees for text searching. The time and space bound for the suffix tree operations are further improved by Grossi and Vitter [9].

For rooted ordered trees (also known as ordinal trees), our representation takes $O(i)$ time to find the ith child of a given node, though every other navigational operation takes only a constant time. Benoit et al. [1] have reported a representation for ordinal trees in which this operation, besides the other constant time operations we support, can be performed in constant time. They have also generalized our binary tree representation for k-ary (cardinal) trees (where one or more subtrees of a node can be empty). Their representation uses $2n + n\lceil \lg k \rceil$ bits and supports searching for the child labeled i in $O(\lg \lg k)$ time while supporting all other navigational operations in constant time. Raman and Rao [20] have improved the time for searching the child labeled i to $O(\lg \lg \lg k)$ time while maintaining every other feature of the structure of Benoit et al. [1].

We have employed our parenthesis based operations to represent a planar graph on n vertices and m edges using $8n + 2m$ bits still supporting adjacency and degree queries in constant time. The space used is at most $14n$ (ignoring lower order terms), as m is at most $3n$. This is a drastic reduction from $64n$ bits required in Jacobson's representation of planar graphs [12] and is much closer to the $3.58m$ bits representation of planar graphs by Keeler and Westbrook [14], which does not allow efficient search operations. Using a more efficient representation for multiple parenthesis sequences, and a canonical ordering of planar graphs, Chuang et al. [4] have improved this space requirement to $2m + (5 + \frac{1}{k})n + o(m + n)$ bits for any constant $k > 0$ with the same query support. The bit count can be further reduced if only adjacency queries are supported in constant time, or if G is simple, triconnected, or triangulated [4]. For recent work on efficient representation of special classes of planar graphs (where the emphasis is on fast encoding and decoding rather than the navigational operations on the compressed representation), see [10] or [11].

More recently, Munro, Raman, and Storm [17] have found a succinct representation for dynamic binary trees where updates can be supported in $O(\lg^2 n)$ amortized time. The basic approach taken in that work is quite different from that used here. Furthermore, it is not clear that it can be adapted to deal with the auxiliary functions we can support with the approach developed here.

Acknowledgments. We thank Meng He for pointing out an error in the *double enclose* operation reported in the earlier version and the referees for their comments that greatly helped improve the presentation of the paper.

REFERENCES

[1] D. Benoit, E. D. Demaine, J. I. Munro, and V. Raman, *Representing trees of higher degree*, in Proceedings of the 6th Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci. 1663, Springer-Verlag, Berlin, 1999, pp. 169–180.

[2] F. Bernhart and P. C. Kainen, *The book thickness of a graph*, J. Combin. Theory Ser. B, 27 (1979), pp. 320–331.

[3] A. Brodnik and J. I. Munro, *Membership in constant time and almost minimum space*, SIAM J. Comput., 28 (1999), pp. 1627–1640.

[4] R. C. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu, *Compact encodings of planar graphs via canonical orderings and multiple parentheses*, in Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 118–129.

[5] K. Chupa, *Efficient Representation of Binary Search Trees*, Math Essay, University of Waterloo, Waterloo, Canada, 1997.

[6] D. R. Clark, *Compact Pat Trees*, Ph.D. thesis, University of Waterloo, Waterloo, Canada, 1996.

[7] D. R. Clark and J. I. Munro, *Efficient suffix trees on secondary storage*, in Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA), 1996, pp. 383–391.

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

[9] R. Grossi and J. Vitter, *Compressed suffix arrays and suffix trees with applications to text indexing and string matching*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC), 2000, pp. 397–406.

[10] X. He, M.-Y. Kao, and H.-I. Lu, *Linear-time succinct encodings of planar graphs via canonical orderings*, SIAM J. Discrete Math., 12 (1999), pp. 317–325.

[11] X. He, M.-Y. Kao, and H.-I. Lu, *A fast general methodology for information-theoretically optimal encodings of graphs*, SIAM J. Comput., 30 (2000), pp. 838–846.

[12] G. Jacobson, *Succinct Static Data Structures*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1988.

[13] G. Jacobson, *Space-efficient static trees and graphs*, in Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), 1989, pp. 549–554.

[14] K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Appl. Math., 58 (1995), pp. 239–252.

[15] J. I. Munro, *Tables*, in Proceedings of the 16th Foundations of Software Technology and Theoretical Computer Science (FST & TCS), Lecture Notes in Comput. Sci. 1180, Springer-Verlag, Berlin, 1996, pp. 37–42.

[16] J. I. Munro, V. Raman, and S. S. Rao, *Space efficient suffix trees*, J. Algorithms, 39 (2001), pp. 205–222.

[17] J. I. Munro, V. Raman, and A. J. Storm, *Representing dynamic trees succinctly*, in Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 529–536.

[18] C. St. J. A. Nash-Williams, *Edge-disjoint spanning trees of finite graphs*, J. London Math. Soc., 36 (1961), pp. 445–450.

[19] R. Pagh, *Low redundancy in dictionaries with $O(1)$ worst case lookup time*, in Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP), Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 595–604.

[20] V. Raman and S. S. Rao, *Static dictionaries supporting rank*, in Proceedings of the Tenth International Symposium on Algorithms and Computation (ISAAC), Lecture Notes in Comput. Sci. 1741, Springer-Verlag, Berlin, 1999, pp. 18–26.

[21] G. Turan, *Succinct representations of graphs*, Discrete Appl. Math., 8 (1984), pp. 289–294.

[22] M. Yannakakis, *Four pages are necessary and sufficient for planar graphs*, in Proceedings of the 18th ACM Symposium on Theory of Computing (STOC), 1986, pp. 104–108.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 777–798

TEMPORAL LOGIC AND SEMIDIRECT PRODUCTS: AN EFFECTIVE CHARACTERIZATION OF THE UNTIL HIERARCHY*

DENIS THÉRIEN† AND THOMAS WILKE‡

Abstract. We reveal an intimate connection between semidirect products of finite semigroups and substitution of formulas in linear temporal logic. We use this connection to obtain an algebraic characterization of the until hierarchy of linear temporal logic. (The kth level of that hierarchy is comprised of all temporal properties that are expressible by a formula of nesting depth k in the until operator.) Applying deep results from finite semigroup theory we are able to prove that each level of the until hierarchy is decidable. By means of Ehrenfeucht–Fraïssé games, we extend the results from linear temporal logic over finite sequences to linear temporal logic over infinite sequences.

Key words. linear temporal logic, until hierarchy, substitution, finite semigroups, pseudovarieties of semigroups, aperiodic semigroups, semidirect products

AMS subject classifications. 03B44, 03B70, 03D05, 68Q45, 68Q70, 20M20, 20M35

PII. S0097539797322772

1. Introduction.

Linear temporal logic is a language for expressing relationships between the order of events occurring over time. It is used in various areas of computer science as different as computer-aided verification (temporal logic as a specification language for reactive and concurrent systems) and databases (temporal logic to represent knowledge involving time and to deduct on time-dependent data). Its basic operators (modalities) are "next," "eventually," and "until," of which "until" is the most powerful one.[1] The latter observation has led to a classification of temporal properties according to the nesting depth in the until operator required to express them. In [8], it was shown that the resulting hierarchy, the so-called until hierarchy, is strict. In fact, "counting up to k" requires nesting depth k in the until operator (and nesting depth k is enough to express this property).

In this paper we show that the "until depth" of a given property (i.e., the lowest level of the until hierarchy the property belongs to) is effectively computable, both in the case where temporal logic is interpreted in finite sequences and in the case where it is interpreted in infinite sequences. This answers a question left open in [6] and [8].

Our approach to the computational problem of determining the until depth of a given property of finite sequences follows a well-known and established pattern. In a first step, we define a hierarchy of finite semigroups and prove that this hierarchy is tightly connected with the until hierarchy: a temporal property belongs to the kth level of the until hierarchy if and only if its syntactic semigroup (the transformation

*Received by the editors June 11, 1997; accepted for publication (in revised form) May 12, 2001; published electronically December 18, 2001.

http://www.siam.org/journals/sicomp/31-3/32277.html

†School of Computer Science, McGill University, 3480 University Montréal, Québec H3A 2A7, Canada (denis@opus.cs.mcgill.ca). This author's work was supported by the FCAR and NSERC.

‡Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany (wilke@ti.informatik.uni-kiel.de). Part of this research was carried out while this author was a postdoctoral fellow at DIMACS during the special year on logic and algorithms. This paper was written during this author's stay at Institut Gaspard Monge, Université de Marne-la-Vallée.

[1] If one chooses the right variant of the until operator, the other two operators become superfluous; i.e., they are no longer needed to achieve the full power of temporal logic.

semigroup of its minimal automaton) belongs to the kth level of the hierarchy of semigroups. In a second step, we show that for a finite semigroup one can effectively compute the lowest level of the hierarchy of semigroups to which it belongs. This yields the desired result, as the syntactic semigroup of a given temporal property can be computed effectively.

It is in the first step of the above process where we introduce new methodology. We develop a completely new approach to understanding semidirect products. We prove that semidirect products can be characterized in terms of substitution of formulas of linear temporal logic. Under reasonably weak assumptions, the semidirect product of two classes of semigroups corresponds to the set of formulas that are obtained by substituting formulas corresponding to the second class for atomic formulas in formulas corresponding to the first class. This is what we call the "semidirect product/substitution principle."

In the second step of the above process, we apply results from [18] and an intricate decidability criterion from [19] involving graph congruences to prove that each level of the hierarchy of semigroups in question is decidable.

The extension to the case where temporal logic is interpreted in infinite sequences is achieved by means of the Ehrenfeucht–Fraïssé game developed in [8]. This game gives us a way to approach the until hierarchy over infinite sequences from an algebraic point of view. We prove that the situation is exactly as in the case of finite sequences: a temporal property of infinite sequences belongs to the kth level of the until hierarchy if and only if its syntactic semigroup (in the sense of [2]) belongs to the kth level of the same semigroup hierarchy as above. The decidability result is then merely a corollary.

Temporal logic, first-order logic over unary predicates with total ordering, and star-free expressions are known to have the same expressive power on both finite sequences and on infinite sequences [15, 9, 11, 10, 22]. Each of the mutually equivalent formalisms motivates its own hierarchy: the until hierarchy, the quantifier-alternation hierarchy, and the dot-depth hierarchy [5]. The two latter are strict [3] and identical [23], whereas [8] describes a family of properties that separates the levels of the until hierarchy but is contained in the second level of the quantifier-alternation/dot-depth hierarchy. It is a long standing problem whether the "dot depth" of a star-free language is computable.

Star-free languages, and hence the class of all temporal properties, have also been shown to correspond to aperiodic (group-free) semigroups [15, 13]. The semigroup-theoretic characterization of the until hierarchy we give in this paper has a natural interpretation as a hierarchy for aperiodic semigroups, which is closely related to the Krohn–Rhodes decomposition theorem. This theorem is the most powerful tool for decomposing finite semigroups (respectively, finite automata).

In [4] it is shown that level 0 of the until hierarchy over finite sequences is decidable. In [6] the hierarchy of semigroups we consider is defined, and partial results about its relation to the until hierarchy over finite sequences are obtained. (For more details, see the remark right after Theorem 4.7.)

In section 2, we provide background on temporal logic and recall the definition of the until hierarchy over finite sequences. In section 3, we establish the connection between semidirect products and substitution. In section 4, we characterize the until hierarchy over finite sequences algebraically and show how the desired decidability result follows. In section 5, our results are extended to infinite sequences. Section 6 contains open problems.

This is a completely revised version of our technical report [20] and our conference

paper [21].

For finite semigroup theory, the reader is referred to the excellent monograph [1], from where our notation also stems. The book [14] provides background on the algebraic theory of regular languages. The handbook chapters [7] and [24] survey temporal logic and the theory of regular ω-languages.

2. Future temporal logic on strings and the until hierarchy. A *future temporal logic (FTL) formula* over a finite set P of propositional variables is built from the elements of P and the logical constant TRUE using the boolean connectives $\neg$ and $\vee$ and the usual temporal logic operators $\bigcirc$ (next), $\Diamond$ (eventually), and U (until), the two former of which are unary operators and the latter of which is a binary operator written in infix notation.

FTL formulas over a set P of propositional variables are interpreted in strings over an alphabet denoted Σ_P, which consists of all subsets of P. Given a string u, its length is denoted by $|u|$, and its positions are numbered 0, 1, 2, ..., $|u|-1$. The symbol at position i is denoted $u(i)$.

Given a string $u \in \Sigma_P^+$ and a position $i < |u|$, one defines what it means for an FTL formula φ over P to be true in u at i, in symbols $(u,i) \models \varphi$. In this paper, we adopt the following conventions:

- $(u,i) \models$ TRUE,
- $(u,i) \models p$ if $p \in u(i)$,
- $(u,i) \models \neg\varphi$ if not $(u,i) \models \varphi$,
- $(u,i) \models \varphi_1 \vee \varphi_2$ if $(u,i) \models \varphi_1$ or $(u,i) \models \varphi_2$,
- $(u,i) \models \bigcirc\varphi$ if $i+1 < |u|$ and $(u,i+1) \models \varphi$,
- $(u,i) \models \Diamond\varphi$ if there exists j with $i \leq j < |u|$ such that $(u,j) \models \varphi$,
- $(u,i) \models \varphi_1 \,\mathsf{U}\, \varphi_2$ if there exists j with $i \leq j < |u|$ such that $(u,i') \models \varphi_1$ for every i' with $i \leq i' < j$ and $(u,j) \models \varphi_2$.

Given an FTL formula φ, we write $u \models \varphi$ and say that u is a model of φ if $(u,0) \models \varphi$. We write $L(\varphi)$ for the set of all models of φ, i.e., $L(\varphi) = \{u \in \Sigma_P^+ \mid u \models \varphi\}$. We say that φ expresses $L(\varphi)$.

Similarly, we say that a language L is expressible in FTL if there exists an FTL formula φ such that $L = L(\varphi)$. Given a class Φ of FTL formulas, we write $L(\Phi)$ for $\{L(\varphi) \mid \varphi \in \Phi\}$.

In some of the commonly accepted definitions of the semantics of the eventually operator the value of j is restricted to positions strictly larger than i. For the until operator, a similar restriction of i' to the values strictly between i and j has been considered. The until depth of a language, however, does not depend on which of the four possible combinations is actually chosen. (For a formal definition of until depth, see below.)

The *until depth* of an FTL formula is its nesting depth in terms of the until operator. When φ is a formula, $\mathrm{ud}(\varphi)$ denotes its until depth. Formally, the until depth is defined inductively by

$$\begin{aligned}
\mathrm{ud}(\text{TRUE}) &= 0,\\
\mathrm{ud}(p) &= 0 \qquad \text{for } p \in P,\\
\mathrm{ud}(\neg\varphi) &= \mathrm{ud}(\varphi),\\
\mathrm{ud}(\varphi_1 \vee \varphi_2) &= \max\{\mathrm{ud}(\varphi_1), \mathrm{ud}(\varphi_2)\},\\
\mathrm{ud}(\bigcirc\varphi) &= \mathrm{ud}(\varphi),\\
\mathrm{ud}(\Diamond\varphi) &= \mathrm{ud}(\varphi),
\end{aligned}$$

$$\mathrm{ud}(\varphi_1 \,\mathsf{U}\, \varphi_2) = \max\{\mathrm{ud}(\varphi_1), \mathrm{ud}(\varphi_2)\} + 1.$$

The class of all FTL formulas of until depth at most k is denoted by $\mathtt{Ut}_k$. That is, a language L belongs to $L(\mathtt{Ut}_k)$ if and only if it is expressed by some FTL formula of until depth at most k. The until hierarchy is the sequence $L(\mathtt{Ut}_0)$, $L(\mathtt{Ut}_1)$, $L(\mathtt{Ut}_2)$, ... In [8], it is shown that this hierarchy is strict in the following sense: $L(\mathtt{Ut}_0) \subsetneq L(\mathtt{Ut}_1) \subsetneq L(\mathtt{Ut}_2) \subsetneq \cdots$.

The main result of this paper is Corollary 4.18, which states that given an FTL formula φ or a regular language L expressible in FTL, one can compute the lowest level of the until hierarchy $L(\varphi)$ (respectively, L) belongs to, i.e., the smallest k such that $L(\varphi) \in L(\mathtt{Ut}_k)$ (respectively, $L \in L(\mathtt{Ut}_k)$). Theorem 5.9 states the same result for ω-words instead of strings.

3. Substitution and semidirect products. This section builds the core of the paper. It provides the correspondence between semidirect products of pseudovarieties of semigroups and substitution in past temporal logic formulas.

3.1. Past temporal logic. *Past temporal logic (PTL) formulas* are built exactly as FTL formulas but use a corresponding set of reversed operators: instead of $\bigcirc$, $\Diamond$, and U, PTL formulas use $\ominus$ (previously), $\diamondminus$ (sometime in the past, once upon a time), and S (since). The semantics of PTL formulas is defined analogously to the semantics of FTL formulas: in the above definition for FTL, every temporal operator is replaced by its reversed form, and every condition that involves the ordering is replaced by the symmetric condition

- $(u, i) \models \ominus\varphi$ if $i > 0$ and $(u, i-1) \models \varphi$,
- $(u, i) \models \diamondminus\varphi$ if there exists j with $j \leq i$ such that $(u, j) \models \varphi$,
- $(u, i) \models \varphi_1 \,\mathsf{S}\, \varphi_2$ if there exists j with $j \leq i$ such that $(u, i') \models \varphi_1$ for every i' with $j < i' \leq i$ and $(u, j) \models \varphi_2$.

Given a PTL formula φ over a set P of propositional variables and $u \in \Sigma_P^+$, we write $u \models \varphi$ and say that u is a model of φ if $(u, |u| - 1) \models \varphi$. As a consequence, if u is a string and v its reverse, and if φ is an FTL formula and ψ its reversed counterpart (obtained by replacing every operator by its reversal), then $u \models \varphi$ if and only if $v \models \psi$.

Abusing notation, we use $L(\varphi)$ to denote the set of models of both FTL and PTL formulas, although there are formulas that are FTL and PTL formulas at the same time and their respective sets of models are distinct; consider, e.g., the atomic formula p. It will, however, always be clear from the context whether we view a given formula as an FTL or a PTL formula. Similarly, we use $L(\Phi)$ to also denote the class of all languages expressible by some PTL formula from Φ.

The *since depth* of a PTL formula is its nesting depth in terms of the since operator; we denote the class of all PTL formulas of since depth at most k by $\mathtt{Sn}_k$. It should be clear that a language belongs to $L(\mathtt{Ut}_k)$ if and only if its reverse belongs to $L(\mathtt{Sn}_k)$.

3.2. Definition of substitution. If φ is a PTL formula over a set of propositions P and $(\psi_p)_{p \in P}$ a family of PTL formulas over Q, then $\varphi[p \mapsto \psi_p]$ stands for the PTL formula over Q obtained from φ by simultaneously replacing every occurrence of a propositional variable $p \in P$ by ψ_p.

Formally, substitution is defined by

$$\textsc{True}[p \mapsto \psi_p] = \textsc{True}, \tag{3.1}$$

$$p'[p \mapsto \psi_p] = \psi_{p'} \qquad \text{for } p' \in P, \tag{3.2}$$

$$\neg\varphi[p \mapsto \psi_p] = \neg(\varphi[p \mapsto \psi_p]), \tag{3.3}$$

$$\varphi_1 \vee \varphi_2[p \mapsto \psi_p] = (\varphi_1[p \mapsto \psi_p]) \vee (\varphi_2[p \mapsto \psi_p]), \tag{3.4}$$

$$\ominus\varphi[p \mapsto \psi_p] = \ominus(\varphi[p \mapsto \psi_p]), \tag{3.5}$$

$$\diamondminus\varphi[p \mapsto \psi_p] = \diamondminus(\varphi[p \mapsto \psi_p]), \tag{3.6}$$

$$\varphi_1 \mathsf{S} \varphi_2[p \mapsto \psi_p] = (\varphi_1[p \mapsto \psi_p]) \mathsf{S} (\varphi_2[p \mapsto \psi_p]). \tag{3.7}$$

3.3. Basic properties of substitution. Obviously, substitution is associative.

REMARK 1 (associativity of substitution). *Let φ be a PTL formula over P, $\{\psi_p\}_{p\in P}$ a family of PTL formulas over Q, and $\{\chi_q\}_{q\in Q}$ a family of PTL formulas over R.*

Then

$$\varphi[p \mapsto \psi_p[q \mapsto \chi_q]] = \varphi[p \mapsto \psi_p][q \mapsto \chi_q]. \tag{3.8}$$

In first-order logic, the most important property of substitution is described by the so-called substitution lemma. There is an analogue with temporal logic. For a family $(\psi_p)_{p\in P}$ of PTL formulas over Q and a string $u \in \Sigma_Q^+$, define a string over Σ_P, denoted $u[p \mapsto \psi_p]$, as follows: $u[p \mapsto \psi_p] = v_0 \dots v_{|u|-1}$ where $v_i = \{p \in P \mid (u, i) \models \psi_p\}$. Using this notation, the substitution lemma for temporal logic can be stated as follows.

LEMMA 3.1 (substitution lemma for PTL). *Let φ be a PTL formula over P and $(\psi_p)_{p\in P}$ a family of PTL formulas over Q.*

For every string u over Σ_Q and every position $i < |u|$, the following then holds:

$$(u, i) \models \varphi[p \mapsto \psi_p] \qquad \textit{if and only if} \qquad (u[p \mapsto \psi_p], i) \models \varphi. \tag{3.9}$$

This can be proven by a straightforward induction on the structure of φ.

An immediate consequence of the substitution lemma is the following corollary.

COROLLARY 3.2. *Let φ and φ' be PTL formulas over P, and $\{\psi_p\}_{p\in P}$ and $\{\psi'_p\}_{p\in P}$ families of PTL formulas over Q, such that $L(\varphi) = L(\varphi')$ and $L(\psi_p) = L(\psi'_p)$ for every $p \in P$.*

Then $L(\varphi[p \mapsto \psi_p]) = L(\varphi'[p \mapsto \psi'_p])$.

3.4. Substitution and sequential functions. Recall that a function $\sigma\colon A^+ \to B^+$ is called a sequential function if it is realized by a transducer. A transducer is a tuple

$$(\mathcal{Q}, A, B, s_I, \delta, \lambda), \tag{3.10}$$

where $\mathcal{Q}$ is a finite set of states, $q_I \in \mathcal{Q}$ is the initial state, A and B are alphabets, $\delta\colon \mathcal{Q} \times A \to \mathcal{Q}$ is a transition function, and $\lambda\colon \mathcal{Q} \times A \to B^+$ is an output function. The transition function δ is inductively extended to a function $\mathcal{Q} \times A^+ \to \mathcal{Q}$ by setting $\delta(q, wa) = \delta(\delta(q, w), a)$. The function $\sigma\colon A^+ \to B^+$ realized by the transducer is inductively defined by

$$\begin{aligned} \sigma(a) &= \lambda(q_I, a) && \text{for } a \in A, \\ \sigma(wa) &= \sigma(w)\,\lambda(\delta(q_I, w), a) && \text{for } a \in A,\ w \in A^+. \end{aligned}$$

A transducer as in (3.10) is called synchronous if $\lambda(q, a) \in B$ for all $q \in \mathcal{Q}$, $a \in A$. Accordingly, a sequential function is called synchronous if it is realized by a synchronous transducer.

Under a certain proviso inverse images of synchronous sequential functions can be described by substitution.

LEMMA 3.3. *Let $\sigma\colon \Sigma_Q^+ \to \Sigma_P^+$ be a synchronous sequential function and $\{\psi_p\}_{p\in P}$ a family of PTL formulas over Q such that*

$$\sigma^{-1}(L(p)) = L(\psi_p) \tag{3.11}$$

for every $p \in P$.

Then, for every PTL formula φ over P,

$$\sigma^{-1}(L(\varphi)) = L(\varphi[p \mapsto \psi_p]). \tag{3.12}$$

Proof. The proof is by induction on the structure of φ. The base case is trivial. In the induction step, we have to consider boolean connectives and temporal operators. Boolean connectives are dealt with easily, because inverse functions respect boolean operations. Of the three temporal operators, we consider only "since," which requires the most involved argument. Given a string u and a position $i < |u|$, we will use $u[0,i]$ to denote the prefix $u(0)\dots u(i)$ of u. Observe that, in general, $u[0,i] \models \varphi$ is equivalent to $(u,i) \models \varphi$ for every PTL formula φ.

Assume φ is of the form $\varphi_1 \mathsf{S} \varphi_2$, and let u be an arbitrary string over Σ_Q. Let $v = \sigma(u)$. Then, since σ is assumed to be synchronous, $|v| = |u|$. The string u belongs to $\sigma^{-1}(L(\varphi_1 \mathsf{S} \varphi_2))$ if and only if there exists $i < |v|$ such that

- $v[0,i] \models \varphi_2$ and
- $v[0,j] \models \varphi_1$ for every j with $i < j < |v|$.

By induction hypothesis, we know that $v[0,i] \models \varphi_2$ if and only if $u[0,i] \models \varphi_2[p \mapsto \psi_p]$. (Observe that, since σ is synchronous, $\sigma(u[0,i]) = v[0,i]$.) Similarly, $v[0,j] \models \varphi_1$ if and only if $u[0,j] \models \varphi_1[p \mapsto \psi_p]$ for every $j < |u|$. Therefore, the above assertion is equivalent to the existence of $i \le |u|$ such that

- $u[0,i] \models \varphi_2[p \mapsto \psi_p]$ and
- $u[0,j] \models \varphi_1[p \mapsto \psi_p]$ for every j with $i < j$.

According to the semantics of PTL formulas, the latter is true if and only if $u \models (\varphi_1[p \mapsto \psi_p]) \mathsf{S} (\varphi_2[p \mapsto \psi_p])$. However, in view of (3.7), $(\varphi_1[p \mapsto \psi_p]) \mathsf{S} (\varphi_2[p \mapsto \psi_p])$ is the same as $(\varphi_1 \mathsf{S} \varphi_2)[p \mapsto \psi_p]$. □

The previous lemma actually holds under seemingly weaker assumptions; besides (3.11) we need only that σ is *length-preserving,* i.e., that σ satisfies $|\sigma(u)| = |u|$ for every $u \in \Sigma_Q^+$, and that it *respects prefixes* in the following sense: $\sigma(uv) \in \sigma(u)\Sigma_P^+$ for all $u, v \in \Sigma_Q^+$. It is, however, easy to see that if these conditions are met, then σ is a synchronous sequential function.

3.5. Substitution and homomorphisms. One way to interpret Lemma 3.3 is to say that substitution can (in many cases) be regarded as applying the inverse of a sequential function. If the substituents are very simple, substitution can even be explained using homomorphisms.

LEMMA 3.4. *Let φ be a PTL formula over P and $\{\psi_p\}_{p\in P}$ a P-indexed family of PTL formulas over Q. Furthermore, assume each formula ψ_p is a disjunction of propositional variables from Q.*

Then $L(\varphi[p \mapsto \psi_p])$ is the preimage of $L(\varphi)$ under a homomorphism $\Sigma_Q^+ \to \Sigma_P^+$.

By convention, an empty disjunction is FALSE.

Proof. For every $p \in P$, let D_p denote the set of disjuncts of ψ_p. Consider the homomorphism $g\colon \Sigma_Q^+ \to \Sigma_P^+$ defined by $g(b) = \bigcup\{p \mid D_p \cap b \neq \emptyset\}$. Let $v \in \Sigma_Q^*$ and $b \in \Sigma_Q$, and assume $g(vb) = ua$ where $a \in \Sigma_P$. Then $p \in a$ if and only if there exists

$q \in b$ such that $q \in D_p$. The latter is equivalent to $vb \models \psi_p$. Thus, $g^{-1}(L(p)) = L(\psi_p)$, and we can apply Lemma 3.3 to conclude $g^{-1}(L(\varphi)) = L(\varphi[p \mapsto \psi_p])$. □

This lemma can be strengthened.

LEMMA 3.5. *Let φ be a PTL formula over P and $\{\psi_p\}_{p\in P}$ a P-indexed family of PTL formulas over Q. Furthermore, assume each formula ψ_p is a boolean combination of propositional variables from Q (i.e., a propositional formula).*

Then $L(\varphi[p \mapsto \psi_p])$ is the preimage of $L(\varphi)$ under a homomorphism $\Sigma_Q^+ \to \Sigma_P^+$.

Proof. For $b \in \Sigma_Q$, let α_b be a propositional formula such that $L(\alpha_b) = \Sigma_Q^* b$. For each $p \in P$, define a formula μ_p over Σ_Q (!) as follows. Let μ_p be the disjunction of all $b \in \Sigma_Q$ such that α_b implies ψ_p (as a propositional formula). Then, $L(\psi_p) = L(\mu_p[b \mapsto \alpha_b])$, and thus $L(\varphi[p \mapsto \psi_p]) = L(\varphi[p \mapsto \mu_p][b \mapsto \alpha_b])$. Write φ' for $\varphi[p \mapsto \mu_p]$. By the previous lemma, $L(\varphi') = g^{-1}(L(\varphi))$ for some homomorphism $g\colon \Sigma_{\Sigma_Q}^+ \to \Sigma_P^+$.

Now consider the homomorphism $h\colon \Sigma_Q^+ \to \Sigma_{\Sigma_Q}^+$ defined by $h(b) = \{b\}$. Trivially, $h^{-1}(L(b)) = L(\alpha_b)$. Lemma 3.3 then implies $L(\varphi'[b \mapsto \alpha_b]) = h^{-1}(L(\varphi'))$, which means $L(\varphi[p \mapsto \psi_p]) = h^{-1}(g^{-1}(L(\varphi)))$. □

3.6. Substitution on classes of formulas. Pseudovarieties of semigroups are classes of finite semigroups closed under homomorphic images, finite direct products, and subsemigroups; see [1, p. 53]. With each such class $\mathbf{V}$, we associate the class $L(\mathbf{V})$ of all languages over alphabets of the form Σ_P that are recognized by elements from $\mathbf{V}$. Recall that a language $L \subseteq A^+$ is said to be recognized by a semigroup S if there exists a semigroup homomorphism $h\colon A^+ \to S$ and a subset $F \subseteq S$ such that $L = h^{-1}(F)$, i.e., if L is the preimage of some subset of S under h.

Complex pseudovarieties of semigroups are often decomposed using semidirect products; that is, they are described as a semidirect product of simpler pseudovarieties. As usual, we denote the semidirect product of two pseudovarieties $\mathbf{V}$ and $\mathbf{W}$ by $\mathbf{V} * \mathbf{W}$; see [1, pp. 266, 269].

On classes of formulas, we now define an operation, denoted $\star$, which corresponds to the semidirect product on classes of semigroups, as we will see below in Theorem 3.6.

Given classes Φ and Ψ of PTL formulas, we define $\Phi \star \Psi$ to contain all formulas that are boolean combinations of formulas from Ψ and formulas that can be written in the form $\varphi[p \mapsto \psi_p]$ where $\varphi \in \Phi$ and the ψ_p's themselves are boolean combinations of formulas from Ψ and propositional variables.

Before we investigate the close connection between $\star$ and $*$, let us observe that $\star$ is a convenient tool for defining classes of formulas. Using $\star$, for instance, the kth level of the since hierarchy can be expressed in terms of its first level:

$$\mathtt{Sn}_k = \underbrace{\mathtt{Sn}_1 \star \cdots \star \mathtt{Sn}_1}_{k \text{ factors}} \qquad \text{for } k > 0. \tag{3.13}$$

Notice that $\star$ is associative. Therefore grouping is not necessary in the above equation.

As is usual with associative binary operations, we allow ourselves to use exponentiation to denote "iterated multiplication"; e.g., we allow ourselves to write $\mathtt{Sn}_1^k$ for the right-hand side of the above equation.

3.7. The semidirect product/substitution principle. In this section we relate the two operations $\star$ and $*$ to each other. There are two parts to this relation, Propositions 3.8 and 3.10. Combined into one statement, they yield the following concise theorem. The set of all PTL formulas without occurrence of $\diamond$ and S is denoted by $\mathtt{Pv}$.

THEOREM 3.6 (semidirect product/substitution principle, SSP). *Let* $\mathbf{V}$, $\mathbf{W}$ *be pseudovarieties of semigroups and* Φ, Ψ *classes of PTL formulas such that* $L(\mathbf{V}) = L(\Phi)$ *and* $L(\mathbf{W}) = L(\Psi)$.

If $L(\Phi \star \mathtt{P}\mathbf{v} \star \Psi) \subseteq L(\Phi \star \Psi)$, *then*

$$L(\mathbf{V} * \mathbf{W}) = L(\Phi \star \Psi). \tag{3.14}$$

The theorem follows from Propositions 3.8 and 3.10 below, each of which proves one inclusion of (3.14).

In the proof of Proposition 3.8 we will make use of the following fact about wreath products (see [1, pp. 265, 266]) and regular languages (see, e.g., [14]). (Wreath products are closely related to semidirect products and come in handy whenever one works with transformation semigroups.) Recall that the transformation semigroup of a transducer as in (3.10) is (Q, T), where T is the subsemigroup of Q^Q generated by $q \mapsto \delta(q, a)$ for $a \in A$.

PROPOSITION 3.7. *Let* $\sigma\colon A^+ \to B^+$ *be a sequential function given by a transducer whose transformation semigroup is* $(\mathcal{Q}, T)$, *and let* $L \subseteq B^+$ *be a language recognized by a semigroup* S.

Then $\sigma^{-1}(L)$ *is recognized by the semigroup of* $(S^1, S) \circ (\mathcal{Q}, T)$, *which is isomorphic to a particular semidirect product* $S^{\mathcal{Q}} * T$.

The inclusion from right to left in (3.14) now follows from the following proposition.

PROPOSITION 3.8 (SSP I). *Let* $\mathbf{V}$, $\mathbf{W}$ *be pseudovarieties of semigroups and* Φ, Ψ *classes of PTL formulas such that* $L(\mathbf{V}) = L(\Phi)$ *and* $L(\mathbf{W}) = L(\Psi)$.

Then

$$L(\Phi \star \Psi) \subseteq L(\mathbf{V} * \mathbf{W}). \tag{3.15}$$

Proof. Assume $\varphi \in \Phi$ is a formula over P and $\{\psi_p\}_{p \in P}$ is a family of formulas over Q where each ψ_p is a boolean combination of propositional variables and formulas from Ψ. We will show that $L(\varphi[p \mapsto \psi_p])$ is recognized by an element of $\mathbf{V} * \mathbf{W}$. This is enough for two reasons. First, $\mathbf{W} \subseteq \mathbf{V} * \mathbf{W}$ holds for arbitrary pseudovarieties of semigroups, unless $\mathbf{V}$ is the empty pseudovariety or the pseudovariety consisting of the empty semigroup only. This means $L(\mathbf{W}) \subseteq L(\mathbf{V} * \mathbf{W})$. Second, $L(\mathbf{V} * \mathbf{W})$ is closed under boolean combinations, as $L(\mathbf{U})$ is closed under boolean combinations for any arbitrary pseudovariety $\mathbf{U}$ of semigroups.

Let $Q = \{q_0, \ldots, q_{m-1}\}$ and assume $\psi_p = \alpha_p(q_0, \ldots, q_{m-1}, \mu_0, \ldots, \mu_{n-1})$, where α_p is a boolean combination and the μ_j's are from Ψ. Since $\mathbf{W}$ is assumed to be a pseudovariety, we can find a single semigroup $T \in \mathbf{W}$ and a homomorphism $h\colon \Sigma_Q^+ \to T$ such that every language $L(\mu_j)$ is recognized by h; i.e., for every $j < n$ there exists $F_j \subseteq T$ such that $L(\mu_j) = h^{-1}(F_j)$. For every $j < n$, let ν_j be the formula $\bigvee_{t \in F_j} t$. And for each $t \in T$, let κ_t be a formula over Q from Ψ such that $L(\kappa_t) = h^{-1}(t)$. Then

$$L(\psi_p) = L(\alpha_p(\bar{q}, \bar{\nu})[q/t \mapsto q/\kappa_t]), \tag{3.16}$$

where $[q/t \mapsto q/\kappa_t]$ means that every element from Q is replaced by itself and every element from t is replaced by κ_t; overstriking indicates a sequence of formulas.

Write β_p for $\alpha_p(\bar{q}, \bar{\nu})$. Then

$$L(\varphi[p \mapsto \psi_p]) = L(\varphi[p \mapsto \beta_p][q/t \mapsto q/\kappa_t]). \tag{3.17}$$

Denote $\varphi[p \mapsto \beta_p]$ by φ'. From Lemma 3.4, we can conclude that $L(\varphi')$ is recognized by an element of $\mathbf{V}$, say S, as $L(\mathbf{V})$ is closed under inverse images.

Consider the sequential function σ defined by the transducer

$$(T^1, \Sigma_Q, \Sigma_{T\cup Q}, 1, \delta, \lambda) \tag{3.18}$$

as specified through

$$\delta(t, a) = t\,h(a), \tag{3.19}$$

$$\lambda(t, a) = \{t\,h(a)\} \cup a \qquad \text{for } t \in T^1,\ a \in \Sigma_Q. \tag{3.20}$$

The transformation semigroup of this transducer is isomorphic to (T^1, T). We have $\sigma(ua) \in \Sigma_T^*(\{h(ua)\} \cup a)$ for $u \in \Sigma_Q^*$, $a \in \Sigma_Q$, i.e., $\sigma^{-1}(L(t)) = h^{-1}(t)$ for every $t \in T$, hence $\sigma^{-1}(L(t)) = L(\kappa_t)$ for every $t \in T$. Also, $\sigma^{-1}(L(q)) = L(q)$ for every $q \in Q$. Thus, by Lemma 3.3, $L(\varphi'[q/t \mapsto q/\kappa_t]) = \sigma^{-1}(L(\varphi'))$. On the other hand, $\sigma^{-1}(L(\varphi'))$ is recognized by the semigroup of $(S^1, S) \circ (T^1, T)$ according to Proposition 3.7. Hence, $L(\varphi'[q/t \mapsto q/\kappa_t])$ is recognized by $S^{S^1} * T$, which is an element of $\mathbf{V} * \mathbf{W}$. □

To prove the other inclusion of (3.14), we need the so-called wreath product principle, introduced in [17]. For use with temporal logic, it can be formulated as follows.

PROPOSITION 3.9 (wreath product principle). *Let $L \subseteq \Sigma_P^+$ be a language recognized by the semigroup of a wreath product $(\mathcal{P}, S) \circ (\mathcal{Q}, T)$ of transformation semigroups via a homomorphism $h\colon \Sigma_P^+ \to S^{\mathcal{Q}} * T$. Denote the projection of h onto the second component by h_2, and let $R = T^1 \cup P$. Define $\sigma\colon \Sigma_P^+ \to \Sigma_R^+$ by $\sigma(a_0 \dots a_{n-1}) = b_0 \dots b_{n-1}$, where $b_i = \{h_2(a_0 \dots a_{i-1})\} \cup a_i$. (By convention, $h(\epsilon) = 1$.)*

This function is a synchronous sequential function which can be realized by a transducer whose transformation semigroup is T's right regular representation.

The language L is a boolean combination of languages of the following two types:

(i) *languages of the form $h_2^{-1}(t)$ for some $t \in T$;*

(ii) *languages of the form $\sigma^{-1}(V)$ where $V \subseteq \Sigma_R^+$ is recognized by S.*

The other inclusion of (3.14) follows from the following proposition.

PROPOSITION 3.10 (SSP II). *Let $\mathbf{V}$, $\mathbf{W}$ be pseudovarieties of semigroups and Φ, Ψ classes of PTL formulas such that $L(\mathbf{V}) = L(\Phi)$ and $L(\mathbf{W}) = L(\Psi)$.*

Then

$$L(\mathbf{V} * \mathbf{W}) \subseteq L(\Phi \star \mathtt{Pv} \star \Psi). \tag{3.21}$$

Proof. Assume $L \subseteq \Sigma_P^+$ is a language recognized by an element of $\mathbf{V} * \mathbf{W}$. There exist transformation semigroups $(\mathcal{P}, S)$ and $(\mathcal{Q}, T)$ with $S \in \mathbf{V}$ and $T \in \mathbf{W}$ such that L is recognized by the semigroup of $(\mathcal{P}, S) \circ (\mathcal{Q}, T)$ via some homomorphism $h\colon \Sigma_P^+ \to S^{\mathcal{Q}} * T$.

Let R, h_2, and σ be as in Proposition 3.9. It is enough to show that the two types of languages described in Proposition 3.9 can be defined by formulas from $\Phi \star \mathtt{Pv} \star \Psi$.

Clearly, every language of type (i) is definable by a formula from Ψ. To complete the proof, we show that every language of type (ii) is definable by a formula from $\Phi \star \mathtt{Pv} \star \Psi$.

Let $V \subseteq \Sigma_R^+$ be a language recognized by S. By assumption, there exists a formula $\varphi \in \Phi$ such that $V = L(\varphi)$. Also, for every $t \in T$, there exists a formula $\mu_t \in \Psi$ such that $h_2^{-1}(t) = L(\mu_t)$.

We want to apply Lemma 3.3. For every $r \in R$, we construct a formula ψ_r such that $\sigma^{-1}(L(r)) = L(\psi_r)$. By definition of σ, the ψ_r's can be chosen as follows:

$$\psi_r = \begin{cases} r & \text{if } r \in \mathcal{P}, \\ \ominus \mu_r & \text{if } r \in T \setminus \{1\}, \\ \ominus \mu_1 \wedge \neg \ominus \textsc{True} & \text{if } r = 1 \text{ and } 1 \in T, \\ \neg \ominus \textsc{True} & \text{if } r = 1 \text{ and } 1 \notin T. \end{cases}$$

(Notice that $\neg\ominus\textsc{True}$ defines the set of all strings of length exactly 1.) We can now apply Lemma 3.3 and conclude $\sigma^{-1}(L(\varphi)) = L(\varphi[r \mapsto \psi_r])$, hence $\sigma^{-1}(V) = L(\varphi[r \mapsto \psi_r])$. Obviously, every formula ψ_r belongs to $\mathtt{Pv} \star \Psi$ or else is atomic. □

4. Since and until hierarchy. In this section, we first obtain an algebraic characterization of the levels of the since hierarchy and will then be able to deduce that each level of this hierarchy is decidable. By the symmetry between FTL and PTL, this also implies that the until hierarchy is decidable.

There are essentially two ways of obtaining an effective characterization of the since hierarchy. In the first approach, one first shows that the kth level of the since hierarchy is characterized by the left-hand side of (4.30), then shows (4.30), and finally proves that membership in the pseudovariety on the right-hand side of (4.30) is decidable. This approach was taken in our technical report [20] and relies heavily on fundamental results from [16]. The second approach is taken here. In it most of the algebraic arguments from [16] are replaced by syntactic/logical arguments. In that way, some of the results from [16] can actually be reproved, which is demonstrated in section 4.4.

4.1. A normal form for PTL. We first prove a strong normal form for temporal logic which replaces the complicated algebraic argument of section 6.1 of our technical report [20]. The normal form is established by syntactic means.

The observation that in PTL application of the previously operator can be restricted to propositional variables or negated propositional variables is folklore. It follows from the following (almost obvious) lemma.

LEMMA 4.1 (switching rules for $\ominus$). *For arbitrary PTL formulas φ and ψ,*

$$\begin{aligned} L(\ominus(\varphi_1 \,\mathsf{S}\, \varphi_2)) &= L(\ominus\varphi_1 \,\mathsf{S}\, \ominus\varphi_2), \\ L(\ominus \diamondminus \varphi) &= L(\diamondminus \ominus \varphi), \\ L(\ominus(\varphi \vee \psi)) &= L(\ominus\varphi \vee \ominus\psi), \\ L(\ominus\neg\varphi) &= L(\neg\ominus\varphi \wedge \ominus\textsc{True}). \end{aligned}$$

Writing $\mathtt{On}$ (reminding of "once upon a time") for the class of all PTL formulas not using S and $\ominus$, we obtain as an immediate consequence the following identity:

$$L(\mathtt{Sn}_0) = L(\mathtt{On} \star \mathtt{Pv}). \tag{4.1}$$

We will extend this to any level of the since hierarchy. Let $\underline{\mathtt{Sn}}_k$ denote all formulas in $\mathtt{Sn}_k$ that don't use $\ominus$ or $\diamondminus$. Then we have the following theorem.

THEOREM 4.2. *For every $k \geq 0$,*

$$L(\mathtt{Sn}_k) = L(\mathtt{On} \star \underline{\mathtt{Sn}}_k \star \mathtt{Pv}). \tag{4.2}$$

In other words,

$$L(\mathtt{Sn}_k) = L(\mathtt{On} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) \qquad \text{for } k \geq 0. \tag{4.3}$$

Obviously, it is enough to prove (4.2) for $k = 1$. However, this follows immediately from Lemmas 4.1 and 4.3. The operator $\boxminus$ is defined by $\boxminus\varphi = \neg\diamondminus\neg\varphi$, and using it makes the formulas in Lemma 4.3 more readable.

LEMMA 4.3 (switching rules for $\diamondminus$ and S). *For arbitrary PTL formulas φ, ψ, and ρ,*

$$\begin{aligned}
L((\varphi \wedge \psi)\ \mathsf{S}\ \rho) &= L((\varphi\ \mathsf{S}\ \rho) \wedge (\psi\ \mathsf{S}\ \rho)),\\
L((\varphi\ \mathsf{S}\ (\psi \vee \rho)) &= L((\varphi\ \mathsf{S}\ \psi) \vee (\varphi\ \mathsf{S}\ \rho)),\\
L((\varphi \vee \diamondminus q)\ \mathsf{S}\ \rho) &= L((\varphi\ \mathsf{S}\ \rho) \vee \diamondminus(\diamondminus q \wedge \ominus(\varphi\ \mathsf{S}\ \rho))),\\
L((\varphi \vee \boxminus\psi)\ \mathsf{S}\ \rho) &= L((\varphi\ \mathsf{S}\ \rho) \vee (\diamondminus\rho \wedge \boxminus(\varphi \vee \boxminus\psi))),\\
L(\varphi\ \mathsf{S}\ (\psi \wedge \diamondminus\rho)) &= L((\varphi\ \mathsf{S}\ \psi) \wedge \diamondminus(\psi \wedge \diamondminus\rho)),\\
L(\varphi\ \mathsf{S}\ (\psi \wedge \boxminus\rho)) &= L(\varphi\ \mathsf{S}\ (\psi \wedge \rho) \wedge \boxminus(\boxminus\rho \vee (\varphi \wedge \varphi\ \mathsf{S}\ (\psi \wedge \rho)))).
\end{aligned}$$

Proof. We prove only the inclusion from right to left in the last equation, which is the only nontrivial assertion. Write φ_L and φ_R for the left- and right-hand sides of this equation, respectively.

Assume $u \models \varphi_R$. Then there exists a position i in u such that

$$(u, i) \models \psi, \tag{4.4}$$

$$(u, i) \models \rho, \tag{4.5}$$

$$(u, j) \models \varphi \qquad \text{for every } j \text{ with } i < j < |u|. \tag{4.6}$$

Fix i to be the minimal such position. Clearly, if $(u, i) \models \boxminus\rho$, then $u \models \varphi_L$. Therefore, for the rest, we assume

$$(u, i) \not\models \boxminus\rho \tag{4.7}$$

and show this assumption leads to a contradiction. As the second conjunct of φ_R holds at i, but $\boxminus\rho$ does not, we conclude

$$(u, i) \models \varphi. \tag{4.8}$$

Also, $i > 0$ and

$$(u, i-1) \not\models \boxminus\rho \tag{4.9}$$

because otherwise $\boxminus\rho$ would hold at i (cf. (4.5)). Hence,

$$(u, i-1) \models \varphi\ \mathsf{S}\ (\psi \wedge \rho), \tag{4.10}$$

again because the second conjunct of φ_R holds at i, but $\boxminus\rho$ does not. Thus there exists a position i' with $i' < i$ such that

$$(u, i') \models \psi, \tag{4.11}$$

$$(u, i') \models \rho, \tag{4.12}$$

$$(u, j) \models \varphi \qquad \text{for every } j \text{ with } i' < j < i. \tag{4.13}$$

From (4.6) and (4.8) we conclude that instead of (4.13) we even have

$$(u, j) \models \varphi \qquad \text{for every } j \text{ with } i' < j < |u|. \tag{4.14}$$

Therefore (4.11), (4.12), and (4.14) show that i was not chosen minimal—a contradiction. □

4.2. Characterization of small classes. Before we can give an algebraic characterization of the levels of the since hierarchy, we need to characterize classes of languages expressible by very simple formulas.

We recall some terminology and notation, following [1]. $\mathrm{B}(1,2)^1$ stands for the semigroup $\{1, a, b\}$ whose multiplication is given by $xa = a$, $xb = b$, and $x1 = x$ for all $x \in \{1, a, b\}$. The subsemigroups $\{a, b\}$ and $\{1, a\}$ of $\mathrm{B}(1,2)^1$ are denoted by $\mathrm{B}(1,2)$ and Sl_2, respectively.

Given a class $\mathcal{C}$ of finite semigroups, $V(\mathcal{C})$ denotes the smallest pseudovariety of semigroups containing $\mathcal{C}$, and $V^*(\mathcal{C})$ denotes the smallest pseudovariety of semigroups closed under semidirect products that contains $\mathcal{C}$. Given a pseudovariety $\mathbf{V}$ of semigroups, $\mathbf{V}^n$ stands for the iterated semidirect product $\mathbf{V} * \cdots * \mathbf{V}$ with n factors, and $\mathbf{LV}$ stands for the class of all finite semigroups whose submonoids belong to $\mathbf{V}$.

For $n \geq 1$, $\mathbf{D}_n$ is the pseudovariety of semigroups defined by the equation

$$yx_1 \dots x_n = x_1 \dots x_n \tag{4.15}$$

and

$$\mathbf{D} = \bigcup_{n>0} \mathbf{D}_n. \tag{4.16}$$

$\mathbf{R}$ is the pseudovariety of all $\mathcal{R}$-trivial finite semigroups, $\mathbf{SL}$ is the pseudovariety of all finite semilattices, and $\mathbf{MD_1}$ is the pseudovariety of semigroups generated by all semigroups of the form S^1 with $S \in \mathbf{D_1}$.

Equivalent descriptions of $\mathbf{D_1}$, $\mathbf{SL}$, $\mathbf{R}$, $\mathbf{MD_1}$, and $\mathbf{LR}$ are (see, e.g., [1])

$$\mathbf{D_1} = V(\{\mathrm{B}(1,2)\}), \tag{4.17}$$

$$\mathbf{SL} = V(\{\mathrm{Sl}_2\}), \tag{4.18}$$

$$\mathbf{D} = V^*(\mathbf{D_1}), \tag{4.19}$$

$$\mathbf{R} = V^*(\mathbf{SL}), \tag{4.20}$$

$$\mathbf{MD_1} = V(\{\mathrm{B}(1,2)^1\}), \tag{4.21}$$

$$\mathbf{LR} = V^*(\{\mathrm{B}(1,2), \mathrm{Sl}_2\}) \;=\; \mathbf{R} * \mathbf{D}. \tag{4.22}$$

Equation (4.22) is from [16].

We first characterize the class of temporal logic formulas that corresponds to the pseudovariety $\mathbf{D}$.

LEMMA 4.4. $L(\mathtt{Pv}) = L(\mathbf{D})$.

Proof. A language $L \subseteq \Sigma_P^+$ belongs to $L(\mathbf{D})$ if and only if L is a boolean combination of singleton sets $\{u\}$ and languages of the form $\Sigma_P^* u$ for $u \in \Sigma_P^+$. This follows almost immediately from (4.15) and (4.16); see also [14].

For each $a \in \Sigma_P$ let α_a be a propositional formula such that $L(\alpha_P) = \Sigma_P^* a$. Assume u is a nonempty word over Σ_P^+ of length n. Then

$$L(\alpha_{u(n-1)} \wedge \ominus(\alpha_{u(n-2)} \wedge \ominus(\alpha_{u(n-3)} \wedge \dots (\alpha_{u(0)} \wedge \neg\ominus \textsc{True})))) = \{u\}. \tag{4.23}$$

Similarly,

$$L(\alpha_{u(n-1)} \wedge \ominus(\alpha_{u(n-2)} \wedge \ominus(\alpha_{u(n-3)} \wedge \dots \alpha_{u(0)}))) = \Sigma_P^* u. \tag{4.24}$$

Thus, $L(\mathbf{D}) \subseteq L(\mathtt{Pv})$.

For the other direction, recall that by Lemma 4.1, every formula in $\mathtt{Pv}$ is equivalent to a formula which is a boolean combination of formulas of the form $\ominus\ominus\ldots\ominus p$ or $\ominus\ominus\ldots\ominus\textsc{True}$. Now observe that

$$L(\ominus\ominus\ldots\ominus p) = \bigcup_u \Sigma^* u, \tag{4.25}$$

where u ranges over all words of length $n+1$ with $p \in u(n)$ and where n is the number of times the previously operator is applied in the formula on the left-hand side. Dropping the restriction "$p \in u(n)$" leads to an expression equivalent to $\ominus\ominus\ldots\ominus\textsc{True}$. This shows $L(\mathtt{Pv}) \subseteq L(\mathbf{D})$. □

Next, we characterize $\mathbf{MD_1}$.

LEMMA 4.5.

1. *A language over A is recognized by $\mathrm{B}(1,2)^1$ if and only if it is of the form A^*BC^* for some $B, C \subseteq A$.*

2. *A language over A is recognized by a semigroup in $\mathbf{MD_1}$ if and only if it is a boolean combination of languages of the form A^*BC^* for sets $B, C \subseteq A$.*

3. $L(\underline{\mathtt{Sn}}_1) = L(\mathbf{MD_1})$.

See also Lemma 1.3 in [4].

Proof. For the proof of part 1, let $h\colon A^+ \to \mathrm{B}(1,2)^1$ be a homomorphism. Set $M_s = h^{-1}(s) \cap A$ for every $s \in \mathrm{B}(1,2)^1$. Then $h^{-1}(a) = A^*M_aM_1^*$, $h^{-1}(b) = A^*M_bM_1^*$, and $h^{-1}(1) = M_1^+$. Therefore, every language recognized by $\mathrm{B}(1,2)^1$ is already of the form A^*BC^* for $B, C \subseteq A$. On the other hand, for every possible choice of $B, C \subseteq A$, the set A^*BC^* is recognized by $\mathrm{B}(1,2)^1$, as can be seen as follows. The claim is trivial if $B = \emptyset$. If $B \neq \emptyset$, consider the homomorphism $h\colon A^+ \to \mathrm{B}(1,2)^1$ defined by

$$h(c) = \begin{cases} a & \text{if } c \in B, \\ 1 & \text{if } c \in C \setminus B, \\ b & \text{otherwise.} \end{cases}$$

Then $h^{-1}(a) = A^*BC^*$.

For the proof of part 2 observe that since $\mathbf{MD_1}$ is generated by $\mathrm{B}(1,2)^1$, a language is recognized by an element of $\mathbf{MD_1}$ if and only if it is a boolean combination of languages recognized by $\mathrm{B}(1,2)^1$. Therefore the claim follows from part 1.

Part 3 is a trivial consequence of part 2 in view of the following equation, where $B = \{b_1, \ldots, b_m\}$ and $C = \{c_1, \ldots, c_n\}$ are (possibly empty) subsets of A.

$$L((c_1 \vee c_2 \vee \cdots \vee c_n)\ \mathsf{S}\ (b_1 \vee b_2 \vee \cdots \vee b_m)) = A^*BC^*. \tag{4.26}$$

Recall also that every formula in $\underline{\mathtt{Sn}}_1$ is equivalent to a boolean combination of formulas as in (4.26) and atomic formulas. □

The next pseudovariety we consider is $\mathbf{R}$. Let $\mathtt{WOn}$ ("$\mathtt{W}$" suggesting "weak") denote the class of formulas that are boolean combinations of formulas $\diamondsuit\varphi$ where φ belongs to $\mathtt{On}$. Similarly, let $\mathtt{WOn}_1$ denote the set of all formulas from $\mathtt{WOn}$ of nesting depth at most 1 in $\diamondsuit$; i.e., a formula belongs to $\mathtt{WOn}_1$ if and only if it is a boolean combination of formulas $\diamondsuit\varphi$ where φ is propositional.

LEMMA 4.6.

1. *A language over A^+ is recognized by* Sl_2 *if and only if it is of the form A^*BA^* for some $B \subseteq A$ or the complement of such a language.*
2. $L(\mathtt{WOn}_1) = L(\mathbf{SL})$.
3. $L(\mathtt{WOn}) \subseteq L(\mathbf{R})$.
4. $L(\mathbf{R}) \subseteq L(\mathtt{WOn} \star \mathtt{Pv})$.

Proof. The proof of part 1 follows the pattern of the proof of the first part of the previous lemma. Part 2 follows from part 1, as a language is a boolean combination of languages of the form A^*BA^* if and only if it is a boolean combination of languages of the form A^*bA^* for $b \in A$.

For the proof of part 3, we first show $L(\mathtt{WOn}_1^k) \subseteq L(\mathbf{SL}^k)$ for every $k > 0$. The induction base follows from part 2. For the inductive step, assume $L(\mathtt{WOn}_1^k) \subseteq L(\mathbf{SL}^k)$ for some k. Let Φ be the set of all PTL formulas φ such that $L(\varphi) \in L(\mathbf{SL}^k)$. Then, by the inductive assumption, $\mathtt{WOn}_1^k \subseteq \Phi$. Moreover,

$$\begin{array}{rcll} L(\mathtt{WOn}_1^{k+1}) & = & L(\mathtt{WOn}_1 \star \mathtt{WOn}_1^k) & \\ & \subseteq & L(\mathtt{WOn}_1 \star \Phi) & \text{Definition of } \Phi \\ & \subseteq & L(\mathbf{SL} * \mathbf{SL}^k) & \text{SSP I} \\ & = & L(\mathbf{SL}^{k+1}). & \end{array}$$

The claim now follows from (4.20).

For the proof of part 4, we first show $L(\mathbf{SL}^k) \subseteq L(\mathtt{WOn}_1^k \star \mathtt{Pv})$ for $k > 0$. The induction base follows from part 2. For the inductive step, assume $L(\mathbf{SL}^k) \subseteq L(\mathtt{WOn}_1^k \star \mathtt{Pv})$ for some k. Let Φ be the set of all formulas $\varphi \in \mathtt{WOn}_1^k \star \mathtt{Pv}$ such that $L(\varphi) \in L(\mathbf{SL}^k)$. Then, by the inductive assumption, $L(\mathbf{SL}^k) = L(\Phi)$. Moreover,

$$\begin{array}{rcll} L(\mathbf{SL}^{k+1}) & = & L(\mathbf{SL} * \mathbf{SL}^k) & \\ & \subseteq & L(\mathtt{WOn}_1 \star \mathtt{Pv} \star \Phi) & \text{SSP II} \\ & \subseteq & L(\mathtt{WOn}_1 \star \mathtt{Pv} \star \mathtt{WOn}_1^k \star \mathtt{Pv}) & \text{Definition of } \Phi \\ & = & L(\mathtt{WOn}_1 \star \mathtt{WOn}_1^k \star \mathtt{Pv}) & \text{Lemma 4.1} \\ & = & L(\mathtt{WOn}_1^{k+1} \star \mathtt{Pv}). & \end{array}$$

The claim of part 4 now follows from (4.20). □

It is not true that $L(\mathtt{WOn}) = L(\mathbf{R})$, and neither is it true that $L(\mathbf{R}) = L(\mathtt{WOn} \star \mathtt{Pv})$. Just note that $L(\ominus \diamondsuit p) \in L(\mathbf{R}) \setminus L(\mathtt{WOn})$ and $L(p) \in L(\mathtt{WOn} \star \mathtt{Pv}) \setminus L(\mathbf{R})$. We mention without proof that $L(\mathbf{R})$ is the set of all languages that are expressible via a formula which is a boolean combination of formulas $\diamondsuit\varphi$, where φ is a PTL formula using the combined operator $\ominus\diamondsuit$ as the only temporal operator.

4.3. Characterization of the hierarchies. We can now give an algebraic characterization of the since hierarchy.

THEOREM 4.7 (characterization of since hierarchy). *For every $k \geq 0$,*

$$L(\mathtt{Sn}_k) = L(\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D}). \tag{4.27}$$

The case $k = 0$ is Theorem 4.2 in [4]. Theorem 1.1 in Chapter 7 of [6] states $L((\mathbf{R} * \mathbf{MD_1})^k * \mathbf{LR}) \subseteq L(\mathtt{Sn}_k)$, which is the inclusion from right to left in (4.27).

A somewhat weaker version of the inclusion from right to left is Theorem 1.1 in Chapter 7 of [6].

Proof. By induction on k, we first prove

$$L(\underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) = L(\mathbf{MD}_1^k * \mathbf{D}) \tag{4.28}$$

for $k \geq 0$.

The induction base is Lemma 4.4. For the inductive step, assume (4.28) holds for some $k \geq 0$. Clearly, $L(\underline{\mathtt{Sn}}_1^{k+1} \star \mathtt{Pv}) = L(\underline{\mathtt{Sn}}_1 \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv})$. From Lemma 4.5, we know $L(\underline{\mathtt{Sn}}_1) = L(\mathbf{MD_1})$. Lemma 4.1 tells us $L(\underline{\mathtt{Sn}}_1 \star \mathtt{Pv} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) \subseteq L(\underline{\mathtt{Sn}}_1^{k+1} \star \mathtt{Pv})$. Using SSP, we obtain $L(\underline{\mathtt{Sn}}_1^{k+1} \star \mathtt{Pv}) = L(\mathbf{MD}_\mathbf{1}^{k+1} * \mathbf{D})$.

The rest of the proof consists of the following chain of inclusions, where Φ stands for the set of all formulas from $\mathtt{Sn}_0$ that are recognized by an element of $\mathbf{R}$:

$$\begin{array}{rcll}
L(\mathtt{Sn}_k) & = & L(\mathtt{On} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Theorem 4.2} \\
& = & L(\mathtt{WOn} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Definition of } \star \\
& \subseteq & L(\Phi \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Lemma 4.6} \\
& = & L(\mathbf{R} * \mathbf{MD}_\mathbf{1}^k * \mathbf{D}) & \text{SSP and (4.28)} \\
& \subseteq & L(\Phi \star \mathtt{Pv} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{SSP II} \\
& \subseteq & L(\mathtt{WOn} \star \mathtt{Pv} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Lemma 4.6} \\
& = & L(\mathtt{WOn} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Lemma 4.1} \\
& = & L(\mathtt{On} \star \underline{\mathtt{Sn}}_1^k \star \mathtt{Pv}) & \text{Definition of } \star \\
& = & L(\mathtt{Sn}_k) & \text{Theorem 4.2.}
\end{array}$$

This completes the proof. □

Using $\mathcal{C}^\rho$ to denote the class of all semigroups for which the reverse belongs to a given class $\mathcal{C}$, we immediately get the following theorem.

THEOREM 4.8 (characterization of until hierarchy). *For every $k \geq 0$,*

$$L(\mathtt{Ut}_k) = L((\mathbf{R} * \mathbf{MD}_\mathbf{1}^k * \mathbf{D})^\rho). \tag{4.29}$$

4.4. Algebraic implications. In this section, we deviate a little from our main route. The purpose is to explain what can be concluded from the previous theorem from a semigroup-theoretic point of view. We prove a switching rule and explain the connection to the Krohn–Rhodes theorem.

The following switching rule is slightly weaker than the switching rule from Theorem 9 in [20], which we used when we proved the decidability of the until hierarchy for the first time and which was derived from fundamental results in [16].

THEOREM 4.9. $\mathbf{MD_1} * \mathbf{R} \subseteq \mathbf{R} * \mathbf{MD_1} * \mathbf{D}$.

Proof. Clearly, $\mathbf{MD_1} * \mathbf{R} \subseteq \mathbf{MD_1} * \mathbf{R} * \mathbf{D}$. Thus, using SSP, Lemmas 4.5, and Theorem 4.7, $L(\mathbf{MD_1} * \mathbf{R} * \mathbf{D}) \subseteq L(\underline{\mathtt{Sn}}_1 \star \mathtt{Sn}_0)$. The latter is the same as $L(\mathtt{Sn}_1)$, which, by Theorem 4.7, is identical with $L(\mathbf{R} * \mathbf{MD_1} * \mathbf{D})$. The claim now follows from Eilenberg's variety theorem, which, among other things, states that if $\mathbf{V}$ and $\mathbf{W}$ are pseudovarieties of semigroups such that $L(\mathbf{V}) = L(\mathbf{W})$, then $\mathbf{V} = \mathbf{W}$. □

The next theorem gives an alternative description of $\mathbf{R} * \mathbf{MD}_\mathbf{1}^k * \mathbf{D}$.

THEOREM 4.10. *For $k \geq 0$,*

$$\mathbf{LR} * (\mathbf{MD_1} * \mathbf{LR})^k = \mathbf{R} * \mathbf{MD}_\mathbf{1}^k * \mathbf{D}. \tag{4.30}$$

Proof. The case $k = 0$ is (4.22). For $k > 0$ observe that $\mathbf{LR} = \mathbf{LR} * \mathbf{LR}$ by definition. Therefore $\mathbf{LR} * (\mathbf{MD_1} * \mathbf{LR})^k = (\mathbf{LR} * \mathbf{MD_1} * \mathbf{LR})^k$ for $k > 0$. By SSP, we have $L(\mathbf{LR} * \mathbf{MD_1} * \mathbf{LR}) = L(\mathtt{Sn}_0 \star \underline{\mathtt{Sn}}_1 \star \mathtt{Sn}_0)$. However, the latter is the same as $L(\mathtt{Sn}_1)$. We thus conclude $L(\mathbf{LR} * (\mathbf{MD_1} * \mathbf{LR})^k) = L(\mathtt{Sn}_k)$. This implies the claim in view of Theorem 4.7. □

The Krohn–Rhodes theorem states that the pseudovariety $\mathbf{A}$, the class of all finite semigroups containing no nontrivial group, is identical with $V^*(\{\mathrm{B}(1,2)^1\})$. In view of this, it is only natural to classify a semigroup S of $\mathbf{V}$ according to how often the

semidirect product operation and the use of direct products of $\mathrm{B}(1,2)^1$ alternate in a decomposition of S. We get a hierarchy of pseudovarieties, the kth level given by

$$V^*(\{\mathrm{B}(1,2), \mathrm{Sl}_2\}) * (V(\{\mathrm{B}(1,2)^1\}) * V^*(\{\mathrm{B}(1,2), \mathrm{Sl}_2\}))^k. \tag{4.31}$$

(Recall that B(1,2) and Sl_2 are the only nontrivial factors of $\mathrm{B}(1,2)^1$.) Using the notation introduced above, we can as well write $\mathbf{LR} * (\mathbf{MD_1} * \mathbf{LR})^k$. The previous theorem now tells us that this hierarchy corresponds to the since hierarchy of temporal logic.

4.5. Decidability. Before we can prove the decidability of the levels of the since hierarchy, we need to recall some facts from finite semigroup theory.

The first lemma gives a description of $\mathbf{MD_1}$ and is folklore.

LEMMA 4.11. *A finite semigroup belongs to* $\mathbf{MD_1}$ *if and only if it satisfies*

$$xyx = yx, \tag{4.32}$$

$$x^2 = x. \tag{4.33}$$

Recall that a pseudovariety $\mathbf{V}$ of semigroups is called locally finite if for every $m > 0$ there exists a finite semigroup $F_m \in \mathbf{V}$ such that every semigroup from $\mathbf{V}$ generated by at most m elements is a homomorphic image of F_m. The semigroup F_m is then unique up to isomorphism and is denoted by $F_m\mathbf{V}$.

We say a pseudovariety $\mathbf{V}$ of semigroups is *effectively locally finite* if it is locally finite and $m \mapsto F_m\mathbf{V}$ is computable.

The following proposition can be found, for instance, in [18], as Proposition 6.2, and in [1], as Theorem 10.2.1.

PROPOSITION 4.12. *If* $\mathbf{V}$ *and* $\mathbf{W}$ *are effectively locally finite pseudovarieties of semigroups, then* $\mathbf{V} * \mathbf{W}$ *is effectively locally finite.*

The next proposition is Theorem 5.7 from [18].

PROPOSITION 4.13 (Straubing). *If* $\mathbf{V}$ *is a pseudovariety of semigroups, then* $S \in \mathbf{V} * \mathbf{D}$ *if and only if* $S \in \mathbf{V} * \mathbf{D}_{|S|}$.

The last fact we need is an obvious consequence of results in [19] on graph congruences.

THEOREM 4.14 (Thérien,Weiss). *If* $\mathbf{V}$ *and* $\mathbf{W}$ *are pseudovarieties such that*

- $\mathbf{LV} = \mathbf{V} * \mathbf{D}$,
- $\mathbf{V}$ *is decidable, and*
- $\mathbf{W}$ *is effectively locally finite,*

then $\mathbf{V} * \mathbf{W}$ *is decidable.*

The decision procedure we have in mind is the obvious modification of the procedure described under (ii) in Example 2.5 of [19]. Its correctness follows from Theorem 2.8 in [19].

We can now approach the decidability of the since hierarchy.

PROPOSITION 4.15. *Let* $k \geq 0$, $l \geq 1$. *The pseudovariety* $\mathbf{MD}_1^k * \mathbf{D}_l$ *is effectively locally finite.*

Proof. From (4.15) it is clear that $F_m\mathbf{D}_l$ is finite for $m, l \geq 1$. From Lemma 4.11 it is clear that $\mathbf{MD_1}$ is effectively locally finite. Therefore, by repeated application of Proposition 4.12, we obtain that $\mathbf{MD}_1^k * \mathbf{D}_l$ is effectively locally finite. □

THEOREM 4.16. *Let* $k \geq 0$. *The problem of determining whether a given finite semigroup* S *belongs to* $\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D}$ *is decidable.*

Proof. In order to check whether $S \in \mathbf{R} * \mathbf{MD}_1^k * \mathbf{D}$, according to Proposition 4.13, we have only to check for $S \in \mathbf{R} * \mathbf{MD}_1^k * \mathbf{D}_{|S|}$. This is effective by Theorem 4.14 and Proposition 4.15 in view of (4.22). □

This finally yields the following theorem.

THEOREM 4.17 (decidability of until hierarchy).

1. *Let $k \geq 0$. The problems of determining whether a regular language belongs to $L(\mathtt{Ut}_k)$ is decidable.*

2. *The problem of determining the smallest number k such that a given regular language belongs to $L(\mathtt{Ut}_k)$ can be solved effectively.*

Proof. In order to determine whether $L \in L(\mathtt{Ut}_k)$ for a given regular language L, say, represented by a finite automaton, one first computes its minimal automaton $\mathfrak{A}$. The next step is to compute the transformation semigroup S of $\mathfrak{A}$. Using Theorem 4.16, it is then checked whether $S \in (\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D})^\rho$, which is also the answer to the question, as can be seen as follows. First, the semigroup S is isomorphic to the syntactic semigroup of L. Second, since $(\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D})^\rho$ is a pseudovariety of semigroups, $L \in L((\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D})^\rho)$ if and only if $S \in (\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D})^\rho$. Third, $L \in L(\mathtt{Ut}_k)$ if and only if $L \in L((\mathbf{R} * \mathbf{MD}_1^k * \mathbf{D})^\rho)$ by Theorem 4.8.

The second part follows immediately from part 1, as the decision procedures described in the proof of Theorem 4.16 are uniform in k, and one can a priori check whether a given language L is at all expressible in FTL. See [9, 15, 11]. □

COROLLARY 4.18. *The problem of determining the smallest number k such that the language defined by a given FTL formula belongs to $L(\mathtt{Ut}_k)$ can be solved effectively.*

Proof. It suffices to note that given an FTL formula φ it is easy to compute a finite automaton that recognizes the language defined by φ; see, for instance, [25] for the case of ω-words. □

Obviously, analogues of the previous two results hold for the since hierarchy.

5. Extension to ω-words. In this chapter we extend the algebraic characterization of the until hierarchy and its decidability to the situation where temporal logic formulas are interpreted in ω-words instead of strings (which is often enough the case, especially when temporal logic is used to specify the behavior of reactive systems).

It should be clear how the definitions of section 2 carry over to ω-words. To distinguish between the two settings we will write $L_\omega(\varphi)$ to denote the set of all ω-models of an FTL formula and keep writing $L(\varphi)$ for the set of all string models of an FTL formula.

5.1. Regular ω-languages. We recall some basic facts about the algebraic treatment of regular ω-languages.

Let $L \subseteq A^\omega$. The *syntactic congruence* of L, as defined in [2] and denoted by $\equiv_L$, is the binary relation on A^+ that relates u and v if and only if

$$(\forall x, y \in A^*)\,(x(uy)^\omega \in L \leftrightarrow x(vy)^\omega \in L) \tag{5.1}$$

and

$$(\forall x, y \in A^*)\,(\forall z \in A^+)\,(xuyz^\omega \in L \leftrightarrow xvyz^\omega \in L). \tag{5.2}$$

Given an arbitrary congruence $\equiv$ on A^+, we define a similarity relation $\sim$ on A^ω as follows: $\alpha \sim \beta$ if and only if there exist decompositions $\alpha = u_0u_1u_2\ldots$ and $\beta = v_0v_1v_2\ldots$ such that $u_i \equiv v_i$ for $i \geq 0$.

For a given language $L \subseteq A^\omega$, we denote by $\sim_L$ the similarity relation induced by $\equiv_L$. The following crucial property of the syntactic congruence of a regular ω-language goes back to [2] and [12].

LEMMA 5.1 (Arnold, Pécuchet). *Let $L \subseteq A^\omega$ be regular and $\sim_L$ the similarity relation induced by $\equiv_L$.*

If $\alpha \sim_L \beta$, then $\alpha \in L$ if and only if $\beta \in L$.

5.2. The Ehrenfeucht–Fraïssé game for FTL. As our main tool we will use a combinatorial description of the Ehrenfeucht–Fraïssé game of [8]. In fact, we will use the following definition of game equivalence.

Let $x, y \in \Sigma_P^+ \cup \Sigma_P^\omega$, $i < |x|$, and $j < |y|$.

We define $(x,i) \equiv_k^l (y,j)$ to hold if and only if the following conditions and their symmetric counterparts (where the roles of (x,i) and (y,j) are interchanged) are met.

Initial condition: $x(i) = y(j)$.

Next condition, required if $l > 0$: If $i+1 < |x|$, then $j+1 < |y|$ and $(x, i+1) \equiv_k^{l-1} (y, j+1)$.

Eventually condition, required if $l > 0$: For every i' with $i \le i' < |x|$, there exists j' with $j \le j' < |y|$ such that $(x,i') \equiv_k^{l-1} (y,j')$.

Until condition, required if $k > 0$: For every i' with $i \le i' < |x|$, there exists j' with $j \le j' < |y|$ such that

(UC1) if $i' = i$, then $j' = j$;

(UC2) $(x,i') \equiv_{k-1}^l (y,j')$;

(UC3) for every j'' with $j \le j'' < j'$, there exists i'' with $i \le i'' < i$ such that $(x,i'') \equiv_{k-1}^l (y,j'')$.

We write $x \equiv_k^l y$ for $(x,0) \equiv_k^l (y,0)$.

For a game-theoretic interpretation of $\equiv_k^l$, see [8].

To describe the connection of $\equiv_k^l$ with temporal logic, we need the notion of residual depth.

The *residual depth* of a FTL formula φ, denoted by $\mathrm{rd}(\varphi)$, is defined to be the depth of the formula in terms of the FTL operators other than U:

$$\begin{aligned}
\mathrm{rd}(\textsc{True}) &= 0,\\
\mathrm{rd}(p) &= 0 \qquad \text{for } p \in P,\\
\mathrm{rd}(\neg\varphi) &= \mathrm{rd}(\varphi),\\
\mathrm{rd}(\varphi_1 \vee \varphi_2) &= \max\{\mathrm{rd}(\varphi_1), \mathrm{rd}(\varphi_2)\},\\
\mathrm{rd}(\bigcirc\varphi) &= \mathrm{rd}(\varphi) + 1,\\
\mathrm{rd}(\Diamond\varphi) &= \mathrm{rd}(\varphi) + 1,\\
\mathrm{rd}(\varphi_1 \,\mathsf{U}\, \varphi_2) &= \max\{\mathrm{rd}(\varphi_1), \mathrm{rd}(\varphi_2)\}.
\end{aligned}$$

The set $\mathtt{Ut}_{k,l}$ is defined to be the set of all FTL formulas of until depth at most k and residual depth at most l.

The following theorem is an adapted version of Theorem 2.1 from [8].

THEOREM 5.2 (Etessami, Wilke; soundness and completeness). *Let $x, y \in \Sigma_P^+ \cup \Sigma_P^\omega$ and $k, l \ge 0$.*

Then $x \equiv_k^l y$ if and only if x and y are models of the same $\mathtt{Ut}_{k,l}$-formulas over P.

The following lemma states that game equivalence is a congruence relation; in fact, it is a Ramsey congruence in the sense of [26]. The proof is straightforward but lengthy. The interested reader is referred to our technical report [20].

LEMMA 5.3 (congruence property). *Let $k, l \geq 0$.*

1. *The binary relation $\equiv_k^l$ restricted to Σ_P^+ is a congruence relation. That is, $\equiv_k^l$ restricted to Σ_P^+ is an equivalence relation and whenever $u \equiv_k^l v$ and $u' \equiv_k^l v'$ for $u, u', v, v' \in \Sigma_P^+$, then $uu' \equiv_k^l vv'$.*
2. *Let $u, v \in \Sigma_P^+$ and $\alpha, \beta \in \Sigma_P^\omega$. If $u \equiv_k^l v$ and $\alpha \equiv_k^l \beta$, then $u\alpha \equiv_k^l v\beta$.*
3. *If $u, v \in \Sigma_P^+$ such that $u \equiv_k^l v$, then $u^\omega \equiv_k^l v^\omega$.*

To conclude this section about game equivalence, we note the following remark.

REMARK 2. *Let $k, l \geq 0$, $x, y \in \Sigma_P^\infty$, $i < |x|$, and $j < |y|$. Assume $u, v \in \Sigma_P^*$ and $(x, i) \equiv_k^l (y, j)$.*

Then $(ux, |u| + i) \equiv_k^l (vy, |v| + j)$.

5.3. Reduction to strings. Using game equivalence we can now characterize the levels of the until hierarchy for ω-words. We start with a lemma.

LEMMA 5.4. *Let $k, l \geq 0$ and $u, v \in \Sigma_P^+$ be such that $u \equiv_k^l v$. For all $x, y \in \Sigma_P^*$ and $z \in \Sigma_P^+$,*

$$x(uy)^\omega \equiv_k^l x(vy)^\omega, \tag{5.3}$$

$$xuyz^\omega \equiv_k^l xvyz^\omega. \tag{5.4}$$

Proof. Relation (5.4) follows immediately from Lemma 5.3 (part 2) and Remark 2.

By part 1 of the same lemma, we obtain $uy \equiv_k^l vy$. Thus, by part 3 of that lemma, $(uy)^\omega \equiv_k^l (vy)^\omega$. Another application of part 2 finally yields $x(uy)^\omega \equiv_k^l x(vy)^\omega$. □

The next proposition states that if a regular ω-language belongs to the kth level of the until hierarchy, then so do the classes of its syntactic congruence relation.

PROPOSITION 5.5. *Let $k \geq 0$ and $L \subseteq \Sigma_P^\omega$ such that $L \in L_\omega(\mathtt{Ut}_k)$. Each class of $\equiv_L$ belongs to $L(\mathtt{Ut}_k)$.*

Proof. There exists l such that L belongs to $L_\omega(\mathtt{Ut}_{k,l})$. We claim that every class of the syntactic congruence of L belongs to $L(\mathtt{Ut}_{k,l})$.

We need only to show that for all $u, v \in \Sigma_P^+$ the following holds. If $u \equiv_k^l v$, then $u \equiv_L v$. Therefore assume $u \equiv_k^l v$, and let $x, y \in \Sigma_P^*$ and $z \in \Sigma_P^+$.

Assume $x(uy)^\omega \in L$. From Lemma 5.4, we know $x(uy)^\omega \equiv_k^l x(vy)^\omega$. Thus, since $L \in L_\omega(\mathtt{Ut}_{k,l})$, $x(vy)^\omega \in L$. By symmetry, if $x(vy)^\omega \in L$, then $x(uy)^\omega \in L$. Similarly, using (5.4) instead of (5.3), one shows $xuyz^\omega \in L$ if and only if $xvyz^\omega \in L$. Therefore (5.1) and (5.2) hold, which means $u \equiv_L v$. □

The converse of the previous proposition is also true and is the crucial observation of this section. It is more difficult to prove, and we will dedicate almost the entire rest of this section to it.

LEMMA 5.6. *Let $k, l \geq 0$, $p = k + l + 1$, and $u, v \in \Sigma_P^*$ and $w \in \Sigma_P^+$ such that $uw^\omega \equiv_k^l vw^\omega$. Then $uw^p \equiv_k^l vw^p$.*

Proof. We prove by induction on $k' + l'$ that the following is true for $k' \leq k$, $l' \leq l$ and $i, j \geq 0$: if $(uw^\omega, i) \equiv_{k'}^{l'} (vw^\omega, j)$, $i \leq |u| + (k - k' + l - l')|w|$, and $j \leq |v| + (k - k' + l - l')|w|$, then $(uw^p, i) \equiv_{k'}^{l'} (vw^p, j)$.

Write x, y, α, and β for uw^p, vw^p, uw^ω, and vw^ω.

Induction base, $k' + l' = 0$. This is trivial.

Induction step, $k' + l' > 0$. We consider the various conditions.

Initial condition. Clearly, this condition is satisfied.

Next condition. Clearly, $(\alpha, i+1) \equiv_{k'}^{l'-1} (\beta, j+1)$, hence $(x, i+1) \equiv_{k'}^{l'-1} (y, j+1)$ by induction hypothesis.

Eventually condition. Let i' be such that $i \leq i' < |x|$. We distinguish two cases.

First case, $i < |u|$. Since $(\alpha, i) \equiv_{k'}^{l'} (\beta, j)$, there exists j' such that $(\alpha, i') \equiv_{k'}^{l'} (\beta, j')$. By Remark 2, we can assume $j' < |v| + (k - k' + l - l' + 1)|w|$. Thus we can apply the induction hypothesis and find $(x, i') \equiv_{k'}^{l'-1} (y, j')$.

Second case, $i' \geq |u|$. This case is simple, as we can choose $j' = |v| + i' - |u|$.

Until condition. Let i' be such that $i \leq i' < |x|$. Since $(\alpha, i) \equiv_{k'}^{l'} (\beta, j)$, there exists j' such that (UC1)–(UC3) hold for α, β, k', and l' instead of x, y, k, and l.

By Remark 2, we can find j_1 such that $j \leq j_1 < |v| + (k - k' + l - l' + 1)|w|$, $j_1 \leq j$, and $(\beta, j') \equiv_{k'-1}^{l'} (\alpha, j_1)$. We claim that (UC1)–(UC3) then hold for k', l', x, y, and j_1 instead of k, l, x, y, and j'.

(UC1) is trivially satisfied, as we know that if $i = i'$, then $j' = j$, and that $j \leq j_1 \leq j'$. By definition of j_1, we know $(\alpha, i') \equiv_{k'-1}^{l'} (\beta, j_1)$, hence, by induction hypothesis, $(x, i') \equiv_{k'-1}^{l'} (y, j_1)$. Therefore (UC2) is satisfied.

To verify (U3), let j'' be such that $j \leq j'' < j_1$. If $j'' \geq |v|$, we simply take $i'' = |u| + j'' - |v|$. In the other case, if $j'' < |v|$, there is i'' such that $i \leq i'' < i$ and $(\alpha, i'') \equiv_{k'-1}^{l'} (\beta, j'')$. We can assume $i'' < |u| + (k - k' + l - l' + 1)|w|$. The induction hypothesis then yields $(x, i'') \equiv_{k'-1}^{l'} (y, j'')$. ∎

We denote by $\sim_k^l$ the similarity relation induced by $\equiv_k^l$.

PROPOSITION 5.7. *Let* $k, l \geq 0$, *and* $s, t, u, v \in \Sigma_P^+$ *such that* $st^\omega \equiv_k^{l+2} uv^\omega$. *Then there exists* $\gamma \in \Sigma_P^\omega$ *such that* $st^\omega \sim_k^l \gamma \sim_k^l uv^\omega$.

Proof. Write α and β for st^ω and uv^ω, respectively.

We first argue that we can assume $t^\omega \equiv_k^l v^\omega$.

Since we have $st^\omega \equiv_k^{l+2} uv^\omega$, there exists j such that $(st^\omega, |s|) \equiv_k^{l+1} (uv^\omega, j)$. Consequently, there exists i' such that $(st^\omega, |s| + i') \equiv_k^l (uv^\omega, j + |u|)$. By Remark 2, we can assume $i' < |t|$ and $j < |v|$. Write v', v'', t', t'' for $v(0, i'')$, $v(i'', |v|)$, $t(0, j')$, and $t(j', |t|)$. Then $uv^\omega = uv'(v''v')^\omega$ and $st^\omega = st'(t''t')^\omega$, where $(v''v')^\omega \equiv_k^l (t''t')^\omega$. We can thus replace u by uv', v by $v''v'$, s by st', and t by $t''t'$ in order to achieve $t^\omega \equiv_k^l v^\omega$.

Let $p = k + l + 1$. Set $\gamma = s(t^p v^p)^\omega$. We prove that this choice is correct.

We first show $\alpha \sim_k^l \gamma$. Let $u_0 = st^p$, $v_0 = st^p$, $u_i = t^{2p}$ for $i > 0$, and $v_i = v^p t^p$ for $i > 0$. Then $\alpha = u_0 u_1 u_2 \ldots$ and $\gamma = v_0 v_1 v_2 \ldots$. Trivially, $u_0 \equiv_k^l v_0$. Moreover,

$$v^p t^\omega \equiv_k^l v^p v^\omega = v^\omega \equiv_k^l t^\omega = t^p t^\omega, \tag{5.5}$$

where the first equivalence follows from Lemma 5.3 and the second one holds by assumption. Therefore Lemma 5.6 yields $v^p t^p \equiv_k^l t^p t^p$, hence $u_i \equiv_k^l v_i$ for $i > 0$, thus $\alpha \sim_k^l \gamma$.

We finally show $\gamma \sim_k^l \beta$. Let $u_0 = st^p v^p$, $v_0 = uv^p$, $u_i = t^p v^p$ for $i > 0$, and $v_i = v^{2p}$ for $i > 0$. Then $\gamma = u_0 u_1 u_2 \ldots$ and $\beta = v_0 v_1 v_2 \ldots$. As in the previous paragraph, we obtain $u_i \equiv_k^l v_i$ for $i > 0$. In order to prove $u_0 \equiv_k^l v_0$, we first observe

$$t^\omega = st^\omega \equiv_k^l uv^\omega, \tag{5.6}$$

where the first equivalence follows from Lemma 5.3 and the second one holds by assumption. From Lemma 5.6, we can thus conclude $st^p v^p \equiv_k^l uv^p$, which means $u_0 \equiv_k^l v_0$. ∎

We can now prove the following theorem.

THEOREM 5.8 (characterization of until hierarchy over infinite sequences). *Let* $k \geq 0$ *and* $L \subseteq \Sigma_P^\omega$ *regular. The following are equivalent.*

- $L \in L_\omega(\mathtt{Ut}_k)$.
- *Each equivalence class of* $\equiv_L$ *belongs to* $L(\mathtt{Ut}_k)$.

Proof. One direction is Proposition 5.5. For the other direction, assume that each class of $\equiv_L$ belongs to $L(\mathtt{Ut}_k)$. Since L is regular, $\equiv_L$ has only finitely many classes, and there thus exists $l \geq 0$ such that each class of $\equiv_L$ belongs to $L(\mathtt{Ut}_{k,l})$. We will prove that L belongs to $L(\mathtt{Ut}_{k,l+2})$.

For $\alpha, \beta \in \Sigma_P^\omega$ such that $\alpha \equiv_k^{l+2} \beta$, we have to show that $\alpha \in L$ if and only if $\beta \in L$.

Since all classes of $\equiv_k^{l+2}$ on Σ_P^ω are regular ω-languages, it is enough to consider only ultimately periodic words; that is, we can assume that α and β are of the form st^ω and uv^ω, respectively. (Recall that every nonempty regular ω-language contains an ultimately periodic word.)

By Proposition 5.7, there exists γ such that $\alpha \sim_k^l \gamma \sim_k^l \beta$. Since we assume that every class of $\equiv_L$ belongs to $L_\omega(\mathtt{Ut}_k)$, $\equiv_L$ is at least as coarse as $\equiv_k^l$, which implies that we also have $\alpha \sim_L \gamma \sim_L \beta$. We can now apply Lemma 5.1 and obtain $\alpha \in L$ if and only if $\gamma \in L$ if and only if $\beta \in L$. □

We can finally prove the counterpart to Theorem 4.17 and Corollary 4.18.

THEOREM 5.9 (decidability of until hierarchy over infinite sequences).

1. *Let $k \geq 0$. The problem of determining whether a given regular ω-language belongs to $L^\omega(\mathtt{Ut}_k)$ is decidable.*

2. *The problem of determining the smallest number k such that a given regular ω-language belongs to $L^\omega(\mathtt{Ut}_k)$ can be solved effectively.*

3. *The problem of determining the smallest number k such that a given FTL formula is equivalent over ω-words to an FTL formula of until depth at most k can be solved effectively.*

Proof. In order to check whether a given regular ω-language L belongs to $L^\omega(\mathtt{Ut}_k)$ one first computes a representation of its syntactic congruence as a finite automaton $\mathfrak{A}$ (on strings). The next step is to compute the transformation semigroup S of $\mathfrak{A}$. Using Theorem 4.16, one then checks whether $S \in (\mathbf{R} * \mathbf{MD_1} * \mathbf{D})^\rho$, which is also the answer to the question, as can be seen as follows. The semigroup is equivalent to $\Sigma_P^+/\equiv_L$, the syntactic semigroup of L. Since $(\mathbf{R} * \mathbf{MD_1} * \mathbf{D})^\rho$ is a pseudovariety of semigroups, each class of $\equiv_L$ belongs to $L((\mathbf{R} * \mathbf{MD_1} * \mathbf{D})^\rho)$ if and only if $S \in (\mathbf{R} * \mathbf{MD_1} * \mathbf{D})^\rho$. Now recall that Theorem 4.8 says that $L((\mathbf{R} * \mathbf{MD_1} * \mathbf{D})^\rho) = L(\mathtt{Ut}_k)$ and that Theorem 5.8 says that $L \in L^\omega(\mathtt{Ut}_k)$ if and only if every equivalence class of $\equiv_L$ belongs to $L(\mathtt{Ut}_k)$. □

6. Problems. We would like to conclude this paper with a list of problems.

PROBLEM 1. *Determine the complexity of the decision problems considered in this paper. It is not even obvious that the procedure described in the proof of Theorem* 4.16 *has an elementary upper bound.*

PROBLEM 2. *Extend our results to the combined until/since hierarchy.*

PROBLEM 3. *Find a description of the kth level of the until hierarchy in terms of structural properties of the corresponding minimal automata, as for instance given in* [4] *for until depth* 0.

REFERENCES

[1] J. ALMEIDA, *Finite Semigroups and Universal Algebra*, Ser. Algebra 3, World Scientific, Singapore, 1995.

[2] A. ARNOLD, *A syntactic congruence for rational ω-languages*, Theoret. Comput. Sci., 39 (1985), pp. 333–335.

[3] J. A. BRZOZOWSKI AND R. KNAST, *The dot-depth hierarchy of star-free languages is infinite*, J. Comput. System Sci., 16 (1978), pp. 37–55.

[4] J. Cohen, D. Perrin, and J.-E. Pin, *On the expressive power of temporal logic*, J. Comput. System Sci., 46 (1993), pp. 271–294.
[5] R. S. Cohen and J. A. Brzozowski, *Dot-depth of star-free events*, J. Comput. System Sci., 5 (1971), pp. 1–16.
[6] J. Cohen-Chesnot, *Etude algébrique de la logique temporelle*, Ph.D. thesis, Université Paris 6, Paris, France, 1989.
[7] A. E. Emerson, *Temporal and modal logic*, in Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 995–1072.
[8] K. Etessami and Th. Wilke, *An until hierarchy for temporal logic*, in Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, NJ, 1996, pp. 108–117.
[9] J. A. W. Kamp, *Tense Logic and the Theory of Linear Order*, Ph.D. thesis, University of California, Los Angeles, CA, 1968.
[10] R. E. Ladner, *Application of model theoretic games to discrete linear orders and finite automata*, Information and Control, 33 (1977), pp. 281–303.
[11] R. McNaughton and S. Papert, *Counter-Free Automata*, MIT Press, Cambridge, MA, 1971.
[12] J.-P. Pécuchet, *Etude syntaxique des parties reconnaissable de mots infinis*, Theoret. Comput. Sci., 58 (1988), pp. 231–248.
[13] D. Perrin, *Recent results on automata on infinite words*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 176, M. P. Chytil and V. Koubek, eds., Springer, Berlin, 1984, pp. 134–148.
[14] J.-E. Pin, *Varieties of Formal Languages*, Plenum Press, New York, 1986.
[15] M. P. Schützenberger, *On finite monoids having only trivial subgroups*, Information and Control, 8 (1965), pp. 190–194.
[16] P. Stiffler, Jr., *Extensions of the fundamental theorem of finite semigroups*, Adv. in Math., 11 (1973), pp. 159–209.
[17] H. Straubing, *Varieties of Recognizable Sets Whose Syntactic Monoids Contain Solvable Groups*, Ph.D. thesis, University of California, Berkeley, CA, 1978.
[18] H. Straubing, *Finite semigroup varieties of the form* $\mathbf{V} * \mathbf{D}$, J. Pure Appl. Algebra, 36 (1985), pp. 53–94.
[19] D. Thérien and A. Weiss, *Graph congruences and wreath products*, J. Pure Appl. Algebra, 36 (1985), pp. 205–215.
[20] D. Thérien and Th. Wilke, *Temporal Logic and Semidirect Products: An Effective Characterization of the Until Hierarchy*, Technical report 96-28, DIMACS, Piscataway, NJ, 1996.
[21] D. Thérien and Th. Wilke, *Temporal logic and semidirect products: An effective characterization of the until hierarchy*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, IEEE, 1996, pp. 256–263.
[22] W. Thomas, *Star-free regular sets of* ω*-sequences*, Inform. and Control, 42 (1979), pp. 148–156.
[23] W. Thomas, *Classifying regular events in symbolic logic*, J. Comput. System Sci., 25 (1982), pp. 360–376.
[24] W. Thomas, *Automata on infinite objects*, in Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 133–191.
[25] M. Y. Vardi and P. Wolper, *Reasoning about infinite computations*, Inform. and Comput., 115 (1994), pp. 1–37.
[26] Th. Wilke, *An algebraic theory for regular languages of finite and infinite words*, Internat. J. Algebra Comput., 3 (1993), pp. 447–489.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 799–815

PARALLEL QUANTUM COMPUTATION AND QUANTUM CODES*

CRISTOPHER MOORE† AND MARTIN NILSSON‡

Abstract. We study the class **QNC** of efficient parallel quantum circuits, the quantum analog of **NC**. We exhibit several useful gadgets and prove that various classes of circuits can be parallelized to logarithmic depth, including circuits for encoding and decoding standard quantum error-correcting codes, or, more generally, any circuit consisting of controlled-not gates, controlled π-shifts, and Hadamard gates. Finally, while we note the exact quantum Fourier transform can be parallelized to linear depth, we conjecture that neither it nor a simpler "staircase" circuit can be parallelized to less than this.

Key words. quantum circuits, parallel computation, quantum complexity classes, quantum error-correcting codes, group theory

AMS subject classifications. 81P68, 68Q10, 68Q15

PII. S0097539799355053

1. Introduction.

Much of computational complexity theory has focused on the question of what problems can be solved in polynomial time. Shor's quantum factoring algorithm [22] suggests that quantum computers might be more powerful than classical computers in this regard, i.e., that **BQP** might be a larger class than **P**, or rather **BPP**, the class of problems solvable in polynomial time by a classical probabilistic Turing machine with bounded error.

A finer distinction can be made between **P** and the class **NC** of efficient parallel computation, namely the subset of **P** of problems which can be solved by a parallel computer with a polynomial number of processors in *polylogarithmic* time, $\mathcal{O}(\log^k n)$ time for some k, where n is the number of bits of the input [18]. Equivalently, **NC** problems are those solvable by Boolean circuits with a polynomial number of gates and polylogarithmic depth.

This distinction seems especially relevant for quantum computers, where decoherence makes it difficult to do more than a limited number of computation steps reliably. Since decoherence due to storage errors is essentially a function of time, we can avoid it by doing as many of our quantum operations at once as possible. (Then again, gate errors will typically get worse since parallel algorithms often involve more gates, and gate errors are typically the dominant source of decoherence in current models of quantum computation such as bulk-spin NMR [9]. In addition, using a large number of work qubits may worsen decoherence and storage errors as well.)

In this paper, we initiate the study of **QNC**, the quantum analog of **NC**, and prove a number of elementary results. Our main theorem is that circuits consisting of controlled-not gates, controlled π-shifts, and Hadamard gates can be parallelized to logarithmic depth. This includes circuits for encoding and decoding standard

*Received by the editors April 19, 1999; accepted for publication (in revised form) October 2, 2000; published electronically December 18, 2001. This work was supported in part by NSF grants ASC-9503162 and PHY-0071139. Preliminary versions of this work appeared in [16].

http://www.siam.org/journals/sicomp/31-3/35505.html

†Computer Science Department and Department of Physics and Astronomy, University of New Mexico, Albuquerque, NM 87131 (moore@cs.unm.edu). Current address: Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501.

‡Chalmers Tekniska Högskola and University of Göteborg, Göteborg, Sweden. Current address: Los Alamos National Laboratory (nilsson@lanl.gov).

quantum error-correcting codes. We end with a conjecture that neither an exact quantum Fourier transform nor a simple "staircase" circuit can be parallelized to less than linear depth. While this would not show that **QNC** $<$ **QP**, it would certainly suggest that not all quantum algorithms can be parallelized.

In another paper [17], one of us explores quantum analogs of $\mathbf{AC}^0$ and $\mathbf{ACC}^0$, the subsets of **NC** consisting of constant-depth circuits with AND, OR, and MOD gates of arbitrary fan-in. Some further results on $\mathbf{QAC}^0$ and $\mathbf{QACC}^0$ can be found in [11, 12].

2. Definitions. We define quantum operators and quantum circuits as follows.

DEFINITION 1. *A* quantum operator *on n qubits is a unitary rank-$2n$ tensor U where $U^{b_1 b_2 \dots b_n}_{a_1 a_2 \dots a_n}$ is the amplitude of the incoming and outgoing truth values being $a_1, a_2, \dots, a_n$ and $b_1, b_2, \dots, b_n$, respectively, with $a_i, b_i \in \{0, 1\}$ for all i. However, we will usually write U as a $2^n \times 2^n$ unitary matrix U_{ab} where a and b's binary representations are $a_1 a_2 \cdots a_n$ and $b_1 b_2 \cdots b_n$, respectively.*

A one-layer circuit *consists of the tensor product of one- and two-qubit gates, i.e., rank* 2 *and* 4 *tensors, or* 2×2 *and* 4×4 *unitary matrices. This is an operator that can be carried out by a set of simultaneous one-qubit and two-qubit gates, where each gate couples a disjoint set of qubits.*

A quantum circuit of depth k *is a quantum operator written as the product of k one-layer circuits.*

Here we are allowing arbitrary two-qubit gates. If we like, we can restrict this to *controlled-U gates*, of the form

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix},$$

or more stringently to the *controlled-not* gate

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

For these, we will call the first and second qubits the *input* and *target* qubits, respectively; however, they do not really leave the input qubit unchanged, since they entangle it with the target.

Since either of these can be combined with one-qubit gates to simulate arbitrary two-qubit gates [1], these restrictions would just multiply our definition of depth by a constant. The same is true if we wish to allow gates that couple $k > 2$ qubits as long as k is constant, since any k-qubit gate can be simulated by $2^{\mathcal{O}(k)}$ two-qubit gates [1].

In order to design a shallow parallel circuit for a given quantum operator, we want to be able to use additional qubits or "ancillae" as workspace in the computation, equivalent to additional processors in a parallel quantum computer. However, to be able to correctly measure the computer's output, we have to make sure that the ancillae qubits are not entangled with the qubits of the final state that we care about. The easiest way to ensure this is to demand that the ancillae start and end in a pure state $|0\rangle$, so that the desired operator appears as the diagonal block of the operator

performed by the circuit on the subspace where the ancillae are zero. This also allows ancillae to be reused, which is important if we are going to compose two separately parallelized operators.

Note that although we do not explicitly allow measurement as an operator in our circuit, measurements can be treated as interactions with additional ancillae which represent the environment.

Then in analogy with **NC** we propose the following definition.

DEFINITION 2. *Let F be a family of quantum operators, where F_n is a $2^n \times 2^n$ unitary matrix on n qubits. We say that F_n is* embedded *in an operator M with m ancillae if M is a $2^{m+n} \times 2^{m+n}$ unitary matrix which preserves the subspace where the ancillae are zero, and if M is identical to $F_n \otimes \mathbf{1}^{2^m}$ on this subspace.*

Then $\mathbf{QNC} = \cup_k \mathbf{QNC}^k$, where $\mathbf{QNC}^k$ is the class of operators parallelizable to $\mathcal{O}(\log^k n)$ depth with a polynomial number of ancillae. That is, F is in $\mathbf{QNC}^k$ if, for all n, F_n can be embedded in a circuit of depth at most $c_1 \log^k n$ with at most $c_2 n^j$ ancillae, where c_1, c_2, and j are constants.

Here we are defining **QNC** as a class of quantum operators. We can extend this to functions and decision problems by embedding them in reversible functions first.

DEFINITION 3. *If ϕ is a Boolean function with n inputs and k outputs, define its* reversible version *ϕ' as the reversible Boolean function on $n + k$ bits defined by $\phi'(x, y) = (x, y \oplus \phi(x))$, where $\oplus$ is bitwise exclusive-or. Thus ϕ' keeps the input x, and xors y with the output $\phi(x)$. We can write ϕ' as a $2^n \times 2^n$ matrix with Boolean values, and we say that a Boolean function is in* **QNC** *if its reversible version is.*

We can consider various definitions of probabilistic acceptance and obtain the classes **EQNC**, **BQNC**, and **PrQNC** in nomenclature drawn from [21, 25]. **EQNC** accepts exactly, i.e., with probability 1, if the input is in the language and 0 otherwise. **BQNC** accepts with two-sided bounded probability $P \geq 2/3$ if the input is in the language and $P \leq 1/3$ if it is not. **PrQNC** accepts with probability $P > 1/2$ if the input is in the language and $P \leq 1/2$ if it is not.

With this definition, it is easy to see that $\mathbf{NC} \subset \mathbf{EQNC}$, since **EQNC** includes reversible Boolean circuits composed, say, of Toffoli gates, and a Boolean circuit of depth d and width w can be converted to a reversible one of depth $2d - 1$ with wd ancillae. Each gate is given its own ancilla which is xored with the gate's output. After obtaining the output of the circuit, we go through the layers of the circuit in reverse order to return the ancillae to zero.

For an upper bound, it is clear that $\mathbf{QNC} \subset \mathbf{QP}$ for the various definitions of acceptance, but it would be very nice if a sufficiently uniform quantum circuit could be evaluated by a quantum Turing machine with space equal to the circuit's depth, where by "uniform" we mean that there is, say, a classical DLOGTIME algorithm for describing the set of gates of the circuit.[1] This would be a quantum analog of Borodin's result [2] that $\mathbf{NC}^k \subset \mathbf{DSPACE}(\log^k n)$, and results of Watrous [25] would then imply that

$$\mathbf{EQNC}^k \subset \mathbf{PrQNC}^k \subset \mathbf{PrQSPACE}(\log^k n) \subset \mathbf{DSPACE}(\log^{2k} n).$$

If this is true, **QNC** can be simulated by classical circuits with polylogarithmic depth and quasipolynomial width, i.e., $\mathcal{O}(2^{\log^{\mathcal{O}(1)} n})$. However, even in the classical probabilistic case an analog of Borodin's result is not known.

[1] It would be nice to define uniformity in terms of quantum machines rather than classical ones. This would presumably allow the circuit itself to be in a superposition of multiple topologies, which a physicist might call a "second quantized" computation.

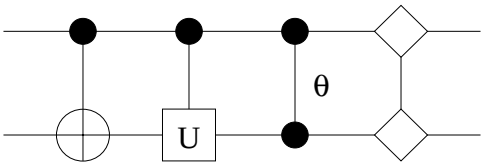


FIG. 1. *Our notation for controlled-not, controlled-U, symmetric phase shift, and arbitrary diagonal gates.*

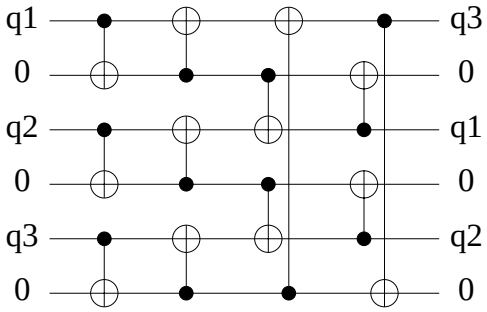


FIG. 2. *Permuting n qubits in four layers using n ancillae. In this example, the three qubits are rotated.*

We will use the notation in Figure 1 for our various gates: the controlled-not and controlled-U, the symmetric phase shift

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix},$$

and arbitrary diagonal gates

$$\begin{pmatrix} e^{i\omega_{00}} & 0 & 0 & 0 \\ 0 & e^{i\omega_{01}} & 0 & 0 \\ 0 & 0 & e^{i\omega_{10}} & 0 \\ 0 & 0 & 0 & e^{i\omega_{11}} \end{pmatrix}.$$

3. Permutations. In classical circuits, one can move wires around as much as one likes. In a quantum computer, it may be more difficult to move a qubit from place to place. However, we can easily do arbitrary permutations in constant depth.

PROPOSITION 1. *Any permutation of n qubits can be performed in four layers of controlled-not gates with n ancillae, or in six layers with no ancillae.*

Proof. The first part is obvious; simply copy the qubits into the ancillae, cancel the originals, recopy them from the ancillae in the desired order, and cancel the ancillae. An example is shown in Figure 2.

Without ancillae, we can use the fact that any permutation can be written as the composition of two involutions, i.e., two sets of disjoint transpositions. To see this, first decompose it into a product of disjoint cycles, and then note that a cycle is the composition of two reflections, as shown in Figure 3. Two qubits can be switched with three layers of controlled-not gates as shown in Figure 4, so any permutation

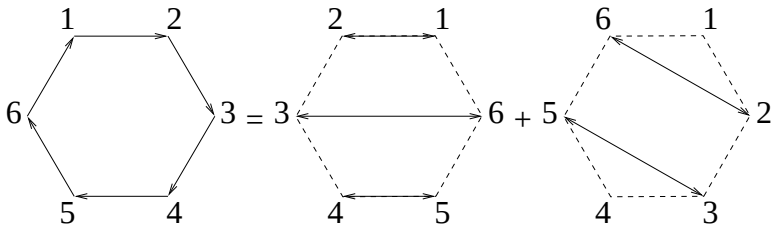


FIG. 3. *Any cycle, and therefore any permutation, is the composition of two sets of disjoint transpositions.*

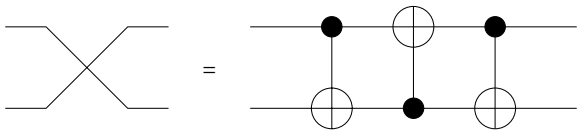


FIG. 4. *Switching two qubits with three controlled-nots.*

can be done in six layers. □

4. Fan-out. To make a shallow parallel circuit, it is often important to *fan out* one of the inputs into multiple copies. The controlled-not gate can be used to copy a qubit onto an ancilla in the pure state $|0\rangle$ by making a nondestructive measurement

$$(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \;\rightarrow\; \alpha|00\rangle + \beta|11\rangle.$$

Note that the final state is not a tensor product of two independent qubits, since the two qubits are completely entangled. Making an unentangled copy requires nonunitary, and in fact nonlinear, processes since

$$(\alpha|0\rangle + \beta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) = \alpha^2|00\rangle + \alpha\beta(|01\rangle + |10\rangle) + \beta^2|11\rangle$$

has coefficients quadratic in α and β. This is one form of the "no cloning" theorem of quantum mechanics [26].

This means that disentangling or *uncopying* the ancillae by the end of the computation, and returning them to their initial state $|0\rangle$, is a nontrivial and important part of a quantum circuit. There are, however, some special cases where this can be done easily.

Suppose we have a series of n controlled-U gates all with the same input qubit. Rather than applying them in series, we can *fan out* the input into n copies by splitting it $\log_2 n$ times, apply them to the target qubits, and uncopy them afterward, thus reducing the circuit's depth to $\mathcal{O}(\log n)$ depth.

PROPOSITION 2. *A series of n controlled-U gates with arbitrarily varying U coupling the same input to n target qubits can be parallelized to $\mathcal{O}(\log n)$ depth with $\mathcal{O}(n)$ ancillae.*

Proof. The circuit in Figure 5 copies the input onto $n-1$ ancillae, applies all the controlled-U gates simultaneously, and uncopies the ancillae back to their original state. Its total depth is $2\log_2 n + 1$. □

This kind of symmetric circuit, in which we uncopy the ancillae to return them to their original state, is similar to circuits designed by the reversible computation group at MIT [8] for reversible classical computers.

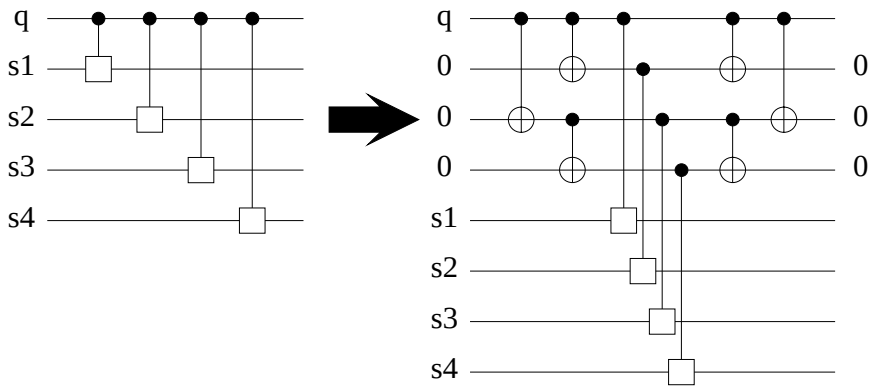


FIG. 5. *Parallelizing n controlled gates on a single input qubit q to $\mathcal{O}(\log n)$ depth.*

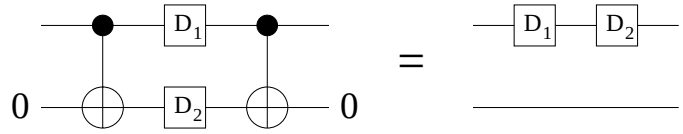


FIG. 6. *Using entanglement to parallelize diagonal operators.*

5. Fan-in: Diagonal and mutually commuting gates. Fan-in seems more difficult in general. Classically, we can calculate the composition of n operators in $\mathcal{O}(\log n)$ time by composing them in pairs; but since matrix multiplication is quadratic, it is unclear when we can do this with unitary linear operators. One special case where it is possible is if all the gates are diagonal.

PROPOSITION 3. *A series of n diagonal k-qubit operators on the same k qubit(s) can be parallelized to $\mathcal{O}(\log n) + 2^{\mathcal{O}(k)}$ depth with $\mathcal{O}(kn)$ ancillae.*

Proof. Here the entanglement between two copies of a qubit becomes an asset. Since diagonal matrices do not mix Boolean states with each other, we can act on one or more qubits and an entangled copy of them with two diagonal matrices D_1 and D_2 as in Figure 6. When we uncopy the ancilla(e), we have the same effect as if we had applied both matrices to the original qubit(s). Then the same kind of circuit as in Proposition 2 works, as shown in Figure 7, in which we make n copies of each of the k qubits, and apply the operators simultaneously. From [1] and Propositions 7 and 8 below, the depth of an arbitrary k-qubit diagonal operator is $2^{\mathcal{O}(k)}$. □

Since matrices commute if and only if they can be simultaneously diagonalized, we can generalize this to the case where a set of controlled-U gates applied to the same target qubit(s) have mutually commuting U's.

PROPOSITION 4. *A series of of n controlled-U gates acting on the same k target qubit(s), where the U's are mutually commuting operators on k qubits, can be parallelized to $\mathcal{O}(\log n) + 2^{\mathcal{O}(k)}$ depth with $\mathcal{O}(kn)$ ancillae.*

Proof. Since the U's all commute, they can be simultaneously diagonalized by some 2×2 unitary operator T. Apply $T^\dagger$ to the target qubit(s), parallelize the circuit using Proposition 3, and put the target qubit(s) back in the original basis by applying T. This is all done with a circuit of depth $2 \log_2 n + 2 \operatorname{depth}(T) + \operatorname{depth}(D)$. From [1] and Propositions 7 and 8 below, $\operatorname{depth}(T)$ and $\operatorname{depth}(D)$ are both $2^{\mathcal{O}(k)}$. □

As an example, in Figure 8 we show a circuit that applies the qth power of an

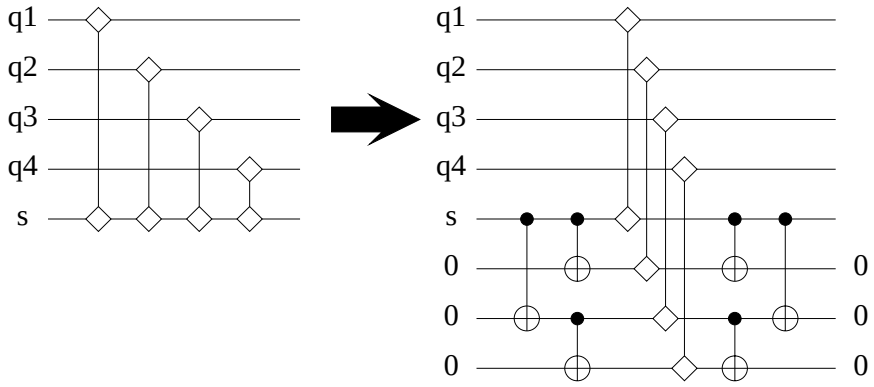


FIG. 7. *Parallelizing n diagonal gates on a single qubit as in Proposition* 2.

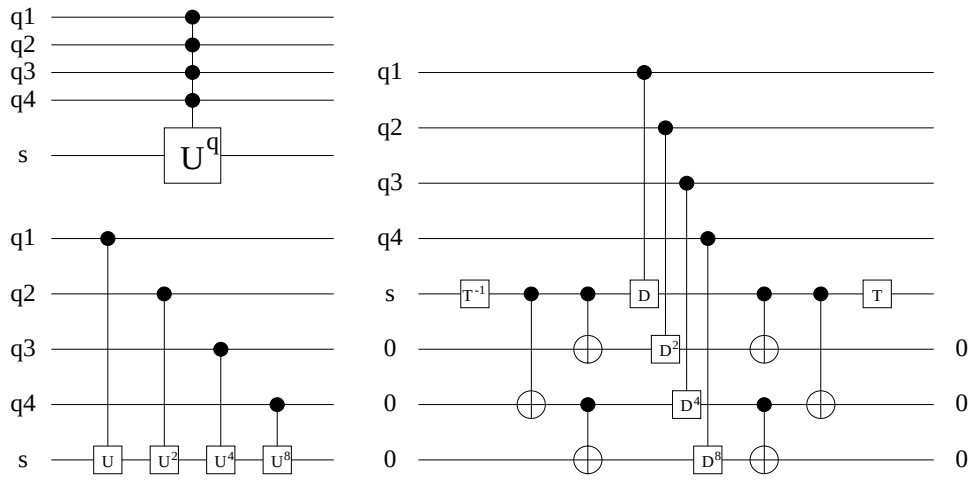


FIG. 8. *Applying an operator U q times, where q is given in binary by the input qubits.*

operator U to a target qubit, where $0 \leq q < 2^p$ is given by p input qubits as a binary integer. We can do this because $U^q = T^\dagger D^q T$.

We can extend this to circuits in general whose gates are mutually commuting, which includes diagonal gates. Note that when we say two gates commute, we mean as applied to particular qubits; for instance, controlled-not gates commute with each other if their input qubits or target qubits are the same, or if they are applied to disjoint pairs of qubits, but not if they overlap in other ways.

PROPOSITION 5. *A circuit of any size on n qubits, consisting of diagonal or mutually commuting gates, each of which couples at most k qubits, can be parallelized to depth* $\mathcal{O}(n^{k-1})2^{\mathcal{O}(k)}$ *with no ancillae, and to depth* $\mathcal{O}(k \log n) + 2^{\mathcal{O}(k)}$ *with* $\mathcal{O}(n^k)$ *ancillae. Therefore, any family of such circuits with constant k is in* $\mathbf{QNC}^1$.

Proof. Since all the gates commute, we can sort them by which qubits they couple and obtain a compressed circuit with one gate for each k-tuple. This gives $\binom{n}{k} = \mathcal{O}(n^k)$ gates, but by performing groups of n/k disjoint gates simultaneously we can do all of them in $\mathcal{O}(n^{k-1})$ layers.

By making $\frac{k}{n}\binom{n}{k} = \mathcal{O}(n^{k-1})$ of each qubit, we can apply each of these $\mathcal{O}(n^k)$

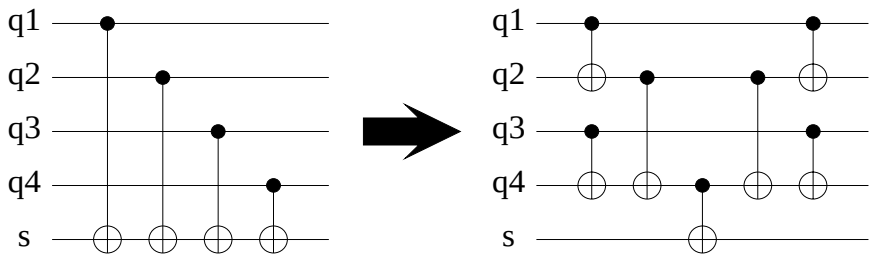


FIG. 9. *Parallelizing n controlled-not gates to $\mathcal{O}(\log n)$ depth by adding them in pairs.*

gates to a disjoint set of copies as in Propositions 3 and 4. This takes $\mathcal{O}(k \log n)$ layers of controlled-not gates.

As in Propositions 3 and 4, the layers coupling a given set of k qubits in the first case, and the operators T and D in the second case, both have depth $2^{\mathcal{O}(k)}$ in general. This is less obvious in the case of a mutual diagonalization operator T for a pair of overlapping but commuting gates. Suppose that two commuting k-qubit gates U and V overlap on some qubits we index with j, m, so that we can write them as tensor products along two different "boundaries," $U_{ij}^{lm}\delta_k^n$ and $\delta_i^l V_{jk}^{mn}$, where δ is the identity matrix. We state without proof that they can then both be diagonalized by an operator of the form $T_{ijk}^{lmn} = X_i^l Y_j^m Z_k^n$. Since this is the tensor product of three operators, each operating on k or fewer qubits, T has depth $2^{\mathcal{O}(k)}$ as before. This completes the proof. □

It comes as no surprise that gates can be performed simultaneously if they commute with each other. Proposition 5 simply confirms this intuition. Note that we could allow $k = \mathcal{O}(\log \log n)$ if we are willing to have a quasi-polynomial number of ancillae, or if the circuit is of polynomial size, in which case there are only a polynomial number of ancillae. In both cases the depth remains polylogarithmic.

6. Circuits of controlled-not gates. We can also fan in controlled-not gates. Figure 9 shows how to implement n controlled-not gates on the same target qubit in depth $2 \log_2 n + 1$. We construct the exclusive-or of subsets of the inputs by combining them in pairs.

We can use a generalization of this circuit to show that any circuit composed entirely of controlled-not gates can be parallelized to logarithmic depth.

PROPOSITION 6. *A circuit of any size on n qubits composed entirely of controlled-not gates can be parallelized to $\mathcal{O}(\log n)$ depth with $\mathcal{O}(n^2)$ ancillae. Therefore, any family of such circuits is in* $\mathbf{QNC}^1$.

Proof. First, note that there are a finite number of such circuits for a given n. In any circuit of controlled-not gates, if the n input qubits have binary values and are given by an n-dimensional vector q, then the output can be written Mq, where $M \in GF(2)$ is an $n \times n$ matrix over the integers mod 2. Each of the output qubits can be written as a sum of up to n inputs, $(Mq)_i = \sum_k q_{j_k}$, where the j_k are those j for which $M_{ij} = 1$.

We can break these sums down into binary trees. Let W_n be the complete output sums, $W_{n/2}$ be their left and right halves consisting of up to $n/2$ inputs, and so on down to single inputs. There are less than n^2 such intermediate sums W_k with $k > 1$. We assign an ancilla to each one, and build them up from the inputs in $\log_2 n$ stages, adding pairs from W_k to make W_{2k}. The first stage takes $\mathcal{O}(\log n)$ time and an

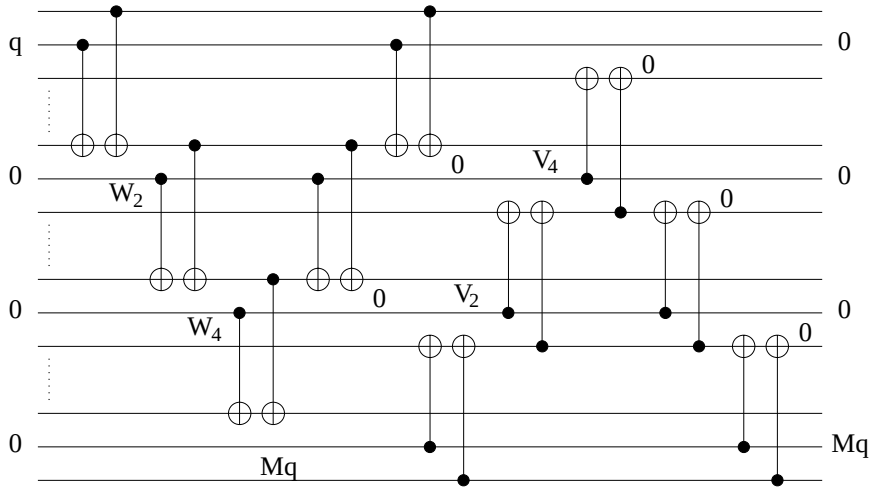


FIG. 10. *Parallelizing an arbitrary circuit of controlled-not gates to logarithmic depth.*

additional $\mathcal{O}(n^2)$ ancillae, since we may need to make $\mathcal{O}(n)$ copies of each input, but each stage after that can be done in depth 2.

To cancel the ancillae, we use the same cascade in reverse order, adding pairs from W_k to cancel W_{2k}. This leaves us with the input q, the output Mq, and the ancillae set to zero.

Now we use the fact that, since the circuit is unitary, M is invertible. Thus we can recalculate the input $q = M^{-1}(Mq)$ and cancel it. We use the same ancillae in reverse order, building the inputs q out of Mq with a series of partial sums $V_2, V_4, \ldots$, cancel q, and cancel the ancillae in reverse as before. All this is illustrated in Figure 10.

This leaves us with the output Mq and all other qubits zero. With four more layers as in Proposition 1, we can shift the output back to the input qubits, and we are done. □

This result is hardly surprising; after all, these circuits are reversible Boolean circuits, and any classical circuit composed of controlled-not gates is in $\mathbf{NC}^1$ (in fact, in the class $\mathbf{ACC}^0[2]$ of constant-depth circuits with sum mod 2 gates of unbounded fan-in). We just did a little extra work to disentangle the ancillae.

7. Controlled-not gates and phase shifts. We have shown that circuits composed of diagonal or controlled-not gates can be parallelized. It is reasonable to ask whether Propositions 5 and 6 can be combined, that is, whether arbitrary circuits composed of controlled-not gates and diagonal operators can be parallelized to logarithmic depth. In this section, we will show that this is not the case.

PROPOSITION 7. *Any diagonal unitary operator on n qubits can be performed by a circuit consisting of an exponential number of controlled-not gates and one-qubit diagonal gates and no ancillae.*

Proof. Any diagonal unitary operator on n qubits consists of 2^n phase shifts,

$$\begin{pmatrix} e^{i\omega_{000}} & & \\ & \ddots & \\ & & e^{i\omega_{111}} \end{pmatrix}.$$

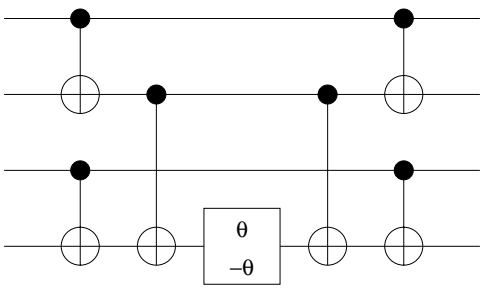


FIG. 11. *A circuit for the phase shift $\theta\mu_s$, i.e., a phase shift of $+\theta$ if the number of true qubits is even and $-\theta$ if it is odd.*

If we write the phase angles as a 2^n-dimensional vector ω, then the effect of composing two diagonal operators is simply to add these vectors mod 2π.

For each subset s of the set of qubits, let μ_s be the 2^n-dimensional vector whose ith component is $+1$ if the number of true qubits in $i \cap s$ is even, and -1 if it is odd. If $s = \{1 \cdots n\}$, for instance, μ_s is the aperiodic Morse sequence $(+1, -1, -1, +1, \ldots)$ when written out as a 2^n-dimensional vector, but it really just means giving the odd and even corners of the Boolean n-cube opposite signs.

It is easy to see that the μ_s for all $s \subset \{1 \cdots n\}$ are linearly independent and form the Hadamard, or Fourier, basis of $\mathbb{R}^{2^n}$. Moreover, while diagonal gates coupling k qubits can only perform phase shifts spanned by those μ_s with $|s| \leq k$, a circuit like that in Figure 11 can perform a phase shift proportional to μ_s for any given s (incidentally, in depth $\mathcal{O}(\log |s|)$ with no ancillae). Therefore, a series of 2^n such circuits, one for each subset of $\{1 \cdots n\}$, can express any diagonal unitary operator. □

This exponential bound is necessary in the worst case.

PROPOSITION 8. *There are diagonal operators that cannot be parallelized to less than exponential depth with a polynomial number of ancillae.*

Proof. Consider setting up a mapping between circuits and operators. A diagonal unitary operator on n qubits is described by 2^n eigenvalues which lie on the unit circle, so the set of such operators is isomorphic to a 2^n-dimensional torus. On the other hand, a circuit of depth d with m ancillae is described by only $\mathcal{O}(d(m+n))$ complex numbers, and the circuit can take one of $2^{\mathcal{O}(d(m+n)\log(m+n))}$ discrete topologies. Thus the set of such circuits is isomorphic to a union of $2^{\mathcal{O}(d(m+n)\log(m+n))}$ complex manifolds, each of which has $\mathcal{O}(d(m+n))$ dimensions. Since the map between circuits and operators is easily seen to be continuous, in order for this map to be surjective we need $2^n = \mathcal{O}(d(m+n))$ and so at least one of m or d must be exponential in n. □

However, the next proposition shows that this will not help us distinguish **QP** from **QNC**. In fact, for circuits consisting of controlled-nots and diagonal gates, **QP** and **QNC** are identical.

PROPOSITION 9. *Any circuit on n qubits consisting of controlled-not gates and m diagonal operators coupling k qubits each can be parallelized to $\mathcal{O}(2^k \log n)$ depth with $\mathcal{O}(\max(kmn, n^2))$ ancillae, using gates that couple a constant number of qubits. Therefore, any such circuit of polynomial size $\mathcal{O}(n^c)$ can be parallelized to $\mathcal{O}(2^k \log n)$ depth with $\mathcal{O}(kn^{c+1})$ ancillae, any family of such circuits with constant k is in $\mathbf{QNC}^1$, and any family with $k = \mathcal{O}(\log \log n)$ is in* **QNC**.

Proof. Any such circuit can be written as the product of a circuit of controlled-

not gates and a diagonal matrix that takes care of the phase shifts. The first part we can parallelize as in Proposition 6 to $\mathcal{O}(\log n)$ depth and $\mathcal{O}(n^2)$ ancillae. However, Proposition 8 shows that diagonal matrices cannot be parallelized in general, so we have to look at the circuit more closely.

We can write the circuit we are trying to parallelize as a product $M = M_0 P_1 M_1 P_2 M_2 \cdots P_m M_m$, where the M_i consist only of controlled-not gates and the P_i are the diagonal operators. By passing the P's through the M's to the right end of the circuit, we can write

$$M = M_0 \cdots M_m \cdot D_1 \cdots D_m,$$

where D_i is the diagonal operator

$$D_i = (M_i \cdots M_m)^\dagger P_i (M_i \cdots M_m).$$

In other words, D_i calculates what state the controlled-not circuit was in when P_i was applied, applies it, and uncalculates.

Each one of the k qubits coupled by P_i is the exclusive-or of some subset of the inputs and can be calculated with a binary tree of $\mathcal{O}(n)$ ancillae as in Proposition 6. Finally, by Proposition 3 we can apply all the D_i at once, by making m copies of the system's entire state. Thus the total number of ancillae needed is $\mathcal{O}(kmn)$, or $\mathcal{O}(kn^{c+1})$ if $m = \mathcal{O}(n^c)$.

Finally, Propositions 7 and 8 show that each diagonal gate takes depth $\mathcal{O}(2^k)$ to perform with diagonal gates coupling a constant number of qubits. Thus the depth is $\mathcal{O}(\log n)$ if k is constant, and $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ if $k = \mathcal{O}(\log \log n)$. □

8. The Hadamard gate and quantum codes. So far, all the circuits we have looked at are essentially classical; each row and each column has only one nonzero entry, so they are just reversible Boolean functions with phase shifts. Obviously, any interesting quantum algorithm will involve mixing between different Boolean states.

The simplest such operator is the *Hadamard gate* $R = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$.By applying it to all n qubits of a state $|000\cdots0\rangle$, we can prepare them in a superposition of all 2^n possible states. It is used, along with phase shifts, in the standard circuit (shown below in Figure 19) for the quantum Fourier transform; in fact, it is itself the quantum Fourier transform on $\mathbb{Z}_2$.

We will call a controlled-U gate a *controlled-Pauli gate* if U is one of the Pauli matrices $\sigma_x = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $-i\sigma_y = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$, or $\sigma_z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. Note that a controlled-X is simply a controlled-not, a controlled-Z is just the symmetric π-shift, and this real version of the controlled-Y is their product.

The π-shift can be written in terms of a controlled-not by conjugating the target with R. Conjugating the input qubit instead gives us the π-shift in the Hadamard basis, which is a symmetric gate

$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{pmatrix}.$$

We call this the w-gate, pronounced "wiggle," and notate it as in Figure 12.

Then we have the following proposition.

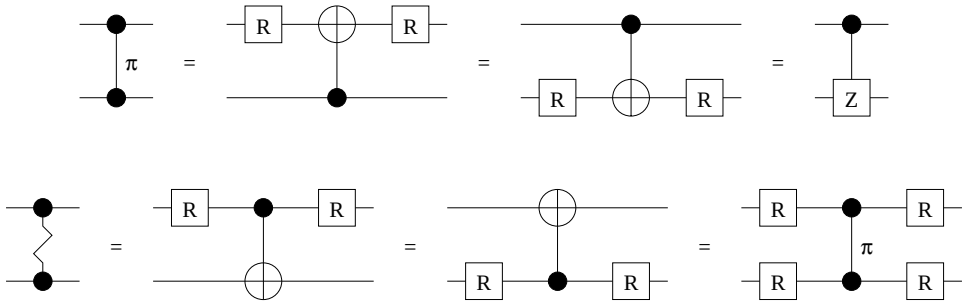


FIG. 12. *Relations between the π-shift, the controlled-not, and the w gate, which we notate with a wiggle.*

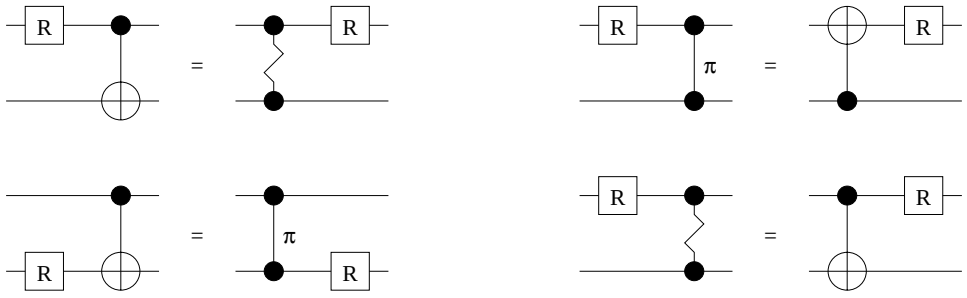


FIG. 13. *Step 1: combing R's to the right through controlled-nots, π-shifts, and w gates.*

PROPOSITION 10. *Circuits of any size consisting of controlled-Pauli gates and the Hadamard gate R can be parallelized to $\mathcal{O}(\log n)$ depth with $\mathcal{O}(n^2)$ ancillae. Thus any family of such circuits is in* $\mathbf{QNC}^1$.

Proof. We will use the algebraic relations between these gates to arrange them into easily parallelizable groups. The relations shown in Figures 12–15 can be verified by multiplying the 4×4 or 8×8 matrices corresponding to these gates.

In step 1, we move Hadamard gates to the right through the other gates as shown in Figure 13. Since $R^2 = \mathbf{1}$, this leaves a circuit of controlled-nots, π-shifts, and w-gates, followed by a single layer of R's and identities.

In step 2, we arrange π-shifts and w-gates into three groups: a set of π-shifts, a set of w's, and another set of π-shifts, with controlled-nots interspersed throughout. We can do this because these gates commute when applied to different pairs of qubits (up to the creation of an additional controlled-not) and generate a finite group when applied to the same pair. Specifically, if we call the 4×4 matrix of the π-shift z, then z and w obey the relations $z^2 = w^2 = 1$ and $wzw = zwz$, and generate the permutation group on three elements $S_3 = \{1, z, w, zw, wz, zwz\}$. Thus a group of z's, a group of w's, and a group of z's are sufficient to generate all possible products of these. These relations are shown in Figure 14.

In step 3, we pull the controlled-nots to the left through the z's and w's as shown in Figure 15. This makes some additional symmetric gates, but always of the same type we pull through, so the grouping of z's, w's, and z's is not disturbed. (We also sometimes create single-qubit gates X and Z, but these can be thought of as controlled-nots or π-shifts whose control qubit is always true.)

Finally, we note that since w is simply z in the Hadamard basis as shown in

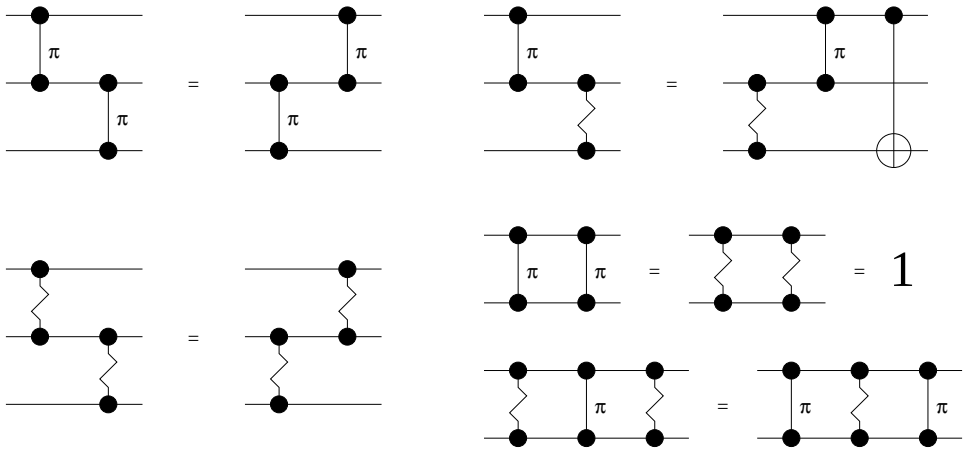


FIG. 14. *Step 2: commuting π-shifts and w's past each other, and combining them into 1, z, w, zw, wz, or zwz.*

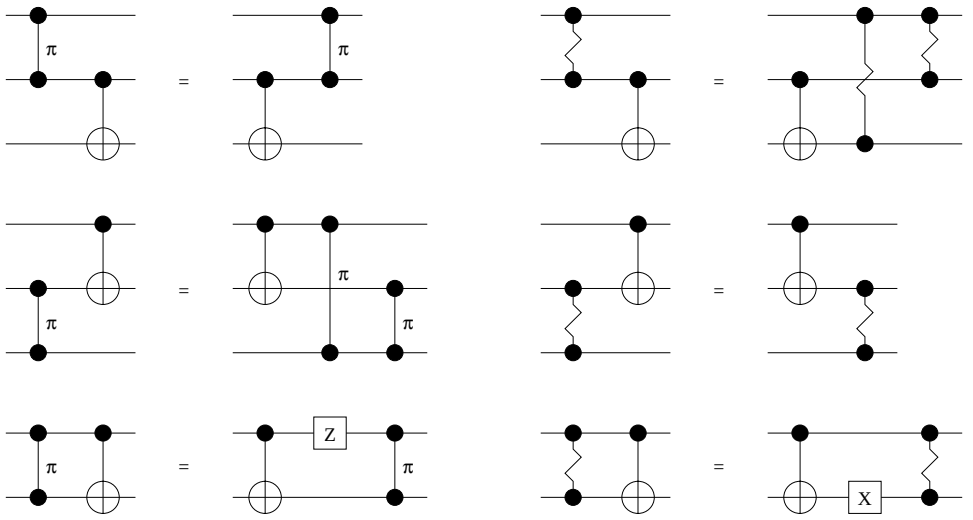


FIG. 15. *Step 3: commuting π-shifts and w's past controlled-nots.*

Figure 12, we can write the group of w's as a group of z's conjugated with R on every qubit. We are left with a circuit of controlled-not gates, followed by three groups of π-shifts separated by two layers of R's, and a single layer of possible R's as shown schematically in Figure 16.

Propositions 5 and 6 show how to parallelize circuits of π-shifts and of controlled-nots to $\mathcal{O}(\log n)$ depth with $\mathcal{O}(n^2)$ ancillae, and the theorem is proved. □

With a little extra work we should also be able to include the one-qubit $\pi/2$ shift $P = \left(\begin{smallmatrix} 1 & 0 \\ 0 & i \end{smallmatrix}\right)$, also known as the "square-root-of-not." This would give us the *Clifford group*, which is the normalizer of the group of tensor products of Pauli matrices. In fact, some of the relations we have used here are equivalent to those used by Gottesman to derive the Heisenberg representation of circuits in the Clifford group [10]. Figure 17 shows some relations between the P gate and the controlled-not, π-shift, and w, but so far we have not found a way to use these in a way analogous to Proposition 10.

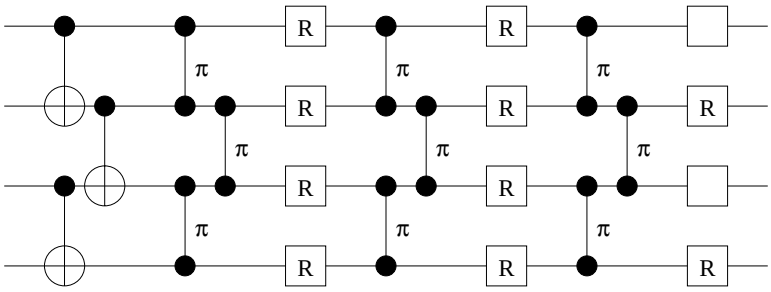


FIG. 16. *The kind of circuit we are left with after steps* 1, 2, *and* 3, *and after writing the* w*'s as* π*-shifts conjugated by* R.

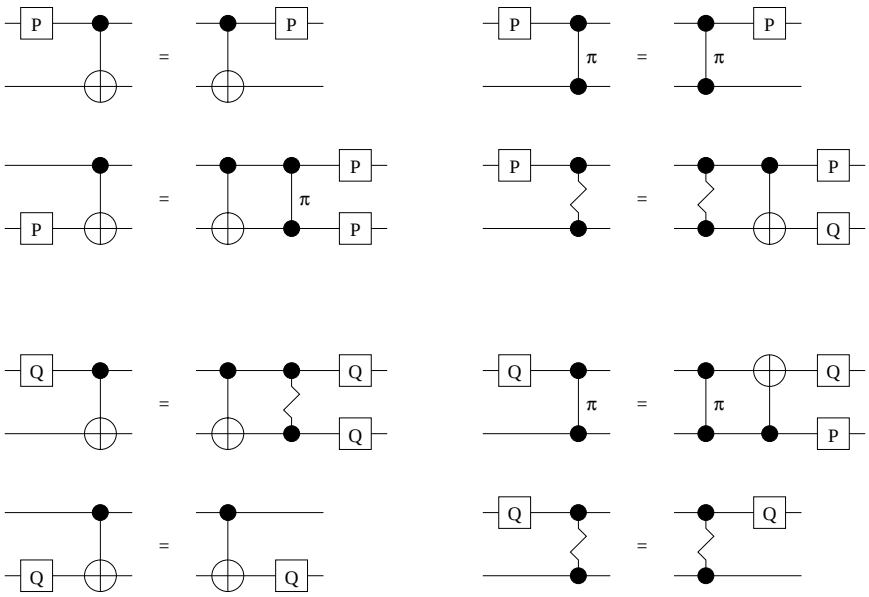


FIG. 17. *Relations between the* P *and the controlled-not,* π*-shift, and* w. *Here* $Q = RPR$, *the* P *gate conjugated with a Hadamard gate.*

The reader can experience our frustration by trying to use these relations to move two P gates, one on each qubit, rightward through a w. The result is an endless loop. In fact, computational search shows that this circuit cannot be written as any product of controlled-nots and π-shifts followed by any pair of one-qubit gates. This indicates that P gates are more difficult to pull out of a circuit than, say, Hadamard gates are. However, we are hopeful enough to conjecture that Proposition 10 can be generalized to include the P gate as well.

There may be other interesting finite subgroups of $O(2^n)$ that we can parallelize. However, even small additions to the Clifford group result in a dense set of operators and hence universal computation. This is because the controlled-Z gate is the only two-qubit phase shift whose conjugate by a controlled-not can be expressed with two- and one-qubit gates, just as P and Z are the only one-qubit phase shifts whose conjugate by a controlled-not can be expressed with themselves and controlled-Z gates. Other phase shifts generate three- and more-qubit interactions when they are

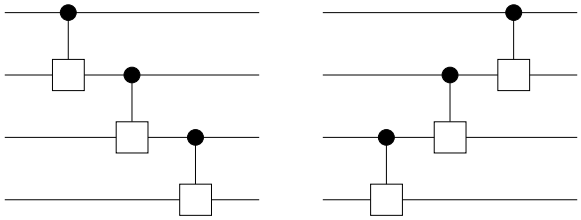


FIG. 18. *These "staircase" circuits seem hard to parallelize unless the operators are purely diagonal or off-diagonal.*

commuted through controlled-nots. The controlled-P gate, for instance, generates the Toffoli gate [1] and hence irrational rotations and a dense set of operators [19].

In any case, Proposition 10 gives us the following corollary.

COROLLARY 1. *Additive (or "stabilizer") quantum error-correcting codes are in* $\mathbf{QNC}^1$*, in the sense that encoding and decoding families of such codes with* n*-qubit code words can be done in* $\mathcal{O}(\log n)$ *depth and* $\mathcal{O}(n^2)$ *ancillae.*

Proof. Since the Pauli matrices σ_x and σ_z generate bit errors and phase errors, respectively, circuits for quantum codes such as those in [23, 14, 3, 7] are composed of controlled-Pauli and Hadamard gates. By a result of Rains [20], additive quantum codes are always equivalent to real ones, so the real version of the controlled-Y gate is sufficient. □

In fact, Cleve and Gottesman [4] and Steane [24] have shown that circuits for additive quantum codes can be constructed out of controlled-Pauli gates, where Hadamard gates appear only in one or two layers. Thus Proposition 9 is already enough to parallelize these circuits.

9. The quantum Fourier transform and the staircase circuit. A simple, perhaps minimal, example of a quantum circuit that seems hard to parallelize is the "staircase" circuit shown in Figure 18. This kind of structure appears in the standard circuit for the quantum Fourier transform (QFT), which has $\mathcal{O}(n^2)$ gates [6, 22]. Careful inspection shows that the QFT can in fact be parallelized to $\mathcal{O}(n)$ depth as shown in Figure 19 (an upside-down version of which appears in [13]), but it seems difficult to do any better.

Cleve and Watrous [5] have shown that fast parallel circuits exist for an approximate QFT, with error small enough to implement Shor's factoring algorithm; this shows that factoring is in $\mathbf{ZPP}^{\mathbf{BQNC}}$ and leaves iterated exponentiation as the main bottleneck. They also showed that any constant-error circuit must have depth at least $\mathcal{O}(\log n)$. However, the question of whether the *exact* QFT can be parallelized remains open, and we give the following conjecture.

Conjecture 1. Staircase circuits composed of controlled-U gates other than diagonal or off-diagonal gates (i.e., other than the special cases handled in Propositions 5 and 6) cannot be parallelized to less than linear depth, and neither can the exact quantum Fourier transform.

Since many different quantum operators can be used to solve a problem, proving this conjecture would *not* establish that $\mathbf{QNC} < \mathbf{QP}$ (which, if it were true, would also imply $\mathbf{NC} < \mathbf{PSPACE}$). However, it would certainly shed light on why quantum algorithms can be difficult to parallelize.

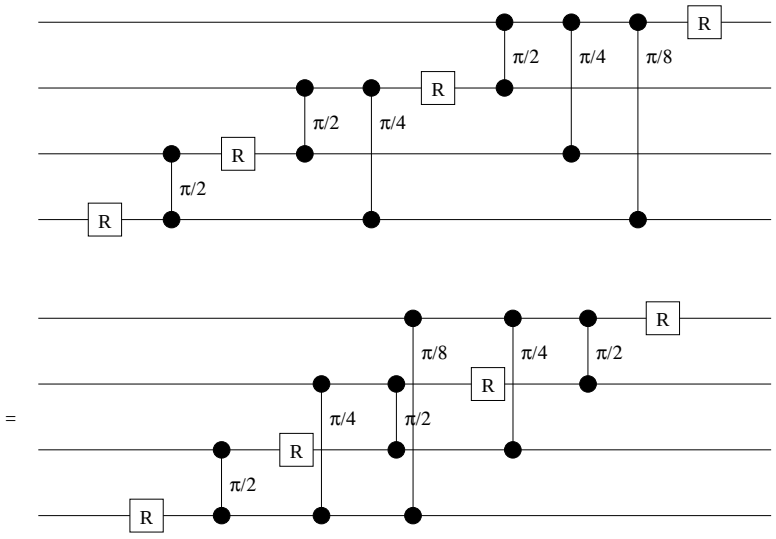


FIG. 19. *The standard circuit for the exact quantum Fourier transform on n qubits can be carried out in $2n-1$ layers. Can it be parallelized to less than linear depth?*

10. Conclusion. We conclude with some questions for further work.

Is **QNC**, or even **EQNC**, contained in **P**? Watrous's results on space-bounded quantum complexity classes [25] show that $\mathbf{QL} \subset \mathbf{NC}^2$, but no quantum analog of Borodin's equivalence [2] between depth and space is known. **BQNC** and **BPP**, or **EQNC** and **P**, might be incomparable.

Does parallelizing the encoding and decoding of error-correcting codes help reduce the error threshold for reliable quantum computation, at least in regimes where storage errors are more significant than gate errors?

Quantum context-free languages have been defined in [15], where it is shown that they are strictly more powerful than classical ones. Parsing classical context-free languages is in **NC**. Is quantum parsing, i.e., producing derivation trees with the appropriate amplitudes, in **QNC**?

Finally, can the reader show that the staircase circuit or the exact QFT cannot be parallelized?

Acknowledgments. M.N. would like to thank the Santa Fe Institute for their hospitality, and Spootie the Cat for her affections. C.M. would like to thank the organizers of the First International Conference on Unconventional Models of Computation in Auckland, New Zealand, as well as Charles Bennett, David Christie, David DiVincenzo, Sebastian Egner, Artur Ekert, Daniel Gottesman, Leonid Gurvits, Tom Knight, Seth Lloyd, Jonathan Machta, Mike Nielsen, Nick Pippenger, John Rickard, Ahto Truu, and John Watrous for helpful conversations. We also wish to express heartfelt sympathy with users of `xfig` everywhere.

REFERENCES

[1] A. BARENCO, C. BENNETT, R. CLEVE, D.P. DIVINCENZO, N. MARGOLUS, P. SHOR, T. SLEATOR, J.A. SMOLIN, AND H. WEINFURTER, *Elementary gates for quantum computation*, Phys. Rev. A(3), 52 (1995), pp. 3457–3467.

[2] A. Borodin, *On relating time and space to size and depth*, SIAM J. Comput., 6 (1977), pp. 733–744.
[3] S.L. Braunstein and J.A. Smolin, *Perfect Quantum Error Correction Coding in* 24 *Laser Pulses*, quant-ph/9604036.
[4] R. Cleve and D. Gottesman, *Efficient Computations of Encodings for Quantum Error Correction*, quant-ph/9607030.
[5] R. Cleve and J. Watrous, *Fast parallel circuits for the quantum Fourier transform*, in Proceedings of the 41st Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000.
[6] D. Coppersmith, *An Approximate Fourier Transform Useful in Quantum Factoring*, IBM research report RC 19642.
[7] D.P. DiVincenzo and P.W. Shor, *Fault-Tolerant Error Correction with Efficient Quantum Codes*, quant-ph/9605031.
[8] M. Frank, C. Vieri, M.J. Ammer, N. Love, N.H. Margolus, and T.F. Knight, *A scalable reversible computer in silicon*, in Proceedings of the First International Conference on Unconventional Models of Computation, Auckland, New Zealand, 1998.
[9] N. Gershenfeld and I. Chuang, *Bulk spin resonance quantum computation*, Science, 275 (1997), pp. 350–356.
[10] D. Gottesman, *The Heisenberg Representation of Quantum Computers*, quant-ph/9807006.
[11] F. Green, S. Homer, and C. Pollett, *On the complexity of quantum ACC*, in Proceedings of the 15th IEEE Conference on Computational Complexity, Florence, Italy, 2000.
[12] F. Green, S. Homer, C. Moore, and C. Pollett, *Counting, fanout, and the complexity of quantum ACC*, Quantum Information and Communication, 2 (2002), pp. 35–65.
[13] R.B. Griffiths and C.-S. Niu, *Semiclassical Fourier transform for quantum computation*, Phys. Rev. Lett., 76 (1996), pp. 3228–3231.
[14] R. Laflamme, C. Miquel, J.P. Paz, and W.H. Zurek, *Perfect quantum error correction code*, Phys. Rev. Lett., 77 (1996), pp. 198–201.
[15] C. Moore and J.P. Crutchfield, *Quantum automata and quantum grammars*, Theoret. Comput. Sci., 237 (2000), pp. 275–306.
[16] C. Moore and M. Nilsson, *Some Notes on Parallel Quantum Computation*, quant-ph/9804034, and *Parallel Quantum Computation and Quantum Codes*, quant-ph/9808027.
[17] C. Moore, *Quantum Circuits: Fanout, Parity, and Counting*, quant-ph/9903046.
[18] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[19] J. Preskill, *Reliable Quantum Computers*, quant-ph/9705031.
[20] E. Rains, *Quantum Shadow Enumerators*, quant-ph/9611001.
[21] M. Saks, *Randomization and derandomization in space-bounded computation*, in Proceedings of the 11th IEEE Conference on Computational Complexity, Philadelphia, PA, 1996, pp. 128–149.
[22] P.W. Shor, *Algorithms for quantum computation: Discrete logarithms and factoring*, in Proceedings of the 35th Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 124–134.
[23] P.W. Shor, *Fault-Tolerant Quantum Computation*, quant-ph/9605011.
[24] A. Steane, *Multiple particle interference and quantum error correction*, Proc. Roy. Soc. London Ser. A, 452 (1996), p. 2551.
[25] J. Watrous, *Relationships between quantum and classical space-bounded complexity classes*, in Proceedings of the 13th IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 210–227.
[26] W.K. Wooters and W.H. Zurek, *A single quantum cannot be cloned*, Nature, 299 (1982), pp. 802–803.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 816–837

ANALYSIS OF TIMING-BASED MUTUAL EXCLUSION WITH RANDOM TIMES*

ELI GAFNI† AND MICHAEL MITZENMACHER‡

Abstract. Various timing-based mutual exclusion algorithms have been proposed that guarantee mutual exclusion if certain timing assumptions hold. In this paper, we examine how these algorithms behave when the time for the basic operations is governed by probability distributions. In particular, we are concerned with how often such algorithms succeed in allowing a processor to obtain a critical region and how this success rate depends on the random variables involved. We explore this question in the case where operation times are governed by exponential and gamma distributions, using both theoretical analysis and simulations.

Key words. mutual exclusion, timed mutual exclusion, Markov chains, locks

AMS subject classifications. 68M14, 68W15, 68W20

PII. S0097539799364912

1. Introduction.

A good design methodology for developing distributed algorithms, as advocated by Liskov [10], is to assume the worst and hope for the best. In assuming the worst, one designs an algorithm which is safe regardless of the amount of time each operation takes. In hoping for the best, one designs the algorithm to optimize some utility function under certain timing assumptions.

A nice example of such a design is the mutual exclusion algorithm of Lynch and Shavit [12]. We describe the algorithm here at a high level; definitions of the relevant terms appear in section 2.1. The Lynch and Shavit algorithm for mutual exclusion is designed to cope with variations in timing of read and write operations. It combines previous mutual exclusion algorithms of Fischer [5] and Lamport [8] in a clever way in order to guarantee mutual exclusion and weak deadlock-freedom, as well as guarantee deadlock-freedom if certain timing constraints are met. Specifically, the algorithm is guaranteed to avoid deadlock if all steps of a process take time in a fixed range $[c_1, c_2]$. Given these timing constraints, specific pauses depending on the bounds c_1 and c_2 are added into the program for each process; these pauses ensure deadlock-freedom. Note that deadlock-freedom comes at a price, namely, the introduction of pauses that delay the completion of operations. In practice, detecting deadlock and breaking it are very costly in terms of time, and therefore a good design should ensure that deadlock never or rarely happens.

It is reasonable to assume that hard timing constraints will rarely or never be violated in the case of interprocess communication through a standard shared memory, such as when processes are running on machines in the same room. The gap between

*Received by the editors December 16, 1999; accepted for publication (in revised form) May 15, 2001; published electronically December 18, 2001. A preliminary version of this work appeared in *Proceedings of the* 18*th Annual Symposium on Principles of Distributed Computing*, Atlanta, GA, 1999, pp. 13–21.

http://www.siam.org/journals/sicomp/31-3/36491.html

†UCLA Computer Science Department, 3731 F Boelter Hall, Los Angeles, CA 90024-1596 (eli@cs.ucla.edu). Part of this author's work was done while visiting Compaq Systems Research Center. This author was supported by grant 4-592560-19914 from the UCLA Council on Research.

‡Maxwell Dworkin Laboratory 331, 33 Oxford Street, Harvard University, Computer Science Department, Cambridge, MA 02138 (michaelm@eecs.harvard.edu). Most of this author's work was done while employed at Compaq Systems Research Center. This author was supported in part by an Alfred P. Sloan Research Fellowship and NSF CAREER grant CCR-9983832.

the minimum and maximum memory reaction time is likely to be sufficiently small enough that pauses based on these timing constraints will generally yield only a small performance penalty. With the rise of fast networks and the Internet, however, there are alternative situations where processors may communicate through a much slower and more variable shared memory medium. For example, interprocess communication can be accomplished via servers reading and writing shared disk pages from a shared farm of disks accessible over a network. The mechanisms for using shared disks in this manner exist today and are described in several works on storage area networks [3, 4, 9, 14]. Indeed, storage area networks offer a shared memory that is cheap, reliable, and large; moreover, with regard to the design of distributed algorithms, the physical model of this architecture is close to the abstract model of shared memory. Operations on a disk-based shared memory might be slow and have large variance; moreover, its timing may not be well understood. Making hard timing assumptions that are guaranteed to hold may entail prohibitively long timeouts or self-delays for practice.

An alternative application that we envision involves multiple processes interacting via the Internet, such as in an auction on eBay. In such a scenario, the number of processes interacting may be extremely large. Also, while operations on shared memory may be instantaneous, users cannot expect response times on the order of shared memory systems, since Internet progagation delay will dominate.

Therefore, we are motivated to expand the analysis of the performance of mutual exclusion algorithms based on shared memory to systems that can potentially have long delays, so that the bounded timing model is not applicable. In many cases, even when timing bounds may prove problematic, knowledge of the distribution of operation times may be possible through systematic study. Consequently, we suggest introducing a probabilistic analysis of mutual exclusion algorithms under random delays.

Besides the above motivation, once we considered the idea of probabilistic analysis, it occurred to us that randomized algorithms for mutual exclusion may be more efficient than previous algorithms even in the context of fast shared memories. Instead of having algorithms introduce deterministic pauses designed for the worst case in order to guarantee mutual exclusion, using shorter pauses with random times may lead to better practical performance. The hope is that smaller random delays will avoid deadlock often enough that it will be more efficient to use small random delays and a mechanism for breaking deadlock than a slower deadlock-free algorithm. This approach may allow tradeoffs between correctness properties and efficiency.

A further motivation for introducing probabilistic models into this area is simply to gain more insight into the features of these algorithms. In particular, our analysis demonstrates that an appropriate pause (even one that lasts a random time) can dramatically change an algorithm's behavior.

We further note that the probabilistic framework we introduce is reminiscent of similar work on contention resolution in multiaccess channels. The contention resolution framework has proven highly successful. (See the notes in [6] or references from [7] or [13].) We suspect that this direction may therefore prove worthwhile in the context of mutual exclusion or other distributed algorithms as well. For example, since the publication of the original version of this paper, a similar probabilistic framework was used by Aspnes to study a deterministic consensus algorithm against an adversary who cannot control random timing noise introduced by the system [2].

In this paper, we focus on the case where operation times have the exponential

distribution. This distribution has properties which prove handy for analysis. Moreover, although the assumption of exponential distributions is not correct in practice, algorithms that behave well under the exponential distribution are generally assumed (whether correctly or not) to behave well under "reasonable" distributions. Thus they make an appropriate starting point for this analysis. We also examine the case where operation times have a gamma distribution, both to offer more insight and to avoid the problem of drawing conclusions specific to the exponential distribution.

We refer to the basic unit of much of our analysis as a *lock*. Loosely speaking, for our purposes a lock is a shared variable that can be inspected (or read, to see if it is clear), written (to attempt to take control), and read (to see if control has been obtained). A processor successfully passes through a lock if it finds it clear on inspection, writes its processor ID to it, and reads back its processor ID. Note that a processor may pause, or self-delay, between any of these steps. A lock is a basic unit in Fischer's mutual exclusion algorithm [5], which we describe in section 2.1. Studying locks provides us with the means and insight to study variations on the algorithm of Lynch and Shavit [12].

We are interested in answers to questions such as the following:

1. How often do locks succeed, and how does this depend on the underlying distributions?
2. Are we better off with one lock with a long pause or two consecutive locks with smaller pauses?
3. How should lock constructions be combined in this setting?

In this paper, we focus on the analysis of the basic lock construction and explore the behavior of these locks and some of our questions with simulations. As a by-product of our work, we explore the behavior of several simple but interesting Markov chains. We believe that further, more detailed analysis of these Markov chains would be interesting, not only because of their connection to timed mutual exclusion algorithms, but also in and of themselves.

Because we focus on the simple lock mechanism, the analysis in this version of the paper is essentially self-contained. However, we encourage the interested reader to peruse the work by Lynch and Shavit on timing-based mutual exclusion [12] for more details on Lamport's algorithm, Fischer's algorithm, and their combination, in order to put this work in context.

2. Background.

2.1. Definitions. For completeness, we describe the basic definitions associated with the mutual exclusion problem. Here we generally follow the definitions and notation of [12]. (See also [11] for extensive references and related work.)

A mutual exclusion algorithm arbitrates among n sequential threads of control, or processes. Processes communicate by reading and writing in some form of shared memory. Read and write operations on this memory are assumed to execute instantaneously; that is, they happen atomically on memory locations. The process itself, however, might not obtain the result of the read or write until some future time, depending on the architecture of the system. For our purposes, the program associated with each process has a form as given in Figure 1. In particular, a process has an associated *critical region*. A system is said to *satisfy mutual exclusion* if in any reachable system state at most one user is in its critical region. Note that the *trying region* and the *exit region* are used to coordinate entry to and exit from the critical region; the *remainder region* is where all other work is done.

```
Basic process
p: current process index

repeat forever:
  remainder region
  trying region
  critical region
  exit region
end repeat;
```

FIG. 1. *Basic process program.*

Two other properties are useful to consider. A system is said to be *weakly deadlock-free* if when any single process's trying region is concurrent only with the remainder regions of other processes, then its trying region terminates, and similarly if when any single process's exit region is concurrent only with the remainder region of other processes, then its exit region terminates. This property corresponds to the requirement that if a process runs alone, it accesses the critical region. The stronger property of being *deadlock-free*, which corresponds to the requirement that the system progress, requires that

- if some process is in the trying region and no process is in the critical region, then subsequently some process enters the critical region; and
- if some process is in the exit region, then subsequently some process enters the remainder region.

The algorithm of Lynch and Shavit relies on Lamport's fast mutual exclusion algorithm [8] to guarantee that mutual exclusion is never violated. (See Figure 2.) It also relies on Fischer's timed mutual exclusion algorithm [5] to provide Lamport's algorithm the environment it requires for deadlock-freedom, namely, a single contender. (See Figure 3.) We discuss the combined algorithms in section 4. Proofs of these properties appear in [12].

Note the appearance of a pause in Fischer's timed mutual exclusion algorithm. The point of the pause is as follows: suppose each *step* of a process, corresponding to a line of code, takes time bounded between $[c_1, c_2]$ for some positive finite values c_1 and c_2. Then if the pause time corresponds to at least $\lceil c_2/c_1 \rceil$ steps (using for example no-op operations), so that the pause takes time at least c_2, then Fischer's algorithm guarantees both mutual exclusion and deadlock-freedom.

2.2. Properties of the exponential distribution. Recall that a random variable that is exponentially distributed with mean μ is defined by its probability density function, $f(x) = (1/\mu)\mathrm{e}^{-x/\mu}$. The exponential distribution proves convenient for theoretical study because of its special properties. We briefly note these properties here and make use of them without further reference throughout this paper.

- *Memoryless property.* Suppose that the time until an event is determined by an exponential random variable with mean μ. Given that the event has not yet happened, the remaining time until the event happens is still an exponential random variable with mean μ.
- *Minimum property.* Suppose that the times until each of k events are determined by independent exponential random variables with mean μ. Then the time until the first of these events occurs is exponential with mean $\frac{\mu}{k}$.

Lamport
x, y: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
$x := p$;
if $y \neq 0$ then goto L;
$y := 1$;
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$y := 0$;
% Exiting ME-lock

FIG. 2. *Lamport style mutual exclusion.*

Fischer
x: shared register, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$x := 0$;
% Exiting ME-lock

FIG. 3. *Fischer's timed mutual exclusion algorithm.*

- *Fairness property.* Suppose that the times until events A and B are determined by independent exponential random variables with means μ_1 and μ_2, respectively. Then event A occurs first with probability $\frac{\mu_2}{\mu_1+\mu_2}$.

2.3. How many pass through? We begin by considering a basic unit for mutual exclusion algorithms, namely, a *lock*. A lock access protocol consists of an *inspect* phase (which is an initial read of the shared variable that comprises the lock), a *write* phase, and a final *read* phase. A processor inspects the lock to see if it is clear; it attempts to write its processor ID to the lock; and then it passes through the lock successfully if it reads its own ID. A processor that successfully passes through the lock eventually clears the shared variable so that others may pass through; until this occurs, the processor is said to own the lock. Mutual exclusion is guaranteed as long as no two processors believe they own the lock at the same time. Recall that a lock is the mechanism behind Fischer's algorithm, as seen in Figure 3. Also, Fischer's mutual exclusion algorithm also allows for pauses. We begin our analyses without considering the effect of a pause; however, we return to consider the pause later in the paper.

We will often compare the behavior of a lock with a *double lock*, by which we mean two successive back-to-back locks, each with its own shared variable. When a processor passes through the first lock of a double lock, it then begins the inspection phase for the second lock of the double lock. A processor is said to own a double lock only after it has passed through the second lock, and mutual exclusion is guaranteed as long as no two processors believe they own the lock at the same time. A natural question we consider here is whether using two short locks in a double lock might be better than using a long single lock.

We denote the three phases by I, W, and R, respectively. In this section, unless otherwise stated, we assume that the times for each of these actions are exponentially distributed, with means i, w, and r respectively, where the values of i, w, and r are fixed constants (independent of the number of processors in the system). For convenience, we scale so that $w = 1$ unless otherwise noted.

We emphasize that an operation is meant to take place *atomically* (that is instantaneously, from the point of view of the processes) at the end of the time interval corresponding to the operation. That is, the fact that operations take time to complete is not to suggest that they do not take place atomically, but only that there is a delay between when an operation is initiated by a processor and when it completes. One way to view this model is that operations initiated by a processor are scheduled in some way, say, on a shared disk system. The scheduling causes a random delay between when an operation is initiated and when it is completed. A processor sees the results of an operation as soon as it is completed.

We begin by presenting some simple arguments regarding how many processors complete successive stages of a lock in the face of contention. These arguments do not answer our main question, which is how often just one processor successfully obtains a lock in the face of contention. They do, however, introduce the flavor of our arguments and provide some initial insight.

THEOREM 1. *Consider a situation where n processors begin inspecting a free lock at the same time. Then, with probability bounded below by some constant, at least $\Omega(\sqrt{n/i})$ processors complete the inspection stage before the first write completes.*

Remark. The assumption that the processors begin at the same time is for convenience; since all times are exponentially distributed, as long as a write has not occurred, we may take any instant when n processors are in the I stage as the beginning.

Proof. We derive a recursive function p_j describing the probability that at least j processors successfully inspect the lock before the first write. Suppose that jth inspection has just completed, and no writes have yet occurred. Then the time until the next inspection completes is exponentially distributed with mean $i/(n-j)$, as there are $n-j$ processors remaining. The time until the first write completes is exponentially distributed with mean $1/j$, as there are j processors attempting a write. Hence, the probability that another inspection completes before the first write is $\frac{n-j}{ij+n-j}$. Recursively, then, we have $p_1 = 1$ and $p_{j+1} = p_j \frac{n-j}{ij+n-j}$.

Let $z = \sqrt{n/i}$. Then

$$\begin{aligned} p_{z+1} &= \prod_{1 \le j \le z} \frac{n-j}{ij+n-j} \\ &= \prod_{1 \le j \le z} \left(1 - \frac{ij}{ij+n-j}\right) \end{aligned}$$

$$\geq \prod_{1 \leq j \leq z} \left(1 - \frac{ij}{(1-\epsilon)n}\right)$$

for an ϵ that goes to 0 as n gets large. Hence,

$$p_{z+1} \geq \prod_{1 \leq j \leq z} \left(1 - \frac{ij}{(1-\epsilon)n}\right) \geq \left(1 - \frac{1}{(1-\epsilon)z}\right)^z,$$

which is arbitrarily close to $\mathrm{e}^{-1/(1-\epsilon)}$ for sufficiently large n. This demonstrates that with at least some constant probability, at least $\Omega(\sqrt{n/i})$ processors complete the I stage. □

It is easy to extend the proof of Theorem 1 to show that the expected number of processors that complete their I stage before the first write is actually $\Theta(\sqrt{n/i})$.

THEOREM 2. *Consider the setting of Theorem* 1. *The expected number of processors that complete the inspection stage before the first write is* $\Theta(\sqrt{n/i})$.

Proof. The lower bound follows from Theorem 1. For the upper bound, note that the expected number of processors to complete the inspection stage before the first write is $\sum_{m \geq 1} p_m$. Let $z = \sqrt{n/i}$; then for any integer $k \geq 1$, for y such that $zk < y \leq z(k+1)$,

$$p_y = \prod_{1 \leq j \leq y-1} \frac{n-j}{ij+n-j} \leq \prod_{z \leq j \leq y-1} \frac{n}{ij+n} \leq \left(1 + \frac{1}{z}\right)^{-(k-1)z} < 2^{-k+1}.$$

It follows that $\sum_{m \geq 1} p_m < 3\sqrt{n/i}$. □

In fact, asymptotically exact formulae can be found with some work. We demonstrate this for the case $i = 1$, which yields an interesting result, although the same technique applies for other cases. When $i = 1$, we have $p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n}$, and the expected number of processors that complete the I stage before the first write is $E_I = \sum_{k=1}^{n} p_k$. Consider plotting the points $((k-1)/n, np_k)$ in the first quadrant of the Euclidean plane for $k = 1, \ldots, n$. The area under the successive axes-parallel rectangles defined by these points equals the desired expectation E_I. Moreover, the area of these rectangles approximates the area under a curve passing through these points. Defining a curve that passes through these points is difficult, but we can find a curve that nearly passes through these points quite easily. Consider moving from $(x, y) = ((k-1)/n, np_k)$ to $(k/n, np_{k+1})$. Note that as we move $\Delta x = 1/n$ on the x-axis, the corresponding y-value drops by $\Delta y = -(x + \Delta x)y$. Hence, our points are well approximated by the curve defined by the differential equation $dy/dx = -nxy$ and the boundary condition $y(0) = n$. This curve is just $y = n\mathrm{e}^{-nx^2/2}$. The area under the curve is

$$\int_0^1 n\mathrm{e}^{-nx^2/2} dx = \sqrt{\frac{\pi n}{2}} + O(1).$$

Hence, if n processors begin an I stage, then (up to lower order terms) on average $\sqrt{\frac{\pi n}{2}}$ processors complete their inspection before the first write occurs.

This argument for $i = 1$ can be formalized by noting that

$$p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n} \leq \prod_{1 \leq j \leq k-1} \mathrm{e}^{-j/n} = \mathrm{e}^{-k(k-1)/2n} \leq \mathrm{e}^{-(k-1)^2/2n}.$$

It easily follows that $\sum_{k=1}^{n} p_k$ is bounded above by

$$1 + \int_0^n \mathrm{e}^{-x^2/2n} dx = \int_0^1 n\mathrm{e}^{-nx^2/2} dx + O(1).$$

Similarly, using $1 - x \geq \mathrm{e}^{-x-x^2}$ for $0 \leq x \leq 1/2$, we have for $k \leq n/2$

$$p_k = \prod_{1 \leq j \leq k-1} \frac{n-j}{n} \geq \prod_{1 \leq j \leq k-1} \mathrm{e}^{-j/n - j^2/n^2} \geq \mathrm{e}^{-k(k-1)/2n - k(k-1)(2k-1)/6n^2}$$
$$\geq \mathrm{e}^{-k^2/2n - k^3/3n^2}.$$

It follows that $\sum_{k=1}^{n} p_k$ is bounded below by $\int_0^{1/2} n\mathrm{e}^{-nx^2/2 - nx^3/3} dx + O(1)$, and it can be checked that this is equal to $\int_0^1 n\mathrm{e}^{-nx^2/2} dx + O(1)$. (For example, split the integral into two parts, the first covering the range $[0, n^{-5/12}]$ and the second $[n^{-5/12}, 1/2]$. The cubic term is lower order in the exponent in the first range and can be absorbed in the $O(1)$. Similarly, the exponential term in the second range is small enough to be absorbed in the $O(1)$, which also explains why the difference between integrating to $1/2$ and integrating to 1 can be dismissed.)

THEOREM 3. *Consider a situation where n processors begin to write to a lock at the same time. Then on average $\Theta(\ln(rn)/r)$ read their own value, and in fact $\Theta(\ln(rn)/r)$ read their own value with probability $1 - o(1)$.*

Proof. The time between the jth and $(j+1)$st write is exponentially distributed with mean $1/(n-j)$. Hence, the probability that the processor that makes the jth write reads its own value is

$$\frac{\frac{1}{n-j}}{r + \frac{1}{n-j}} = \frac{1}{r(n-j)+1}.$$

The expected number of processors that read their own value is therefore

$$\sum_{j=1}^{n} \frac{1}{r(n-j)+1}.$$

When $r = 1$, this is simply $\sum_{j=1}^{n} 1/j = H(n) \approx \ln n$. Otherwise, bounding the sum by appropriate integrals we have

$$\int_{x=1}^{n} \frac{1}{xr+1} dx \leq \sum_{j=1}^{n} \frac{1}{r(n-j)+1} \leq 1 + \int_{x=0}^{n} \frac{1}{xr+1} dx,$$

and hence

$$\frac{\ln(rn+1)}{r} - \frac{\ln(r+1)}{r} \leq \sum_{j=1}^{n} \frac{1}{r(n-j)+1} \leq 1 + \frac{\ln(rn+1)}{r}.$$

The argument can be easily extended to show that the number of processors that read their own value is $\Theta(\ln n)$ with high probability. Let X_j be the event that the processor that makes the jth write reads its own value. Under the assumption of exponentially distributed read and write times, the X_j are independent. Letting

$X = \sum_{j=1}^{n} X_j$, we may use the standard Chernoff bound (see, for example, Corollary A.14 of [1])

$$\Pr(|X - E[X]| \geq \epsilon E[X]) \leq 2\mathrm{e}^{-\epsilon^2 E[X]/3}.$$

Hence, for any fixed r the probability of X deviating from the mean by more than $\epsilon E[X]$ falls inverse polynomially in n, proving the theorem. □

From Theorems 1 and 3 we immediately obtain as a corollary that two locks are significantly better than one, in terms of the number of processors that can get through (in the case of no pauses). Specifically, for a single lock with all times having the same mean, $\Theta(\sqrt{n})$ processors inspect the free lock before a write occurs with constant probability. Of these processors, with high probability $\Theta(\ln \sqrt{n}) = \Theta(\ln n)$ then read their own values and hence pass through the lock. For a double lock, from Theorem 3, with high probability $\mathrm{O}(\ln n)$ get through the first lock, and hence with high probability at most $\mathrm{O}(\ln \ln n)$ pass through the second. Note that changing the mean times for the I, W, or R operations (while keeping them constant) changes only these expressions by constant factors, and hence this remains true even if the average time to pass through the lock is the same in both scenarios. Hence, in the face of sufficiently large contention, double locks are much better with regard to the number of processors that pass through (on average, with no pauses).

2.4. How often does one pass through? Showing that on average fewer processors pass through a double lock than a long single lock does not really answer our question of which is better. The proper measure of performance is how often a lock successfully allows only one processor through. We now focus on this variable. First, we show that for a single lock with exponentially distributed read and write times (and no pause), a single lock can perform quite poorly under high contention.

THEOREM 4. *Consider a single lock with n processors beginning a write at the same time. The probability that just a single processor reads its own value is $O(\sqrt[r]{1/rn})$.*

Proof. We begin with the case $r = 1$. Recall from Theorem 3 that the jth processor to write reads its own value with probability $1/(r(n-j)+1)$ and that all such events can be treated as independent. Clearly, the last processor to write will read its own value. The probability that it is the only one to do so is

$$\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\cdots\left(1 - \frac{1}{n}\right) = \frac{1}{2}\frac{2}{3}\cdots\frac{n-1}{n} = \frac{1}{n}.$$

Thus, when $r = 1$, the probability that only one processor believes it obtains the lock is $1/n$. For a general r, this probability is

$$\begin{aligned}\prod_{j=1}^{n-1}\left(1 - \frac{1}{r(n-j)+1}\right) &\leq \prod_{i=1}^{n-1}\mathrm{e}^{-1/(r(n-i)+1)} \\ &= \mathrm{e}^{-\sum_{i=1}^{n-1} 1/(r(n-i)+1)} \\ &\leq \mathrm{e}^{-1-(\ln(rn)+1)/r},\end{aligned}$$

and the last term is $O(\sqrt[r]{1/rn})$. □

The result of Theorem 4 demonstrates how the probability of success increases with r and decreases with n. Although increasing r substantially increases the probability of just one processor successfully obtaining the lock, as n grows large, for any fixed r this probability falls to 0.

We now consider the probability of exactly one processor taking control of a double lock. Under a reasonable assumption, we find that in this case, the probability that a single processor obtains the lock is bounded below by a constant, *regardless of how n grows.* This result is somewhat surprising, given the previous result for a single lock.

In this setting, we adopt the following assumption: once a processor passes through the second lock, it will hold that lock for a reasonably long amount of time. Hence, if one processor writes to the second lock before any others read it, we assume that this processor does not clear the second lock until well after all others read that it has possession. This assumption simplifies the problem, as now we need to consider only the problem of whether one processor writes to the second lock before any others read it. It is also reasonable, since a lock is held long enough so that the critical region can be executed.

THEOREM 5. *Let n processors begin a write for a first lock of a double lock at the same time. Then with probability bounded below by some constant, one processor writes to obtain the second lock before any other processors successfully pass through the first lock.*

Proof. The intuition behind the theorem is relatively simple. With some constant probability, one lucky processor passes through the first lock quickly. It then writes to obtain the second lock before any other lucky processors can pass through the first lock. We now formalize this intuition. We first consider the case where $i = r = w = 1$ for convenience. Also, we assume all relevant quantities are integers and avoid floor and ceiling notation for convenience as well.

The jth processor to write passes through the first lock with probability $\frac{1}{n-j+1}$. Hence the probability that none of the first $n/2$ processors passes through the first lock is

$$\prod_{j=1}^{n/2}\left(1-\frac{1}{n-j+1}\right)=\frac{n-1}{n}\frac{n-2}{n-1}\cdots\frac{n/2}{n/2+1}=\frac{1}{2}.$$

Similarly, the probability that exactly one of the first $n/2$ processors passes through the first lock is

$$\sum_{j=1}^{n/2}\left[\frac{\frac{1}{n-j+1}}{1-\frac{1}{n-j+1}}\prod_{k=1}^{n/2}\left(1-\frac{1}{n-k+1}\right)\right]=\frac{1}{2}\sum_{j=1}^{n/2}\frac{1}{n-j}\geq\frac{\ln 2}{2}-o(1).$$

Now suppose exactly one processor from the first $n/2$ passes through the first lock; let it be the jth to write. We now lower bound the probability this processor writes to obtain the second lock before any other processor passes through the first lock. To do so, this processor must complete both an I and W operation. Since all operation times are exponential, with constant probability both these operations complete before the $(7n/8)$th processor completes its write to the first lock. This is clear since with probability 1/2, the I operation occurs before 1/2 of the remaining $n-j$ writes to the first lock. Assuming this happens, with probability 1/2 again, the second W operation completes before 1/2 the remaining writes to the first lock. Hence, with the probability 1/4, the jth processor finishes the I and W operation for the second lock by the time processor $j+(n-j)/2+(1/2)(n-(j+(n-j)/2))$ writes to the first lock. Since $j \leq n/2$, we have that with constant probability the jth processor finishes the I and W operation for the second lock by the time the $(7n/8)$th

processor writes to the first lock. Now, however, by the same argument as previously, the probability that no processors from the $(n/2)$nd to the $(7n/8)$th finish their first write and pass through to the second lock is

$$\prod_{i=n/2+1}^{7n/8}\left(1-\frac{1}{n-i+1}\right)=\frac{1}{4}.$$

Because of the memorylessness of the exponential distribution, all of these events can be treated as independent, and hence with probability bounded below by some constant a single processor successfully writes to the second lock as in the statement of the theorem.

When r and i are fixed constants other than 1, the same argument suffices; various constants in the argument must be changed to reflect the change in r and i. We sketch the required changes. The jth processor passes through the first lock with probability $\frac{1}{1+r(n-j)}$. Hence, the probability that none of the first $n/2$ processors passes through the first lock is

$$\prod_{j=1}^{n/2}\left(1-\frac{1}{r(n-j)+1}\right).$$

We may bound this by noting $1-x\geq \mathrm{e}^{-x-x^2}$ for $0\leq x\leq 1/2$. Hence,

$$\begin{aligned}\prod_{j=1}^{n/2}\left(1-\frac{1}{r(n-j)+1}\right)&\leq\prod_{j=1}^{n/2}\mathrm{e}^{-1/(r(n-j)+1)-1/(r(n-j)+1)^2}\\&=\mathrm{e}^{\sum_{j=1}^{n/2}-1/r(n-j)}(1-o(1))\\&=\mathrm{e}^{-(H(n-1)-H(n/2))/r}(1-o(1))\\&=2^{-1/r}(1-o(1)).\end{aligned}$$

Similarly, the probability that exactly one such processor passes through the first lock is

$$\begin{aligned}\sum_{j=1}^{n/2}\left[\frac{\frac{1}{r(n-j)+1}}{1-\frac{1}{r(n-j)+1}}\prod_{k=1}^{n/2}\left(1-\frac{1}{r(n-k)+1}\right)\right]&=2^{-1/r}\sum_{j=1}^{n/2}\frac{1}{r(n-j)}(1-o(1))\\&=\frac{2^{-1/r}\ln 2}{r}(1-o(1)).\end{aligned}$$

Now suppose exactly one processor passes through the first lock. For this processor to write to obtain the second lock before any other processor passes through the first lock, it must complete both an I and W operation. Since all operation times are exponential, with constant probability both these operations complete before the (αn)th processor completes its write for some constant α depending on i. However, the probability that no processors from the $(n/2)$nd to the (αn)th finish their first write and pass through to the second lock is

$$\prod_{j=n/2+1}^{\alpha n}\left(1-\frac{1}{r(n-j)+1}\right),$$

which can be bounded above and below by some constant independent of n. Hence, again with probability bounded below by some constant, a single processor successfully writes to the second lock as in the statement of the theorem. □

The rather loose analysis of Theorem 5 greatly underestimates the probability that a single processor successfully writes to the second lock before all others. The true probabilities are best determined by simulations, and hence we return to this question in section 5.

We also note that another way to gain better insight into the exact probability that a single processor successfully passes through the double lock is to consider the underlying Markov chain. For instance, this chain can easily be represented as a six-dimensional Markov chain, where each dimension tracks the number of processors in each state. Examining this Markov chain could lead to provable bounds on various probabilities associated with the lock's behavior. Of course, a complete analysis of this complex chain appears rather difficult. We therefore feel that our intuitive proof, combined with simulation results, is a natural approach to the problem.

Given that two locks have a different behavior than one, one might naturally ask whether three (or more) locks have a different behavior than two. Using induction and the above results one can show that using $2k$ consecutive locks, the probability of more than one processor successively passing through is at most γ^k for some constant γ. Could the behavior be even better than exponentially decreasing? We show that the answer to this question is negative by considering the limiting case where just two processors start together at the first lock.

THEOREM 6. *Consider two processors starting at a sequence of k locks. Then the probability that both processors pass through the final lock is at least β^k for some constant β depending on i, w, and r.*

Proof. We first show that the probability that two processors "follow each other" through the lock is a constant. That is, consider the following sequence of events:

1. Both processors inspect the lock before either writes.
2. The first processor to complete a write to the lock reads back its value before the other processor completes its write to the lock.
3. The second processor to complete a write to the lock reads back its value before the other processor inspects the subsequent lock.

If these events occur, because of the memorylessness property of the exponential distribution, the two processors are then in a similar state as though they had both just begun competing for the subsequent lock. It is clear that the intersection of these events hold with constant probability. In fact, by the fairness property and the memorylessness property, we can calculate that all of these events occur with probability β, where

$$\beta = \frac{w}{i+w}\frac{w}{r+w}\frac{i}{i+w}\frac{i}{i+r}.$$

By induction, the probability of both processors passing together through k consecutive locks is at least β^k. □

Thus, for any number of successive locks in this setting, the best one can hope for is a failure probability that decreases exponentially in the number of locks.

3. The gamma distribution. While the previous section, in which we considered exponential random variables, showed that a double lock is better than a single lock, the results must of course be taken in context. Since we know that in the case where all times are deterministic (and, for example, all operations require the same

time) that a single lock is sufficient, it becomes interesting to consider how strongly this behavior depends on the underlying distribution. We offer some insight into this problem by considering the *gamma distribution*. Recall that a gamma distribution is the sum of a number of independent exponential random variables (of the same mean). For example, a gamma(2) distributed random variable with mean 1 is the sum of two exponential random variables, each with mean 1/2.

We show that for a gamma(2) distribution, the probability that only a single processor obtains a single lock is bounded below by a constant independent of n, the number of processors contending for the lock. Hence, in this case, a single lock behaves more like a double lock under the exponential distribution.

The intuition behind this performance is as follows. Consider the case where n processors are initiating the write stage for the lock at the same time. We may think of the write phase for a processor as consisting of two subphases, each corresponding to an exponentially distributed amount of time. Let us say that a processor is *half-done* with the write stage if it has completed its first subphase, *done* or *completed* if its write is fully complete, and *unstarted* if it is not yet even half-done. Before the first processor to complete a write finishes the write, several processors will be half-done. The number of processors half-done with their write are very likely to prevent this first processor from reading its value, for it is very likely that one of these half-done processors will complete its write before this processor can finish its read. This situation, where half-done writes overwrite completed writes before the corresponding read finishes, is likely to occur until few processors remain to complete their writes. When there are few processors remaining, it is possible for a read to complete before the processor value is overwritten, but this happens only with constant probability.

We present the above argument more formally in the theorem below for the case where reads and writes execute with the same average time. For convenience, we take this mean to be 2.

THEOREM 7. *Consider n processors beginning the write for a single lock, where the times for writes and reads have independent gamma(2) distributions with mean 2. Then a single processor reads its own value with probability bounded below by some constant.*

Proof. We assume that n is sufficiently large throughout. We wish to show that only the last processor to write its value reads its own value with constant probability. The proof is divided into three parts, corresponding to the beginning, the middle, and the end of the process.

For the beginning, we wish to show that with constant probability, by the time the first write completes, with constant probability there are at least $6\sqrt{n}$ processors that are half-done. This will ensure that sufficiently many half-done processors are around to block the completion of any write for all but the end of the process. Consider the time until the first write completes. Let p_j be the probability that at least j processors are at least half-done by this point. By the same argument as Theorem 1, $p_1 = 1$ and $p_{j+1} = p_j \frac{n-j}{n}$. It is straightforward to use this recurrence in a manner similar to Theorem 1 to show that at least $6\sqrt{n}$ are half-done when the first write completes with constant probability. (Note that, if we wished to bound this probability, we might do better to consider explicitly the behavior of the process until the first few writes complete; however, for our purposes the above is sufficient.)

For the middle, we show that with constant probability, conditioned on the fact that at least $6\sqrt{n}$ half-done processors exist at the time the first write completes, there are always many half-done processors until the very end of the process. Explicitly,

we claim that with constant probability there are always at least $2\sqrt{n}$ processors half-done with their writes as long as there are at least $10\sqrt{n}$ unstarted processors. This is easily seen by making a stochastic comparison with the number of half-done processors and a simple random walk. When there are u unstarted processors and h half-done processors, the probability that h increases (and u decreases) is $\frac{u}{h+u}$, and the probability that h decreases (and u stays the same) is $\frac{h}{h+u}$. Moreover, because all distributions are exponential, each step is independent. In particular, when $u \geq 10\sqrt{n}$ and $h < 10\sqrt{n}$, the number of half-done processors h is biased upwards.

Now consider an unbiased random walk that starts at $6\sqrt{n}$ with boundaries at $2\sqrt{n}$ and $10\sqrt{n}$ that runs for $2n$ steps. We claim that it is more likely that h is at least $2\sqrt{n}$ until there are $10\sqrt{n}$ unstarted processors than that this unbiased random walk reaches the boundary $2\sqrt{n}$. This follows from a standard stochastic domination argument; the value h also begins at $6\sqrt{n}$, it changes less than $2n$ times, and it is always more likely to increase than the unbiased random walk. Standard results in probability theory now yield that the random walk (and hence h) stays above $2\sqrt{n}$ with constant probability.

This is most easily seen by noting that for the walk to reach $2\sqrt{n}$, it must fall $2\sqrt{n}$ in either the first n or the last n steps. Let $X_i = 1$ if a walk of n steps goes up on the ith step, and let $X_i = -1$ if it goes down on the ith step. Then from Theorem A.1 of [1], which is derived in a manner similar to Chernoff bounds, the probability that $\sum_{i=1}^{n} X_i \leq 2\sqrt{n}$ is at most e^{-2}. By a union bound, the probability that the walk falls $2\sqrt{n}$ in either the first n or the last n steps is at most $2\mathrm{e}^{-2}$.

Now, conditioned on all of the above, up to the point where there are $10\sqrt{n}$ unstarted processors, with constant probability no processor will read its own value. For to do so, any such processor must complete two read phases before any of the half-done writes complete. In each case the probability of doing so is $(\frac{1}{2\sqrt{n}})^2 = \frac{1}{4n}$, and hence by the union bound with probability at least $\frac{3}{4}$ no processor to this point reads its own value.

We clarify that this statement follows using conditional probabilities, not a union bound. That is, we define the following: let A be the event that $6\sqrt{n}$ processes are half-done when the first write completes. Let B be the event that at least $2\sqrt{n}$ processes are half-done as long as there are at least $10\sqrt{n}$ unstarted processes. Let C be the event that, up to the point where there are $10\sqrt{n}$ unstarted processes, no processor reads its own value. Then

$$\Pr(C) \geq \Pr(C|B) \cdot \Pr(B|A) \cdot \Pr(A),$$

and we have shown that all of the above on the right-hand side are constants.

We now need to consider the end of the process. To see what happens toward the end of the process, consider what would happen if the system began with all processors half-done with their writes. The jth processor to complete its write would then successfully read its own value if it completed two read phases before any of the half-done writes completed, which occurs with probability $(\frac{1}{n-j+1})^2$. Hence, the probability that any processor other than the last to write would read its own value would be at most $\sum_{j=1}^{n-1}(\frac{1}{n-j+1})^2 < \frac{6}{\pi^2}$. (We elaborate on this in Theorem 8.)

In the actual process, we have already seen that all behaves well up to the point when there are $10\sqrt{n}$ unstarted processors. After this point, we claim the system behaves similarly to one where all remaining processors begin half-done with their writes. Specifically, we show that at the last point in time when there are k processors left unstarted, there are at least $k \log n/2$ processors left with probability $1 - o(1)$ for

all k from 1 to $10\sqrt{n}$. This implies that at the end of the process, we always have that most processors are half-done, which will suffice.

Let us consider the specific case where $k = 1$. The probability that the $(n-j)$th processor to become half-done has not yet completed at the first time when there is one processor left unstarted is just $1/(j+1)$. Hence, the expected number of half-done processors at the point where there is just one unstarted processor is approximately $H(n) \approx \log n$. Moreover, the events (that the jth processor to become half-done has not yet completed) are independent, so we may apply Chernoff bounds. Hence, we find the probability that there are not at least $\log n/2$ half-done processors is at most $1/n^{1/16}$.

We may attack larger k similarly. The probability that the $(n-j)$th processor to become half-done has not yet completed while there are k processors left unstarted is just $k/(j+1)$. Hence, the expected number of half-done processors at the point where there are k unstarted processors is approximately $k(H(n) - H(k)) \approx k \log(n/k)$. For $k \leq \log n$, a Chernoff bounds yields that the probability that there are not at least $k \log(n/k)/2$ half-done processors is at most $1/n^{1/16}$. For $\log n \leq k \leq 10\sqrt{n}$, Chernoff bounds yield that the probability that there are not at least $k \log(n/k)/2$ half-done processors is at most $n^{-\log n/8}$. Using a union bound, we find that, for all k from 1 to $10\sqrt{n}$, at the last point in time when there are k process left unstarted, there are at least $k \log(n/k)/2$ half-done processors left with probability $1 - o(1)$.

Let us temporarily assume that this is the case. Let $u(j)$ be the number of unstarted processors when the jth processor to write completes its write. Then the probability that the jth processor to write reads its own value is at most

$$\left(\frac{1}{n-j-u(j)+1}\right)^2.$$

We are interested only in the situation when $u(j) \leq 10\sqrt{n}$. Hence, the probability that some processor reads its own value when $u(j) \leq 10\sqrt{n}$ is bounded above by

$$\sum_{j=1}^{n-1} \left(\frac{1}{n-j-\min(10\sqrt{n}, u(j))+1}\right)^2.$$

Note that summing to $n-1$ is clearly overcounting. Also, with high probability, for $j = n - \log n/2$ to $n-1$ the value of $u(j)$ is 0. It follows that with high probability the above sum is just

$$\sum_{j=1}^{n-1} \left(\frac{1}{n-j+1}\right)^2 + o(1) < \frac{6}{\pi^2} + o(1),$$

where the $o(1)$ term corrects for the $\min(10\sqrt{n}, u(j))$ term.

To show that this suffices, let D be the event that, from the point where there are $10\sqrt{n}$ unstarted processes, no processor except the last reads its own value. Let E be the event that for all k from 1 to $10\sqrt{n}$, when there are k processors left unstarted, there are at least $k \log(n/k)/2$ processors left. Finally, let S be the successful event that no processor except the last reads its own value.

Then

$$\Pr(S) = \Pr(D \wedge C) \geq \Pr(D \wedge C \wedge E) = \Pr(D|C \wedge E) \cdot \Pr(C \wedge E).$$

The argument regarding the end of the process shows that $\Pr(D|C \wedge E)$ is bounded below by a constant. Also, $\Pr(C \wedge E)$ is bounded below by a constant, since $\Pr(C)$ is and $\Pr(E)$ is $1 - o(1)$. Hence, $\Pr(S)$ is bounded below by a constant and the theorem is proven.

To summarize, we find that in the very beginning no processors pass through the lock with high probability, and several processors become half-done with their writes. Conditioned on this, with constant probability the number of processors half-done with their writes remains high, and hence no processors pass through the lock in the middle. Finally, at the end, with high probability we are always in a state where "almost all" of the processors are half-done. By combining all of the conditioning appropriately, we find that no processor except the last passes through the lock at the end with probability bounded below by a constant. □

The proof of Theorem 7 is somewhat limiting, in that the read and write times are taken to be equal, and in practice one may desire a different initial state, such as when all processors start at the inspect phase. It appears that the theorem above should hold for more general cases; however, writing an appropriate generalization appears difficult. Finding a more elementary proof therefore remains an interesting question.

Theorem 7 has an interesting implication. Because a gamma(2) distribution is just the sum of two exponential distributions, we could easily turn a setting with exponentially distributed read and write times into one with gamma(2) distributed read and write times. Each read and write operation would simply be preceded by a "dummy" read or write operation. If the operations are uncorrelated, this effectively changes the distributions from exponential to gamma(2). Although this doubles the average time to obtain a lock, it changes the probability that a single processor successfully accesses the lock from a diminishing function of the number of processors n to something bounded below by a constant.

In fact, the dummy read or write operations are equivalent to a pause operation, where a pause takes a random amount of time. In Fischer's algorithm, only the read and not the write operation is delayed in this manner. It is therefore natural to now consider the case of Fischer's algorithm, where all operation times are exponential and there is a pause before the final read.

THEOREM 8. *Consider n processors beginning the write for a single lock, where writes and reads have independent exponential distributions with mean* 1, *and there is a pause before each final read of time that is also independent and exponentially distributed with mean* 1. *Then a single processor reads its own value with probability* $\frac{n+1}{2n}$.

Proof. For the jth processor to complete its write to read its own value, the corresponding pause and read operation must occur before any other writes occur. This happens with probability $(\frac{1}{n-j+1})^2$. Hence, all but the last processor to write fail to pass through the lock with probability

$$\begin{aligned}\prod_{j=2}^{n}\left(1-\frac{1}{j^2}\right) &= \prod_{j=2}^{n}\frac{j^2-1}{j^2}\\ &= \frac{\prod_{j=2}^{n}(j-1)\prod_{j=2}^{n}(j+1)}{\prod_{j=2}^{n}j\prod_{j=2}^{n}j}\\ &= \frac{n+1}{2n}.\end{aligned}$$

□

Theorem 8 demonstrates the importance of the pause operation in the context of Fischer's algorithm in the case of exponentially distributed operation times. The pause leads to a completely different type of behavior, avoiding conflict in the critical region over half of the time.

It is worth noting also that the approach to lower bound the failure probability for multiple locks from Theorem 6 can be extended to the case where operation times have the gamma distribution as well. Again, we just imagine two processors following each other through the proper stages and use the properties of the exponential distributions. Hence, the best we could hope for is a failure probability that decreases exponentially with the number of sequential locks.

4. Two protocols. We now apply some of the previous results in considering the performance of two mutual exclusion algorithms first suggested by Lynch and Shavit [12]. Both provide mutual exclusion and weak deadlock-freedom.

The first protocol we consider, given in Figure 4, is the combined Fischer–Lamport algorithm presented as Algorithm 3 in [12]. It uses two registers. We also consider an algorithm using three registers also discussed in [12] that is obtained by directly replacing the critical region of Fischer's algorithm with a Lamport style algorithm for mutual exclusion, as shown in Figure 5.

The scheme using three registers (FL2) behaves similarly to a double lock. The first lock is represented by the x register, and the second "lock" consists of both the y and z registers. Hence, with exponential service times, even without a pause, we would expect a constant probability for some processor to successfully execute the critical region on each trial. The logic is the same as that of Theorem 5; one fortunate early processor passes through the lock represented by register x and then reaches the critical region before another processor can block it.

The scheme using two registers behaves essentially like a single lock on the register x with the additional register y to ensure that only a single processor enters the critical region. It follows immediately from Theorem 8 that if the operation times are independently and exponentially distributed (including the pause), then a single

FL1
x, y: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq p$ then goto L;
if $y \neq 0$ then goto L;
$y := 1$;
if $x \neq p$ then goto L;
enter critical region;
exit critical region;
$y := 0$;
$x := 0$;
% Exiting ME-lock

FIG. 4. *A clever Fischer–Lamport combination.*

FL2
x, y, z: shared registers, initially 0
p: current process index

% Entering ME-lock
L:
if $x \neq 0$ then goto L;
$x := p$;
pause
if $x \neq i$ then goto L;
$y := p$;
if $z \neq 0$ then goto L;
$z := 1$;
if $y \neq p$ then goto L;
enter critical region;
exit critical region;
$z := 0$;
$x := 0$;
% Exiting ME-lock

FIG. 5. *A direct Fischer–Lamport combination.*

processor passes through the x lock and hence successfully executes the critical region with probability bounded below by some constant. Similarly, it is easy to show that the probability of a processor obtaining the critical region goes to 0 as the number of processors increases when the pause is removed. We formalize this explicitly.

THEOREM 9. *Consider n processors beginning at L in the algorithm FL*1 *of Figure* 4. *If writes and reads have independent exponential distributions with mean* 1, *and the pause takes time* 0 *(i.e., no pause), then the probability that any processor enters the critical region is $o(1)$.*

Proof. As usual, we assume that n is sufficiently large throughout. First, we note that with high probability $(1-o(1))$, at least $\Omega(\sqrt[3]{n})$ of the n processors starting at L reach the write step, as can be seen using the argument of Theorem 1 with $z = n^{1/3}$. We may therefore assume that we begin with $m = \Omega(\sqrt[3]{n})$ processors at the write stage.

We derive two bounds. The first shows that processors that complete the write to x early are unlikely to reach the critical region, and the second shows that processors that complete the write to x late are unlikely to reach the critical region.

The jth processor to write its own value in register x must read back its value, read register y, write register y, and read its own value again before any other processor writes to register x to obtain the critical region. By now familiar reasoning, the probability of all of these events occurring is $1/(m-j+1)^4$. By the union bound, the probability that any of first $m - m^{1/3}$ processors that write to register x read back its own value is

$$\sum_{j=1}^{m-m^{1/3}} \frac{1}{(m-j+1)^4} = o(1).$$

For the second bound, we consider the final $m^{1/3}$ processors that write their values into register x. Note that the jth processor to write its own value in register x can

reach the critical region only if no processor writes the value 1 on register y before this processor can read the register y. Consider the first $m - 5m^{1/3}$ processors. We claim that with probability $1 - o(1)$, one of these processors writes a 1 on register y before any of the final $m^{1/3}$ processors to write into register x reads register y.

By the same argument as Theorem 4, the probability that none of the first $m - 5m^{1/3}$ reads back its value from register x and proceeds to write to register y is

$$\left(1 - \frac{1}{(5m^{1/3}+1)}\right) \cdots \left(1 - \frac{1}{m}\right) = \frac{1}{5m^{1/3}} = o(1).$$

Hence, with probability $1-o(1)$ at least one processor attempts a write to y. Consider any such processor. For it to fail to write before the $(m - m^{1/3})$rd write to x, either y must have already been written over with a 1 (in which case we are done), or one the following events must occur:

- the read of y must occur after the $(m - 3m^{1/3})$rd write to x;
- the read of y occurs before the $(m - 3m^{1/3})$rd write to x and the write to y occurs after the $(m - m^{1/3})$rd write to x.

Since all operation times have the same mean, the probability of the first event is at most $1/2m^{1/3}$, and the probability of the second event is at most $1/2m^{1/3}$. By a union bound, the probability y still holds the value 0, for any of the last $(m - m^{1/3})$ writes is thus only $o(1)$.

Hence, considering both cases, a processor successfully enters the critical region with probability only $o(1)$. □

We note that we have not attempted to optimize the bounds of Theorem 9. A tight analysis would be interesting.

5. Simulations. In this section, we present the results of simulations of locks and double locks with varying service times, as well as examine the performance of some mutual exclusion algorithms that use locklike structures. The goal of this section is to demonstrate that our previous theorems accurately describe perceived performance, as well as gain more insight into the actual performance of mutual exclusion algorithms under these distributions.

We simulated single and double locks using operation times with an exponential distribution, a gamma(2) distribution, and a gamma(3) distribution. For the double lock, all operations have the same mean time, which we scale to be 1. For the single lock, we have simulated two cases: one where all operations have the same mean time, and one where the final read operation has mean 4, so that the total average time for a lock to try a processor is the same as that for a double lock. We call this a *long lock*. Each data point represents the fraction of 10,000 trials for which a single processor successfully passed through the lock.

The results are presented in Figure 6. We point out some features of interest. As expected, we find that a double lock dramatically outperforms a single lock in the case of the exponential distribution. Moreover, the poor performance of a single lock as the contention grows is clear. For the gamma distributions, however, the single lock performance does not deteriorate with contention, as expected. With a gamma(3) distribution, a single long lock outperforms a double lock.

Interestingly, the behavior as the number of processors increases is different for the three distributions. For the exponential distribution, the probability of success appears to decrease monotonically in the number of processors, while for the gamma(3) distribution the probability appears to increase monotonically in the number of processors. Meanwhile, for the gamma(2) distribution, the probability is nonmonotonic

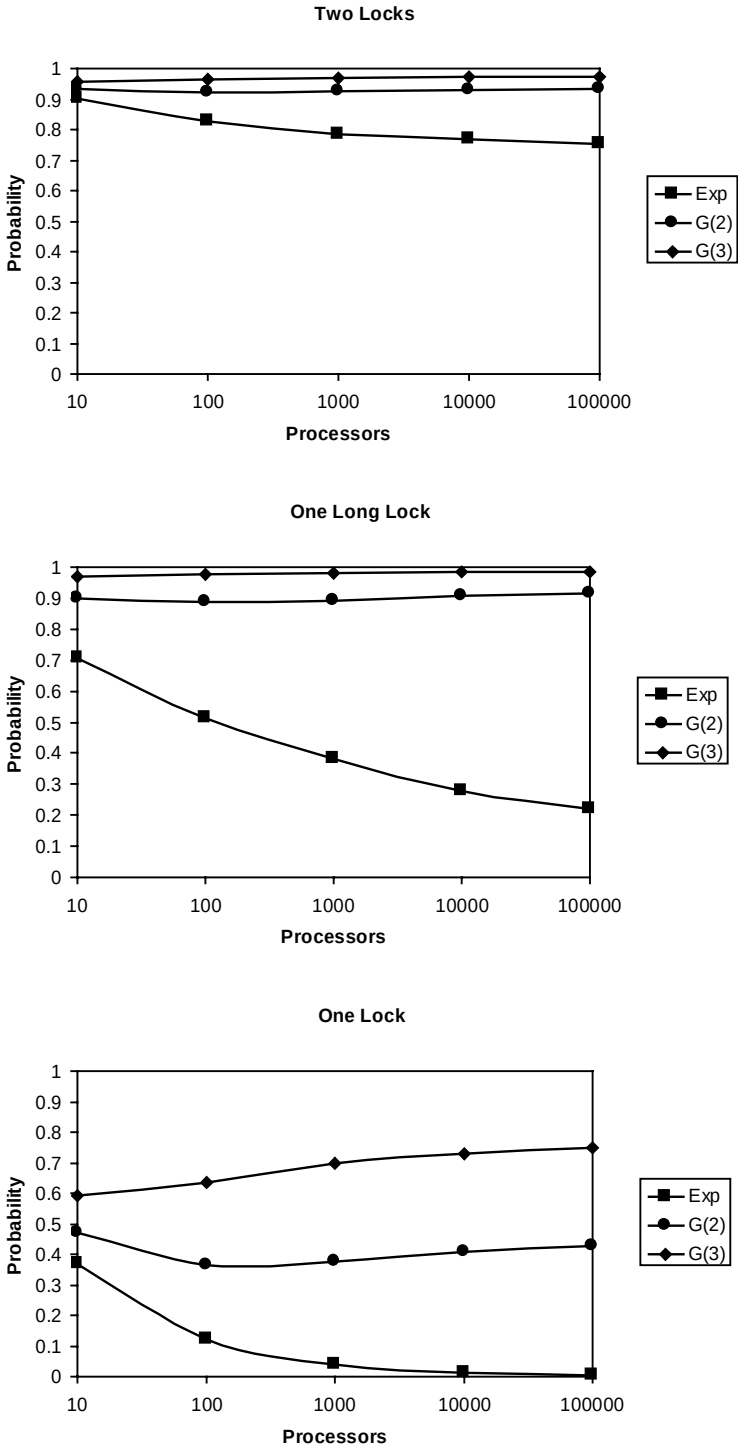


FIG. 6. *Comparing the behavior of a single lock and a double lock.*

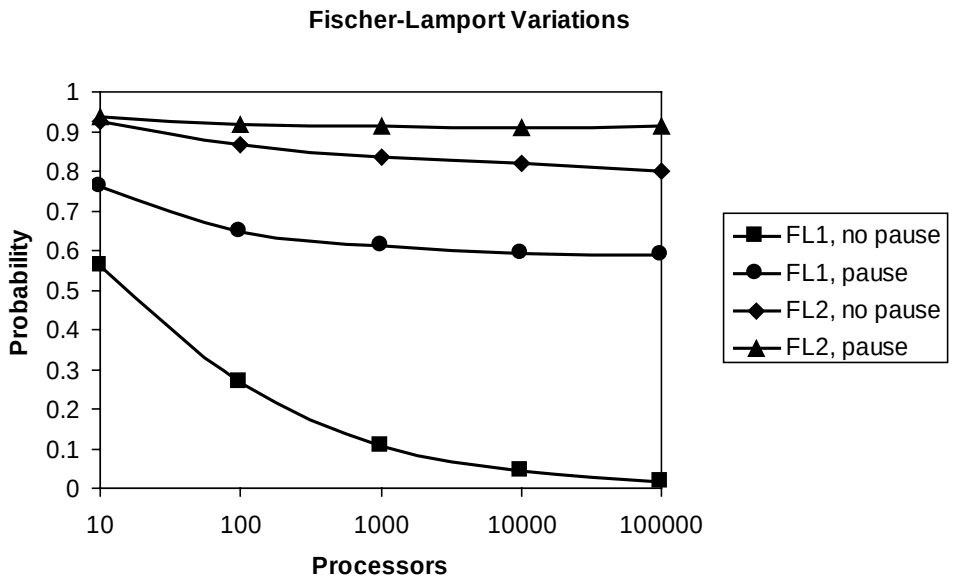


FIG. 7. *Comparing combined mutual exclusion algorithms.*

in the number of processors. This behavior may be worthy of future study, if only as a mathematical curiosity.

We also present some results for the mutual exclusion algorithms of section 4 in Figure 7. For these results, the distribution of the time for all operations is taken to be exponential with mean 1.

Note the dramatic effect of the pause in the performance of FL1. This is not surprising, given the analysis of section 4. Also, note that with the pause the FL1 algorithm succeeds a little more than 1/2 of the time. A rough approximation of this behavior is derivable from Theorem 8. Slightly over 1/2 of the time, a single processor will pass through the first lock. When multiple processors pass through the first lock, sometimes one will reach the critical region before any other processor can block it; this accounts for the additional probability of success. The mutual exclusion algorithm FL2 performs better, but of course it uses an extra register, and on average more time, since more reads and writes are performed by each processor. Tighter analyses or exhaustive simulations of the behavior of these algorithms might lead to a better comparison. It seems difficult to develop a more general statement as to which algorithm is preferable, as the decision may simply depend on the underlying timing distributions.

6. Conclusions and open questions. We have examined the behavior of timed locks under simple distributions, including exponential and gamma distributions, using both theoretical analysis and simulations. In particular, we have focused on the question of whether two locks are better than one and shown how it may depend on the distribution of the completion time of operations. We have also considered how this affects the design of mutual exclusion algorithms. Our work represents the first step toward designing a mutual exclusion algorithm based on random times that offers better performance in realistic situations than algorithms designed for the worst case.

We believe there are several ways to extend this work. A better understanding of the Markov chains underlying double or more extensive sequences of locks would

be interesting. For example, it would be appealing to determine with some accuracy the probability that only one processor passes through a double lock (even if only in the limiting case) by analyzing the underlying Markov chain in a more careful manner. Also, it would be worthwhile to understand the behavior of timed locks under more general distributions. In particular, truncated distributions where events occur within some bounded period of time may provide a more realistic description of actual behavior. Situations where the read and write times are somehow correlated may also be more realistic.

REFERENCES

[1] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[2] J. Aspnes, *Fast deterministic consensus in a noisy environment*, in Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000, pp. 299–308.

[3] D. Attanasio, M. Butrico, J. Peterson, C. Polyzois, and S. Smith, *Design and Implementation of a Recoverable Virtual Shared Disk*, IBM Research Report, RC 19843, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1994.

[4] P. Cao, S. B. Lim, S. Venkataraman, and J. Wilkes, *The tickerTAIP parallel RAID architecture*, ACM Trans. Comput. Systems, 12 (1994), pp. 236–267.

[5] M. Fischer, *personal communication* from [12].

[6] L. Goldberg, *Contention resolution notes*, available at http://www.dcs.warwick.ac.uk/∼leslie/contention.html.

[7] L. Goldberg and P. MacKenzie, *Analysis of backoff protocols for contention resolution with multiple servers*, J. Comput. System Sci., 58 (1999), pp. 232–258.

[8] L. Lamport, *A fast mutual exclusion algorithm*, ACM Trans. Comput. Systems, 5 (1987), pp. 1–11.

[9] E. K. Lee and C. A. Thekkath, *Petal: Distributed virtual disks*, in Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, 1996, pp. 84–92.

[10] B. Liskov, *Practical uses of synchronized clocks in distributed systems*, Distrib. Comput., 6 (1993), pp. 211–219.

[11] N. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, 1996.

[12] N. Lynch and N. Shavit, *Timing based mutual exclusion*, in Proceedings of the Annual IEEE Real-Time Symposium (RTSS), Phoenix, AZ, 1992, pp. 2–11.

[13] P. Raghavan and E. Upfal, *Stochastic contention resolution with short delays*, SIAM J. Comput., 28 (1998), pp. 709–719.

[14] C. A. Thekkath, T. Mann, and E. K. Lee, *Frangipani: A scalable distributed file system*, in Proceedings of the 16th ACM Symposium on Operating Systems Principles, 1997, pp. 224–237.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 838–865

A CHARACTERIZATION OF EVENTUAL BYZANTINE AGREEMENT*

JOSEPH Y. HALPERN†, YORAM MOSES‡, AND ORLI WAARTS§

Abstract. We investigate eventual Byzantine agreement (EBA) in the crash and omission failure modes. The emphasis is on characterizing optimal EBA protocols in terms of the states of knowledge required by the processors in order to attain EBA. It is well known that common knowledge among the nonfaulty processors is a necessary and sufficient condition for attaining *simultaneous* Byzantine agreement (SBA). We define a new variant that we call *continual* common knowledge and use it to provide necessary and sufficient conditions for attaining EBA. Using this characterization, we provide a technique that allows us to start with any EBA protocol and convert it to an optimal EBA protocol using a two-step process.

Key words. fault-tolerance, eventual Byzantine agreement, common knowledge, continual common knowledge, optimal protocol

AMS subject classifications. 68M14, 68M15, 68W15, 68N30

PII. S0097539798340217

1. Introduction.

In a distributed system, it is important to be able to design protocols that attain their goals despite the presence of unreliable components. The essence of the difficulties in doing this is captured by the well-studied problem of *Byzantine agreement* (BA), first introduced in [17]. Roughly speaking, a BA protocol provides a means for n processors, at most t of which may be faulty, to agree on a value in such a way that all nonfaulty processors decide on the same value, and when all processors start with the same initial value, the nonfaulty processors decide on this value. The version of the problem formulated in [17] did not require all processors to decide on the value simultaneously, yet the algorithms given in [17] (and many later papers) had the property that the processors did indeed decide simultaneously. In [3], it was pointed out that requiring simultaneous agreement could significantly affect the problem. Since then, both simultaneous Byzantine agreement (SBA), where processors are required to decide simultaneously, and eventual Byzantine agreement (EBA), where they are not, have received a great deal of attention.

We will be particularly interested here in the problem of reaching EBA as quickly as possible. The number of rounds it takes for a protocol to reach a decision depends in general on the pattern of failures, that is, on how and when failures occur. Thus, roughly speaking, we say that protocol P_1 *dominates* protocol P_2 if every nonfaulty

*Received by the editors June 12, 1998; accepted for publication (in revised form) July 17, 2001; published electronically December 18, 2001. An early version of this work appeared in *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing,* 1990.
http://www.siam.org/journals/sicomp/31-3/34021.html

†Computer Science Department, Cornell University, Ithaca, NY 14853 (halpern@cs.cornell.edu, http://www.cs.cornell.edu/home/halpern). Much of this work was carried out while the author was at the IBM Almaden Research Center. The work of this author was also supported in part by NSF under grants IRI-96-25901 and IIS-0090145, by ONR under grants N00014-00-1-03-41 and N00014-01-1-0795, and by the Air Force Office of Scientific Research under contract F49620-91-C-0080 and grant F49620-96-1-0323.

‡Department of Electrical Engineering, Technion, Israel (yoram@ee.technion.ac.il). This author was supported by a grant from the Israel Academy of Sciences. Much of this work was carried out while this author was at the Weizmann Institute.

§Stanford University, Stanford, CA 94035 (orli@neon.stanford.edu). This author was supported in part by contract ONR N00014-88-K-0166.

processor in a run of P_1 decides at least as soon as it does in the *corresponding* run of P_2 (where two runs are said to correspond if the initial values of all processors and the pattern of failures are the same in both); we say that P_1 *strictly dominates* P_2 if it dominates P_2 and in some run of P_1 at least one nonfaulty processor decides earlier than it does in the corresponding run of P_2. A protocol is *optimal* if it is not strictly dominated by another protocol; an *optimum* protocol is one that dominates every other protocol.

Optimum protocols are given for SBA in the case of *crash failures* [5], where a processor that is faulty sends no messages after it has failed, and in the case of the more general *(sending) omission* failures [14], where a processor may continue to send messages after it has failed. The construction of these protocols, as well as the analysis of their optimality, is done in terms of reasoning about knowledge [8, 6]. The key observation is that simultaneous actions are intimately related to *common knowledge*: it can be shown that a necessary and sufficient condition for optimal SBA is that the nonfaulty processors decide once they have common knowledge of a particular initial value.

By definition of EBA, different processors have the freedom to decide at different times. This freedom makes it possible to construct EBA protocols that typically decide much faster than SBA protocols [3]; it is also the cause of a number of subtle, but significant, differences between SBA and EBA. In particular (as already pointed out in [14]), although there are *optimal* protocols for EBA there are no *optimum* protocols for EBA.

Our goal here is to better understand the structure of EBA protocols and, in particular, optimal EBA protocols. A decision is made in an optimum SBA protocol as soon as common knowledge of some initial value is attained. This suggests that in an optimal EBA protocol, a decision should be made when some variant of common knowledge of an initial value is attained. A number of variants of common knowledge are discussed in [8]. Perhaps the most plausible candidate is *eventual* common knowledge. A fact φ is common knowledge if everyone knows that everyone knows that everyone knows ... φ; similarly, φ is eventual common knowledge essentially if eventually everyone will know that eventually everyone will know ... φ. However, a closer inspection shows that eventual common knowledge is an insufficient basis for decision in EBA. Specifically, it is not safe for a processor to decide as soon as it knows that some initial value is eventual common knowledge. For example, consider a protocol where a processor decides on the value v ($v \in \{0, 1\}$) once it knows that there is eventual common knowledge that someone started with initial value v. It is quite possible that at some point, processor i knows that there is eventual common knowledge that some initial value was 0 and does not know that there is eventual common knowledge that some initial value was 1, while processor j knows that there is eventual common knowledge that some initial value was 1 and does not know that there is eventual common knowledge that some initial value was 0. Thus, the obvious decision rule based on eventual common knowledge leads to inconsistency. (Note that, by way of contrast, with common knowledge it is the case that if one processor has common knowledge of φ, then all the processors do, so it is possible to define a consistent decision rule using common knowledge, for example, deciding on 0 if it is common knowledge that some initial value was 0 and deciding on 1 if it is common knowledge that some initial value was 1 and it is not common knowledge that some initial value was 0.)

One way to deal with the inconsistency described above is to require a processor

to decide 0 when it knows that there is eventual common knowledge that some initial value was 0 but to decide 1 only when it knows that there will *never* be eventual common knowledge that some initial value was 0. Although this state of knowledge is indeed sufficient for consistency, it does not guarantee optimality. That is, this protocol can be modified so that processors will be able to decide 1 earlier while still preserving consistency.

What is needed here is some condition that says "decide 1 as long as you are sure that everybody that ever knows there is eventual common knowledge that some initial value was 0 also knows (at the same time) that there is eventual common knowledge that some initial value was 1 and hence can decide 1." It turns out that in order to capture this type of condition, we need a new variant of common knowledge that we call *continual common knowledge*. Roughly speaking, a fact φ is continual common knowledge if throughout the run everyone knows that throughout the run everyone knows that ... φ.

We show that continual common knowledge plays a role in EBA that is somewhat analogous to that played by common knowledge in SBA. In particular, we can characterize the state of knowledge needed to attain EBA in terms of continual common knowledge, although it turns out that the precise characterization is more complicated than that of SBA. Using our characterization, we are also able to characterize optimal EBA protocols. Moreover, we provide a general technique for converting *any* EBA protocol P to an optimal EBA protocol P' dominating P. All of the analysis and the conversion is done at the knowledge level by working with high-level protocols with tests for knowledge.

This paper is organized as follows. In the next section we present our formal model of protocols and review the basic definitions of BA. In section 3, we review the knowledge formalism and introduce the notion of continual common knowledge. In section 4 we show that continual common knowledge is both a necessary and sufficient state of knowledge for an agreement protocol. We use these results in section 5 to characterize optimal knowledge-based protocols for EBA and then use our characterization to show how we can construct optimal EBA protocols, starting with arbitrary EBA protocols. Section 6 uses the technique described in section 5 to construct examples of optimal protocols, including a polynomial time optimal EBA protocol for the case of crash failures. We conclude with some discussion in section 7.

2. BA and full-information protocols. In this section, we review the BA problem, define the notion of optimality we are interested in, present our formal model of protocols, and show that, if optimality is our only concern, we can restrict attention to special protocols known as *full-information protocols*.

2.1. The BA problem. Suppose we are given a system with n processors, at most t of which might be faulty. Each processor i has an initial value $v_i \in V$. A *BA protocol* is one that satisfies the following properties:

1. *Decision.* Every nonfaulty processor i eventually decides (irreversibly) on a value $y_i \in V$.
2. *Consistency.* All nonfaulty processors decide on the same value.
3. *Validity.* If a nonfaulty processor decides v, then v must be the initial value of some processor.

The problem as stated above is actually EBA. In order to define SBA, we need to add one more condition:

4. *Simultaneity.* All the nonfaulty processors decide at the same round.

For simplicity, in this paper we focus on the case of *binary agreement*, where $V = \{0, 1\}$. Extending our methods to the case where $|V| > 2$ is straightforward.

It is well known that the BA problem is sensitive to the types of failures that might occur. We consider two basic failure modes in this paper:

1. *Crash failures.* A faulty processor behaves according to the protocol, except that it might commit a crash failure at an arbitrary round $k > 0$. If a processor commits a crash failure in round k (or simply *fails* in round k), then it obeys its protocol in all rounds preceding round k, it does not send any message in the rounds following k, and in round k it sends an arbitrary (not necessarily strict) subset of the messages it is required to send by its protocol.
2. *Omission failures.* A faulty processor behaves according to the protocol, except that it may omit sending an arbitrary set of message in any given round. (What we are calling "omission failures" here are what were termed *sending omission failures* in [14]; we do not consider *general omission failures* [18] here, where a processor may omit receiving a message as well as omitting to send one.)

We do not consider the more general *Byzantine failures*, where faulty processors may behave arbitrarily.

The first subtlety in comparing EBA and SBA already arises when we consider what we mean by a *nonfaulty* processor. Should we consider a processor that fails at some point in a run as faulty only after it has actually displayed faulty behavior, or should it be considered faulty throughout the run? In the case of SBA both interpretations work, while for EBA the distinction matters. Since all *nonfaulty* processors are required to decide consistently, if we choose the first interpretation, then a processor can decide on some value only when it is guaranteed that even if it fails after its decision, all nonfaulty processors will finally decide upon the same value. On the other hand, choosing the second interpretation enables the processor to decide on a value as long as it knows that, provided that it does not fail, all nonfaulty processors will decide on the same value. Clearly, in the crash or omission failure modes, when considering protocols in which the processors do not send any message after they decide, there is essentially no difference between these choices. Consequently, there is no difference between the choices in the case of SBA. However, there is a substantial difference in the case of EBA. For the purposes of this paper, we call a processor "nonfaulty" in a particular run only if it is nonfaulty throughout the run. This usage is consistent with the usual usage in other papers on EBA for crash and omission failures (see, e.g., [7, 11, 18]), although it differs somewhat from that of [5], where Dwork and Moses concentrate on the set of "active" processors, which can decrease over time.

Notice that in EBA we require all the nonfaulty processors to eventually decide on some value. It is occasionally useful to consider weaker notions. An *agreement protocol* is a protocol that satisfies the following two properties:

2′. *Weak consistency.* Nonfaulty processors do not decide on different values.

3′. *Weak validity.* If all initial values v_i are identical, then all nonfaulty processors that decide, decide v_i.

Note that an agreement protocol does not necessarily satisfy the decision property; a nonfaulty processor may not decide at all.

We discussed the notion of one protocol dominating another in the introduction; we will provide a formal definition of it in the next section. An *optimal protocol*

for EBA (SBA, agreement, etc.) is one that is not strictly dominated by any other protocol. An *optimum protocol* is one that dominates every other protocol. Although it is possible to find optimum protocols for SBA [5, 14], this is not the case for EBA. We give a proof here for the sake of completeness.

PROPOSITION 2.1 (see [14]). *There are no optimum EBA protocols.*

Proof. Consider a variant of an EBA protocol for the crash failure mode that was first introduced in [11]. When a processor first learns that some processor has an initial value of 0, it decides 0, relays 0 (i.e., sends 0 to all the other processors), and halts; if by time $t+1$ a processor does not know of any processors with initial value 0, it decides 1 and halts. It is easy to see that this protocol achieves EBA. Moreover, all nonfaulty processors with initial value 0 decide at time 0. Call this protocol $P0$. There is a symmetric protocol $P1$ where the roles of 0 and 1 are reversed. In $P1$, all nonfaulty processors with initial value 1 decide at time 0. An optimum EBA protocol would have to dominate both $P0$ and $P1$. Thus, in an optimum EBA protocol, all processors would have to decide at time 0. However, this is provably impossible [4]. □

This proof shows even more. Since it is well known [4] that in any EBA protocol there will always be some run in which some processor takes $t+1$ rounds to decide, it follows that given any EBA protocol P, there must be some run in which some processor takes at least $t+1$ rounds longer to decide than it does in one of $P0$ and $P1$. Thus, no protocol is guaranteed to even be close to optimum in all runs.

2.2. Optimal protocols: An example. We have just seen that there is no hope of obtaining optimum protocols for EBA. The existence of optimal protocols is, however, guaranteed. Intuitively, the reason is that there are a finite number of initial configurations of votes and a finite number of behaviors of faulty processors in the first k rounds for every finite k. An easy application of König's lemma thus shows that there must be a bound on the time at which all processors halt in all executions of any given EBA protocol. Thus, there cannot exist an infinite sequence $P_1, P_2, \ldots$ of correct EBA protocols where P_{i+1} strictly dominates P_i. It follows that if we start with an EBA protocol and repeatedly generate a strictly dominating protocol, then this process will terminate after a finite number of steps. Moreover, the last protocol in this process will be optimal.

One of the main results of this paper is an effective method of generating an optimal protocol dominating a given protocol. We now describe a simple optimal protocol in the crash failure mode that is generated in this way. Consider the protocol $P0$ defined in the proof of Proposition 2.1. In this protocol, processors decide 0 and send 0 to everyone else as soon as they learn that some processor had initial value 0. A processor that has not learned about a value of 0 by time $t+1$ decides 1 at time $t+1$. While this protocol is fairly efficient, it is not optimal.

Clearly, the fact that some processor had an initial value of 0 is propagated as fast as possible in this protocol. It follows that no correct EBA protocol can decide 0 any faster than $P0$ does because, by the specification of EBA, in order to decide 0, it is necessary that some processor started with an initial value of 0, and in $P0$, processors decide 0 as soon as they learn that some processor had an initial value of 0. How about deciding on 1? Here, it seems, the protocol $P0$ is not being as efficient as possible. For example, we know from the specification of EBA that if all initial values are 1, then a decision of 1 is forced. Obviously, in order to reach this decision as soon as possible in runs with no failures, a processor should notify the other processors in the first round about its having an initial value of 1 as well as about its having an

initial value of 0.

One way to get an optimal EBA protocol is to get a rule for deciding 1 as soon as possible without changing the rule used by $P0$ for deciding on 0. We now design a protocol with this property. The idea is that processors decide 1 as soon as they know that nobody is ever going to know that some processor had an initial value of 0. The protocol uses the same rule for deciding on 0 as $P0$. The rule for deciding on 1 is based on the observation that processor i knows that no nonfaulty processor will ever know that some processor had an initial value of 0 if either

(a) processor i knows that all initial values are 1 or
(b) every processor sends a message to all processors in every round telling them which initial values each processor has (as far as it knows), i hears from the same set of processors in two consecutive rounds, and i still does not know that some processor had an initial value of 0.

Thus, each processor i maintains a list of the initial values it knows about and sends this list to all other processors in every round. It decides 1 and communicates for one more round if one of the two properties above holds. We call this protocol $P0_{\text{opt}}$. It is easy to check that by time $t+1$, either processor i will learn that some processor had an initial value of 0, or one of the properties (a) or (b) above holds. $P0_{\text{opt}}$ thus dominates $P0$. Indeed, as we formally show in section 6, $P0_{\text{opt}}$ is an optimal EBA protocol for the crash failure mode. In fact, it is the unique optimal protocol dominating $P0$.

2.3. Protocols and systems. Up to now we have spoken informally of "protocols." We now give a more formal definition of the model and notions involved along the lines of [5, 14, 6]; the interested reader is referred to [6] for more detail.

We consider a synchronous distributed system consisting of a finite collection of $n \geq 2$ processors (automata) $\{i_1, \ldots, i_n\}$, each pair of which is connected by a two-way communication link. The processors share a global clock that starts out at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round k taking place between time $k-1$ and time k. In each round, every processor first sends the messages it needs to send to other processors and then receives the messages that were sent to it by other processors in the same round. The identity of the sender and destination of each message, as well as the round in which it is sent, are assumed to be part of the message.

We think of the processors as following a *protocol*, which specifies the actions of each of the nonfaulty processors as a *deterministic* function of the processor's local state.[1]

A *run* of a protocol is a complete description of the system at each time step. This includes each processor's initial state the messages sent in every round, the decisions taken by the processors, and the behavior of the faulty processors. We assume that at every point in time each processor has a well-defined local state, which is recorded in the run and which the protocol uses to determine the actions that the processor takes. A *point* is a pair (r, k) consisting of a run r and a time k. We use $r_i(k)$ to denote processor i's local state at the point (r, k). For technical reasons, it is convenient to assume that communication happens during a round (that is, between two points (r, m) and $(r, m+1)$) but that a decision is actually made at a given point, not during the round. Thus, we talk about a message being sent in round k and a decision being

[1] As discussed in detail in [14], a protocol is typically stated in terms of parameters n and t, standing for the number of processors, and an upper bound on the number of failures. We do not represent this explicitly here, since this fact does not affect our results in this paper.

made at time k (of some run r). For simplicity, we assume that the only possible outputs (decisions) are 0 and 1, and that a processor can output at most one of these values at a given time. A *system* $\mathcal{R}$ is just a set of runs. We can associate with a protocol P the systems $\mathcal{R}_P^{cr}(n,t)$ for all $n \geq t \geq 0$, consisting of its possible runs with n processors in which no more than t processors crash. We similarly associate with P the analogous systems $\mathcal{R}_P^{om}(n,t)$ corresponding to the omissions failure model. We omit the parameters of n and t as the results are typically independent of these values. Similarly, for the most part, our results do not depend on whether we consider crash failures or omission failures. We omit the superscript and just write $\mathcal{R}_P$ whenever the type of failure is not relevant to the discussion.

As discussed in the previous section, we say that a processor is *nonfaulty* in a given run if it follows the protocol throughout the run. Since we are considering only crash and omission failures here, if i is a faulty processor in a run r, then its faulty behavior can be characterized by describing to which processes i omits sending required messages at each round. The *failure pattern* of a run contains the faulty behavior of all the processors that fail in the run. (A more detailed and formal exposition of the model can be found in [6, 14]; this brief exposition should suffice for stating and proving our results.)

The list of processors' initial states is the system's *initial configuration*. A protocol P, an initial configuration, and a failure pattern uniquely determine a run. Two runs r and r' of protocols P and P' are called *corresponding runs* if they have the same initial configuration and result from the same failure pattern. Two points (r,m) and (r',m') of protocols P and P' are *corresponding points* if r and r' are corresponding runs of the two protocols and $m = m'$. As we said in the introduction, given two protocols P and P', we say that P *dominates* P' if every nonfaulty processor that makes a decision (i.e., outputs 0 or 1 at some point in a run) in a run r' of P' also does so in the corresponding run r of P, and in fact decides at least as soon in r as it does in r'. We say P *strictly dominates* P' if P dominates P' and in some run r of P there is a nonfaulty processor that decides sooner in r than it does in the corresponding run of P'.

Notice that our definition of "dominate" focuses on when the processors decide and not when they halt. However, in all our protocols the processors can always halt after they know that all the nonfaulty processors have decided. In many cases they can in fact halt one round after they decide. Thus, we ignore issues of termination here and assume that protocols run forever.

2.4. Full-information protocols. We now review the notion of a full-information protocol and some of its implications. The treatment in this section follows the lines of [2].

A protocol is said to be a *full-information* protocol if each processor is required to send its current state to all processors at each round. The state of a processor in a full-information protocol consists of the processor's name, initial state, message history, and the time on the global clock. Thus, the states of processors following a full-information protocol are completely independent of their final decisions; at corresponding points in two full-information protocols, processors have the same states. Indeed, the only way in which full-information protocols can differ is in the decisions made by the processors.

Intuitively, the states of the processors in a full-information protocol make the finest possible distinctions among histories. That is why the full-information protocol is particularly well suited for proving possibility and impossibility of achieving certain

goals in distributed systems, and for the design and analysis of distributed protocols. The following proposition, essentially due to Coan [2], and corollary formalize this intuition.

PROPOSITION 2.2. *Let P be an arbitrary protocol in a system with omission (resp., crash) failures. Then for each processor i there is a function f_i from i's state in a full-information protocol F to its state in P such that for every pair (r, m) and (r', m) of corresponding points of F and P, we have $f_i(r_i(m)) = r'_i(m)$.*

As an immediate consequence, we get the following corollary.

COROLLARY 2.3. *Let P be an arbitrary protocol in a system with omission (resp., crash) failures. Then there is a full-information protocol that dominates P.*

This proposition shows that, just as in [5, 14] for the case of SBA, we can restrict attention to full-information protocols when looking for optimal EBA protocols.

3. Continual common knowledge. As we mentioned in the introduction, we want to perform a knowledge-theoretic analysis of EBA. In the case of SBA, the notion of common knowledge was shown to be central, and its use facilitated the construction of optimal (indeed optimum) protocols. A similar role for EBA will be played by the notion of continual common knowledge, which is a strengthening of common knowledge. Our purpose in this section is to define continual common knowledge and present some of its most useful properties. To do so, we start by describing the knowledge formalism and give definitions and properties of knowledge, belief, and common knowledge. The reader familiar with these notions can skip directly to section 3.2.

3.1. The knowledge formalism. We want to be able to reason about the states of knowledge of the processors in the system. In order to do so, we use the formalism first introduced in [8].

We start with a collection of basic facts (which can essentially be thought of as primitive propositions). For each such basic fact, it will typically be clear whether or not it is true at a given point.[2] We define various basic facts as we go along. Among the basic facts of interest is $\exists 0$, which is true at a point (r, k) in $\mathcal{R}$ if some processor started with initial value 0 in r; we similarly define $\exists 1$. We close this language under the standard Boolean connectives $\wedge$, $\neg$, and $\Rightarrow$, interpreted as conjunction, negation, and implication, as well as various knowledge operators. The basic operator is K_i, where $K_i\varphi$ is read "processor i knows φ."

A processor is said to *know* a fact at a given point (r, m) exactly if the fact holds at all of the points in which the processor has the same state as at (r, m). Thus, we have $(\mathcal{R}, r, m) \models K_i\varphi$ if $(\mathcal{R}, r', m') \models \varphi$ for all points (r', m') such that $r_i(m) = r'_i(m')$. Given a system $\mathcal{R}$, a formula φ is said to be *valid in* $\mathcal{R}$, denoted $\mathcal{R} \models \varphi$, if it holds at all points in $\mathcal{R}$. It is well known that our definition of knowledge satisfies the axioms of the modal logic S5.

PROPOSITION 3.1 (see [9]). *For every system $\mathcal{R}$ we have*

(a) *if $\mathcal{R} \models \varphi$, then $\mathcal{R} \models K_i\varphi$ (knowledge generalization);*
(b) *$\mathcal{R} \models (K_i\varphi \wedge K_i(\varphi \Rightarrow \psi)) \Rightarrow K_i\psi$ (distribution axiom);*
(c) *$\mathcal{R} \models K_i\varphi \Rightarrow \varphi$ (knowledge axiom);*
(d) *$\mathcal{R} \models K_i\varphi \Rightarrow K_iK_i\varphi$ (positive introspection axiom);*

[2] Strictly speaking, we need not just the system, but an *interpreted system*, which is a system together with an *interpretation* of the primitive propositions (see [6] for details). In this paper the basic facts of interest are few and their interpretation will be clear from context, so we do not explicitly use interpretations here.

(e) $\mathcal{R} \models \neg K_i\varphi \Rightarrow K_i\neg K_i\varphi$ *(negative introspection axiom).*

Having defined knowledge for individual processors, we now extend this definition to states of knowledge of a group of processors. We are mainly interested in the state of knowledge called *common knowledge*, since this has been shown to be closely related to coordination and agreement [8]. Given a set G of processors, let $E_G\varphi$ be an abbreviation for $\wedge_{i\in G}K_i\varphi$. Thus, $E_G\varphi$ holds if every processor in G knows φ. $C_G\varphi$ (φ is common knowledge among the processors in G) holds if everyone in G knows φ, everyone knows that everyone knows, and so on. Formally, taking $E_G^1\varphi$ to be an abbreviation for $E_G\varphi$ and $E_G^{k+1}\varphi$ to be an abbreviation for $E_GE_G^k\varphi$, we define

$$(\mathcal{R}, r, m) \models C_G\varphi \text{ iff } (\mathcal{R}, r, m) \models E_G^k\varphi \text{ for } k = 1, 2, 3, \ldots.$$

In the analysis in [5, 14] of SBA, the key concept is not common knowledge among a fixed set G of processors but common knowledge among the nonfaulty processors. The set of nonfaulty processors is a *nonrigid* set: its elements vary from one point to another. Formally, given a system $\mathcal{R}$, a nonrigid set $\mathcal{S}$ of processors in the system is a function associating with every point of the system a subset of the processors. In other words, $\mathcal{S}(r, m)$ is a (possibly different) set of processors for every point (r, m) of $\mathcal{R}$. We denote by $\mathcal{N}$ the nonrigid set of nonfaulty processors. Since a nonfaulty processor does not necessarily know that it is nonfaulty, we would like a notion of knowledge that is appropriate even when processors are not guaranteed to know whether they belong to the nonrigid set. Thus, following [14], given a nonrigid set $\mathcal{S}$ and a processor i, we define $B_i^{\mathcal{S}}\varphi = K_i(i \in \mathcal{S} \Rightarrow \varphi)$. More formally,

$$\begin{aligned}(\mathcal{R}, r, m) \models B_i^{\mathcal{S}}\varphi \text{ iff } &(\mathcal{R}, r', m') \models \varphi \text{ for all } (r', m')\\ &\text{such that } r_i(m) = r_i'(m') \text{ and } i \in \mathcal{S}(r', m').\end{aligned}$$

In other words, $B_i^{\mathcal{S}}\varphi$ holds if i knows that *if it is in* $\mathcal{S}$, then φ holds. This way of defining belief is a special case of a notion of belief defined by Moses and Shoham [13]. As they show, $B_i^{\mathcal{S}}$ behaves much like knowledge. The following proposition summarizes some of its properties.

PROPOSITION 3.2 (see [13]). *For every system* $\mathcal{R}$ *we have*

(a) *if* $\mathcal{R} \models (i \in \mathcal{S}) \Rightarrow \varphi$, *then* $\mathcal{R} \models B_i^{\mathcal{S}}\varphi$;
(b) $\mathcal{R} \models (B_i^{\mathcal{S}}\varphi \wedge B_i^{\mathcal{S}}(\varphi \Rightarrow \psi)) \Rightarrow B_i^{\mathcal{S}}\psi$;
(c) $\mathcal{R} \models i \in \mathcal{S} \Rightarrow (B_i^{\mathcal{S}}\varphi \Rightarrow \varphi)$;
(d) $\mathcal{R} \models B_i^{\mathcal{S}}\varphi \Rightarrow B_i^{\mathcal{S}}B_i^{\mathcal{S}}\varphi$;
(e) $\mathcal{R} \models \neg B_i^{\mathcal{S}}\varphi \Rightarrow B_i^{\mathcal{S}}\neg B_i^{\mathcal{S}}\varphi$;
(f) $\mathcal{R} \models K_i\varphi \Rightarrow B_i^{\mathcal{S}}\varphi$.

Properties (a), (b), (d), (e) are the obvious analogues of the corresponding properties in Proposition 3.1. Property (c) characterizes what is considered to be the key difference between knowledge and belief. By definition, it is impossible to know something which is false, while it is possible to have false beliefs. Thus, $B_i^{\mathcal{S}}$ can be considered to be a notion of belief, but a special one, in that a processor in $\mathcal{S}$ cannot have false beliefs. An extreme case that will appear in our technical analysis is when a processor knows that it is *not* in the set $\mathcal{S}$. When this happens, the processor believes everything:

$$\models \quad K_i(i \notin \mathcal{S}) \Rightarrow B_i^{\mathcal{S}}\varphi.$$

We define $E_{\mathcal{S}}\varphi$ as $\bigwedge_{i\in\mathcal{S}} B_i^{\mathcal{S}}\varphi$. In other words, *everyone in* $\mathcal{S}$ *knows* φ if every processor in $\mathcal{S}$ knows that if it is in $\mathcal{S}$, then φ holds. Notice that if $\mathcal{S}(r, m)$ is empty, then, by definition, $E_{\mathcal{S}}\varphi$ holds at (r, m) (for all formulas φ).

The notion of $C_{\mathcal{S}}\varphi$ is now defined as an infinite conjunction in terms of $E_{\mathcal{S}}\varphi$. Defining $E_{\mathcal{S}}^{k+1}\varphi$ inductively as an abbreviation for $E_{\mathcal{S}}E_{\mathcal{S}}^{k}\varphi$, we have

$$(\mathcal{R}, r, m) \models C_{\mathcal{S}}\varphi \quad \text{iff} \quad (\mathcal{R}, r, m) \models E_{\mathcal{S}}^{k}\varphi \text{ for } k = 1, 2, \ldots.$$

It is easy to see that if $\mathcal{S}$ is the constant function that always returns G, then $C_{\mathcal{S}}\varphi$ is equivalent to $C_G\varphi$. Thus, this definition of $C_{\mathcal{S}}\varphi$ extends the original definition of $C_G\varphi$ to nonrigid sets.

For the sake of completeness, we briefly mention the formal definition of the linear-time temporal operator $\Diamond$ (standing for "eventually") and its dual $\Box$ ("henceforth") in our setting. These operators will be used later on; for details about linear-time temporal logic, see [12]:

$$(\mathcal{R}, r, m) \models \Diamond\varphi \quad \text{iff} \quad (\mathcal{R}, r, m') \models \varphi \text{ for some } m' \geq m,$$

$$(\mathcal{R}, r, m) \models \Box\varphi \quad \text{iff} \quad (\mathcal{R}, r, m') \models \varphi \text{ for all } m' \geq m.$$

3.2. Eventual common knowledge and agreement. As shown in [5, 14], common knowledge among the nonfaulty processors is just the right tool for characterizing optimal SBA. It is not quite right for EBA though, since in EBA it is clear that a processor can decide 0 (or 1) well before it knows that other processors know about this value and thus, a fortiori, before this value is common knowledge. As we said in the introduction, we might have hoped that *eventual common knowledge* would be the appropriate replacement for common knowledge, but that does not quite work.

To understand why, consider the following full-information protocol F_0 that uses eventual common knowledge in its decision rule. While we have not given a formal definition of eventual common knowledge here (one can be found in [6, 9]), for the purposes of this discussion it suffices to know that if φ eventually becomes common knowledge, then it is eventual common knowledge. That is, if we denote eventual common knowledge of φ by $C^{\Diamond}\varphi$, then $\Diamond C\varphi \Rightarrow C^{\Diamond}\varphi$ is valid. According to F_0, a processor decides 0 when it knows there is eventual common knowledge that some initial value was 0 (i.e., when it knows that $C^{\Diamond}\exists 0$ holds) and decides 1 when it knows not only that there is eventual common knowledge that some initial value was 1, but also that there can never be eventual common knowledge that some initial value was 0 (i.e., when it knows that $C^{\Diamond}\exists 1 \wedge \Box\neg C^{\Diamond}\exists 0$ holds). Clearly F_0 is an agreement protocol.

Although the state of knowledge required for decision according to this rule is sufficient for agreement, it is not necessary. Just as in the case of the protocol $P0$ of section 2.2, it is possible to decide 1 earlier than F_0. For example, consider a run of the full-information protocol in the omission failure mode, in which all processors start with initial value 1, there are t faulty processors, and the t faulty processors send no messages in the first two rounds. Techniques of [14] show that in this case it is common knowledge at the end of the second round that there exists an initial value of 1 (indeed, the values of all nonfaulty processors are common knowledge), but no processor knows that all initial values are 1. Indeed, each processor considers it possible that one of the faulty processors has an initial value of 0 and that it will send it to some nonfaulty processor in the third round. The analysis in [14] can be used to show that, should such a message be received in the third round, the existence of a value of 0 will become common knowledge (and thus also be eventual common knowledge) at the end of the fourth round. Thus, according to the protocol F_0, no nonfaulty processor decides at the end of the second round in such a run. However, it is not hard to construct a protocol which dominates F_0 and does decide at this point.

To understand how this can be done, note that the reason we require a processor to wait until it knows that $C^\diamond\exists 1 \wedge \Box\neg C^\diamond\exists 0$ holds before it decides 1 is that otherwise we may have inconsistency: some processor may decide 0 while another may decide 1. Can we decide earlier while still avoiding inconsistency? One thought might be to have a processor decide 1 if it knows that $C^\diamond\exists 1$ holds and that every nonfaulty processor that learns $C^\diamond\exists 0$ does so after it learns $C^\diamond\exists 1$. However, to avoid inconsistency, we then have to modify the rule for deciding 0 so that a processor decides 0 if it knows that $C^\diamond\exists 0$ holds and it considers it possible that some nonfaulty processor will learn $C^\diamond\exists 0$ no later than $C^\diamond\exists 1$. It is easy to see that this gives us an agreement protocol: it is impossible for one nonfaulty processor to decide 0 and another to decide 1. Moreover, it is not hard to show that this protocol dominates F_0: all processors decide no later in runs of this protocol than in the corresponding run of F_0 (although their decisions in corresponding runs may be different).

This protocol itself can be improved though by refining the rule for deciding 0 in a similar fashion. It turns out that an optimal protocol is obtained when the process of refining the decision rules reaches a fixed point. These ideas can be formally captured by use of a new state of knowledge that we call *continual common knowledge*. This is the subject of the next section.

3.3. The definition of continual common knowledge. Roughly speaking, a fact φ is continual common knowledge in a run r among the processors in the nonrigid set $\mathcal{S}$ if, at all points (r, m'), every processor that belongs to $\mathcal{S}(r, m')$ knows that at every point in the run every processor that belongs to $\mathcal{S}$ at that point knows that ... φ holds. More formally, we first define

$$(\mathcal{R}, r, m) \models \boxdot\psi \quad \text{iff} \quad (\mathcal{R}, r, m') \models \psi \ \text{ for all } m' \geq 0.$$

Thus, $\boxdot$ is analogous to the standard temporal logic operator $\Box$, except that instead of restricting attention to the present and future as we do with $\Box$, with $\boxdot$ we consider all times past, present, and future. Note that $(\mathcal{R}, r, m) \models \boxdot\psi$ iff $(\mathcal{R}, r, 0) \models \Box\psi$.

We define $E^{\boxdot}_{\mathcal{S}}\varphi$ as an abbreviation for $\boxdot E_{\mathcal{S}}\varphi$. In other words, $E^{\boxdot}_{\mathcal{S}}\varphi$ holds at the point (r, m) if for all times m', every processor in $\mathcal{S}(r, m')$ knows at time m' that if it is in $\mathcal{S}$, then φ holds. The twist here is, of course, that because $\mathcal{S}$ is nonrigid, the members of $\mathcal{S}(r, m')$ may be different for different values of r and m'. Notice that if $\mathcal{S}(r, m')$ is empty for all $m' \geq 0$, then by definition $E^{\boxdot}_{\mathcal{S}}\varphi$ holds at (r, m) (and, in fact, throughout the run r).

The notion of continual common knowledge of φ is now defined as an infinite conjunction in terms of $E^{\boxdot}_{\mathcal{S}}\varphi$. Defining $(E^{\boxdot}_{\mathcal{S}})^{k+1}\varphi$ inductively as an abbreviation for $E^{\boxdot}_{\mathcal{S}}(E^{\boxdot}_{\mathcal{S}})^k\varphi$, we define

$$(\mathcal{R}, r, m) \models C^{\boxdot}_{\mathcal{S}}\varphi \quad \text{iff} \quad (\mathcal{R}, r, m) \models (E^{\boxdot}_{\mathcal{S}})^k\varphi \text{ for every } k \geq 1.$$

Thus, $C^{\boxdot}_{\mathcal{S}}\varphi$ holds if at all times everyone in $\mathcal{S}$ knows that at all times everyone in $\mathcal{S}$ knows that ... φ holds. We remark that just as $C_{\mathcal{S}}\varphi$ can be shown to be a greatest fixed point of the equation $X \Leftrightarrow E_{\mathcal{S}}(\varphi \wedge X)$ [6, 8], so $C^{\boxdot}_{\mathcal{S}}\varphi$ is the greatest fixed point of the equation $X \Leftrightarrow E^{\boxdot}_{\mathcal{S}}(\varphi \wedge X)$; we omit further details here.

We can characterize continual common knowledge as follows. A point (r', m') is said to be $\mathcal{S}$-$\boxdot$-*reachable in k steps* from a point (r, m) if there exist runs $r^0, \ldots, r^k$, times $m_0, m'_0, \ldots, m_{k-1}, m'_{k-1}$, and processors $i_0, \ldots, i_{k-1}$ such that $r^0 = r$, $r^k = r'$, $m'_{k-1} = m'$, and, for $j = 0, \ldots, k-1$, we have that $i_j \in \mathcal{S}(r^j, m_j) \cap \mathcal{S}(r^{j+1}, m'_j)$

and $r_{i_j}^j(m_j) = r_{i_j}^{j+1}(m'_j)$.[3] We say that (r', m') is $\mathcal{S}$-$\boxdot$-*reachable from* (r, m) if it is $\mathcal{S}$-$\boxdot$-reachable from (r, m) in k steps for some k. It is not too hard to check that the following proposition holds.

PROPOSITION 3.3. $(\mathcal{R}, r, m) \models (E_{\mathcal{S}}^{\boxdot})^k \varphi$ *iff* $(\mathcal{R}, r', m') \models \varphi$ *for all points* (r', m') *that are* $\mathcal{S}$-$\boxdot$-*reachable from* (r, m) *in* k *steps.*

COROLLARY 3.4. $(\mathcal{R}, r, m) \models C_{\mathcal{S}}^{\boxdot} \varphi$ *iff* $(\mathcal{R}, r', m') \models \varphi$ *for all points* (r', m') *that are* $\mathcal{S}$-$\boxdot$-*reachable from* (r, m).

We remark that this result is a variant of the characterization of $C_{\mathcal{S}}\varphi$ given in [5]. Using that characterization, it was shown that $C_{\mathcal{S}}$ satisfies all the S5 axioms except the knowledge axiom[4] and that $C_{\mathcal{S}}\varphi \Rightarrow \varphi$ holds at points where $\mathcal{S}$ is nonempty.[5] $C_{\mathcal{S}}$ was also shown to satisfy two additional properties, known as the *induction rule* and the *fixed-point axiom*. Using Corollary 3.4, we can prove the analogous properties for continual common knowledge.

LEMMA 3.5. *For every system* $\mathcal{R}$ *we have that*

(a) *if* $\mathcal{R} \models \varphi$, *then* $\mathcal{R} \models C_{\mathcal{S}}^{\boxdot} \varphi$ *(common knowledge generalization);*
(b) $\mathcal{R} \models (C_{\mathcal{S}}^{\boxdot} \varphi \wedge C_{\mathcal{S}}^{\boxdot} (\varphi \Rightarrow \psi)) \Rightarrow C_{\mathcal{S}}^{\boxdot} \psi$ *(distribution axiom);*
(c) $\mathcal{R} \models C_{\mathcal{S}}^{\boxdot} \varphi \Rightarrow C_{\mathcal{S}}^{\boxdot} C_{\mathcal{S}}^{\boxdot} \varphi$ *(positive introspection);*
(d) $\mathcal{R} \models \neg C_{\mathcal{S}}^{\boxdot} \varphi \Rightarrow C_{\mathcal{S}}^{\boxdot} \neg C_{\mathcal{S}}^{\boxdot} \varphi$ *(negative introspection);*
(e) $\mathcal{R} \models C_{\mathcal{S}}^{\boxdot} \varphi \Leftrightarrow E_{\mathcal{S}}^{\boxdot} (\varphi \wedge C_{\mathcal{S}}^{\boxdot} \varphi)$ *(fixed-point axiom);*
(f) *if* $\mathcal{R} \models \varphi \Rightarrow E_{\mathcal{S}}^{\boxdot} (\varphi \wedge \psi)$, *then* $\mathcal{R} \models \varphi \Rightarrow C_{\mathcal{S}}^{\boxdot} \psi$ *(induction rule);*
(g) $\mathcal{R} \models C_{\mathcal{S}}^{\boxdot} \varphi \Rightarrow \boxdot C_{\mathcal{S}}^{\boxdot} \varphi$;
(h) $\mathcal{R} \models (i \in \mathcal{S}) \Rightarrow (C_{\mathcal{S}}^{\boxdot} \varphi \Leftrightarrow B_i^{\mathcal{S}} C_{\mathcal{S}}^{\boxdot} \varphi)$.

Proof. The proof of parts (a)–(f) is essentially identical to the proof of the analogous properties for $C_{\mathcal{S}}$ in [5], so it is omitted here. Part (g) follows from the observation that for all m, m', the set of points $\mathcal{S}$-$\boxdot$-reachable from (r, m) and (r, m') is identical. Part (h) follows easily from part (e) and part (c) of Proposition 3.2. We leave details to the reader. □

It is easy to see that $\models C_{\mathcal{S}}^{\boxdot} \varphi \Rightarrow C_{\mathcal{S}}\varphi$ for all formulas φ. It is also not hard to show that the converse does not hold in general. Thus, continual common knowledge is a variant of common knowledge that is strictly stronger than common knowledge.

4. Continual common knowledge and agreement. In this section, we begin our analysis of EBA in terms of continual common knowledge. Let $decide_i(y)$, $y = 0, 1$, be the basic fact which is true at all points where processor i decides y. Since the truth of $decide_i(y)$ at a point depends only on processor i's local state at that point, when $decide_i(y)$ holds, processor i knows it. Moreover, since we have assumed that a processor cannot output 0 and 1 at the same time, we cannot have both $decide_i(0)$ and $decide_i(1)$ holding at any point. Combining these observations with parts (c) and (f) of Proposition 3.2, we immediately get the following result.

[3] The particular models we consider in this paper are *synchronous*, which means that if $r_j(m) = r_j(m')$, then $m = m'$. In synchronous models, we can simplify the notion of reachability slightly because we can take $m_i = m'_i$ for $i = 0, \ldots, k-1$.

[4] The modal system satisfying precisely these properties is known as K45 [1].

[5] Actually, in [5] it is claimed that $C_{\mathcal{S}}$ even satisfies the knowledge axiom. This is indeed the case in the particular application in [5], in which the nonrigid sets $\mathcal{S}$ of interest are always nonempty. However, $C_{\mathcal{S}}$ does not satisfy the knowledge axiom in the more general case in which the nonrigid set might occasionally be empty.

PROPOSITION 4.1. *Let $\mathcal{N}$ stand for the set of nonfaulty processors. If P is an agreement protocol, then for $y = 0, 1$ we have*

(a) $\mathcal{R}_P \models \mathit{decide}_i(y) \Rightarrow \neg \mathit{decide}_i(1-y)$;
(b) $\mathcal{R}_P \models (K_i \mathit{decide}_i(y) \Leftrightarrow \mathit{decide}_i(y)) \wedge (K_i \neg \mathit{decide}_i(y) \Leftrightarrow \neg \mathit{decide}_i(y))$;
(c) $\mathcal{R}_P \models i \in \mathcal{N} \Rightarrow ((B_i^{\mathcal{N}} \mathit{decide}_i(y) \Leftrightarrow \mathit{decide}_i(y)) \wedge (B_i^{\mathcal{N}} \neg \mathit{decide}_i(y) \Leftrightarrow \neg \mathit{decide}_i(y)))$.

Another property of agreement protocols that we will use later on is the following.

LEMMA 4.2. *Let P be an agreement protocol and assume that i, j are nonfaulty in run r of P. Then for all times m we have $(\mathcal{R}_P, r, m) \models \mathit{decide}_i(0) \Rightarrow \boxdot \neg \mathit{decide}_j(1)$.*

Proof. The result is immediate from the definitions. □

We want to focus on the decisions made according to protocol P. To do this, the following definitions will be helpful. We say that a *decision set* $\mathcal{A}$ is a tuple $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, where $\mathcal{A}_i$ is a subset of processor i's states. We think of the $\mathcal{A}_i$ component of a decision set as consisting of all the states where processor i has decided on a particular value. A *decision pair* is a pair $(\mathcal{Z}, \mathcal{O})$ of decision sets. Intuitively, $\mathcal{Z}$ (for *zero*) describes the local states of processors at points where they are deciding or have decided on 0, while $\mathcal{O}$ (for *one*) describes the analogous states for a decision of 1. We say that $(\mathcal{Z}, \mathcal{O})$ *is the decision pair for protocol P* if $\mathcal{Z}_i$ consists of the local states at which processor i decides or has decided 0 when executing the protocol P, and $\mathcal{O}_i$ consists of the local states at which processor i decides or has decided 1. Typically, we shall describe a decision set such as $\mathcal{Z}_i$ or $\mathcal{O}_i$ by a formula of the form $K_i\varphi$ or $B_i^{\mathcal{N}}\psi$. Given a system, the decision set is then the set of local states in the system at which the formula is satisfied. For example, the decision set defined by $K_i\varphi$ is $\{r_i(m) : (\mathcal{R}, r, m) \models K_i\varphi\}$; similarly, the set defined by $B_i^{\mathcal{N}}\psi$ is $\{r_i(m) : (\mathcal{R}, r, m) \models B_i^{\mathcal{N}}\psi\}$.

Recall that $\mathcal{N}$ denotes the set of nonfaulty processors. Given a decision set $\mathcal{A}$, let $\mathcal{N} \wedge \mathcal{A}$ denote the nonrigid set described by

$$(\mathcal{N} \wedge \mathcal{A})(r, m) = \{i : i \in \mathcal{N}(r, m) \text{ and } r_i(m) \in \mathcal{A}_i\}.$$

With this machinery, we can now give a necessary condition for agreement.

PROPOSITION 4.3. *Let P be an agreement protocol with decision pair $(\mathcal{Z}, \mathcal{O})$. Then*

(a) $\mathcal{R}_P \models \mathit{decide}_i(0) \Rightarrow B_i^{\mathcal{N}}(\exists 0 \wedge \neg \mathit{decide}_i(1) \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} \exists 0)$;
(b) $\mathcal{R}_P \models \mathit{decide}_i(1) \Rightarrow B_i^{\mathcal{N}}(\exists 1 \wedge \neg \mathit{decide}_i(0) \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}} \exists 1)$.

Proof. We prove part (a); the proof of (b) is completely symmetric. The argument is split into showing that $\mathit{decide}_i(0)$ implies belief in each of the three conjuncts. Suppose that $(\mathcal{R}_P, r, m) \models \mathit{decide}_i(0)$. By Proposition 4.1(b) we have that $(\mathcal{R}_P, r, m) \models K_i \mathit{decide}_i(0)$. From Proposition 3.2(f) we obtain that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}} \mathit{decide}_i(0)$. The weak validity property for P implies that $\mathcal{R}_P \models (i \in \mathcal{N}) \Rightarrow (\mathit{decide}_i(0) \Rightarrow \mathit{exist}(0))$. By Proposition 3.2(a) we have that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}}(\mathit{decide}_i(0)) \Rightarrow \mathit{exist}(0))$, and it follows by the distribution axiom for $B_i^{\mathcal{N}}$ (Proposition 3.2(b)) that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}} \mathit{exist}(0)$. Similarly, we explicitly assumed that a processor cannot at once decide on two values, and hence $\mathcal{R}_P \models \mathit{decide}_i(0) \Rightarrow \neg \mathit{decide}_i(1)$. As before, this implies that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}}(\mathit{decide}_i(0) \Rightarrow \neg \mathit{decide}_i(1))$, and the distribution axiom yields belief in the second conjunct: $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}} \neg \mathit{decide}_i(1)$. It remains to show that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}} C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} \exists 0$. Let (r', m) be a point such that $r'_i(m) = r_i(m)$ and $i \in \mathcal{N}(r', m)$. Since $r'_i(m) = r_i(m)$ we have by Proposition 4.1 that $(\mathcal{R}_P, r', m) \models \mathit{decide}_i(0)$. Lemma 4.2 implies that $(\mathcal{R}_P, r', m) \models \boxdot \neg \mathit{decide}_j(1)$ for every nonfaulty processor j. Thus, for all $m' \geq 0$, we have that $(\mathcal{N} \wedge \mathcal{O})(r', m') = \emptyset$. By definition of

continual common knowledge, this immediately implies that $(\mathcal{R}_P, r', m) \models C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$. It follows by definition of $B_i^{\mathcal{N}}$ that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}} C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$ as desired, and we are done. □

Proposition 4.3 shows that in order to decide 0, processor i must believe, not surprisingly, that some initial value was 0 and that it does not decide 1. More interesting is the fact that it also must believe that it is continual common knowledge among the nonfaulty processors that eventually decide 1 that there is an initial value of 0. A moment's thought will show that this is true simply because, in an agreement protocol, if a nonfaulty processor decides 0 in run r it believes that there are no nonfaulty processors that decide 1, so $(\mathcal{N}\wedge\mathcal{O})(r,m) = \emptyset$ for all m. Nevertheless, this characterization is of interest because for optimal protocols this condition will turn out to be necessary and sufficient. We now show that a closely related condition is sufficient for agreement in general.

PROPOSITION 4.4. *Let P be a protocol with decision pair $(\mathcal{Z}, \mathcal{O})$ such that either*

(a) $\mathcal{R}_P \models \mathit{decide}_i(0) \Rightarrow B_i^{\mathcal{N}}\exists 0$ *and*

(b) $\mathcal{R}_P \models \mathit{decide}_i(1) \Leftrightarrow B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1)$

or

(a′) $\mathcal{R}_P \models \mathit{decide}_i(0) \Leftrightarrow B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$ *and*

(b′) $\mathcal{R}_P \models \mathit{decide}_i(1) \Rightarrow B_i^{\mathcal{N}}\exists 1$.

Then P is an agreement protocol.

Proof. We prove the first claim regarding the pair (a) and (b), as the other part is again completely symmetric. Assume that P satisfies (a) and (b). We first show that P satisfies weak validity. Let i be a nonfaulty processor in the run r. We claim that i can decide only on a value that appeared as one of the initial values. Assume that i decides 0 in r. Part (a) implies that i believes $\exists 0$ at that point. By Proposition 3.2(c) we have that $\exists 0$ indeed holds at that point, which implies that some initial value was 0. Similarly, if i decides 1, then part (b) implies that i believes 1 and similarly at least one initial value was 1. It remains to show weak consistency. Assume by way of contradiction that in a run r of $\mathcal{R}_P$ nonfaulty processor i decides 0 at (r,m), while nonfaulty processor j decides 1 at (r,m'). Part (b) implies that $(\mathcal{R}_P, r, m') \models (j \in \mathcal{N}) \wedge B_j^{\mathcal{N}} C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1$. Since the belief of a nonfaulty processor is guaranteed to be true we have that $(\mathcal{R}_P, r, m') \models C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1$. By Lemma 3.5(g) we have that $(\mathcal{R}_P, r, m) \models C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1$ holds as well. From the fact that i decides 0 at (r,m) it follows that $i \in \mathcal{N}\wedge\mathcal{Z}(r,m)$. The fixed-point axiom for continual common knowledge (Lemma 3.5(e)) implies that $(\mathcal{R}_P, r, m) \models B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1)$. We can now apply property (b) to i at (r,m) and conclude that i decides 1 at that point, contradicting the assumption that i decides 0 there. □

We remark that the asymmetry between the decision sets for 0 and 1 in Proposition 4.4 is essential. While the conditions in Propositions 4.3 and 4.4 are not identical, once we move to *optimal* protocols, we will be able to get a single necessary and sufficient condition, expressed in terms of continual common knowledge, that characterizes optimality.

5. Constructing and characterizing optimal agreement. It is not too hard to show that for optimal agreement protocols the implications ("⇒") in parts (a) and (b) of Proposition 4.3 become equivalences ("⇔"). More importantly, we can show that if "sufficient information" is transmitted, as is the case in a full-information protocol, then a protocol satisfying these conditions is optimal. These points are made precise and proved in this section. In the process, we also show how to construct an optimal protocol dominating any given protocol.

From here on in, we focus on full-information protocols. As we indicated in section 2.4, full-information protocols are all we need to consider if we are interested in optimal protocols for EBA. We use $FIP(\mathcal{Z},\mathcal{O})$ to denote the (unique) full-information protocol with decision pair $(\mathcal{Z},\mathcal{O})$. We will be focusing on protocols of the form $FIP(\mathcal{Z},\mathcal{O})$ in the rest of our analysis. Notice that for any pair of protocols of this form, there is a one-to-one and onto mapping from the runs of the first protocol to corresponding runs of the second protocol. The only possible difference between corresponding runs is in what decisions are made and when they are taken; the same messages are sent at the same times, and the same processors fail in the same way and at the same time. As a result, we can assume without loss of generality that all decision sets $\mathcal{A}$ that we consider are defined in terms of the initial configuration and the pattern of messages that are communicated. More formally, we assume that all sets $\mathcal{A}$ that we consider for decision sets $\mathcal{Z}$ and $\mathcal{O}$ have the property that if F and F' are full-information protocols and r and r' are corresponding runs in $\mathcal{R}_F$ and $\mathcal{R}_{F'}$, respectively, then $r_i(m) \in \mathcal{A}$ iff $r'_i(m) \in \mathcal{A}$. We use this observation in our construction of optimal EBA protocols. During the construction, we start with one full-information protocol and then modify its decision pair to get another full-information protocol. After a finite number of modifications, we end up with an optimal protocol.

When we represent full-information protocols in this way, the sets determining the decision pair can be specified in a fairly flexible manner. For example, we will often use knowledge formulas to describe these sets. As long as these formulas talk about initial values (and not decisions), they will be uniquely determined by the communication that takes place in the run and thus result in a well-defined protocol. For example, a formula such as $B_i^{\mathcal{N}}\exists 0$ uniquely specifies a set of local states for processor i in runs of full-information protocols of this form. It can therefore be used to specify a decision set in a decision pair.

The following result is the core of our technique for constructing optimal protocols. It provides a knowledge-theoretic method for constructing protocols that dominate a given (full-information) agreement protocol and plays a key role in our characterization of optimal agreement protocols. It can be viewed as a formalization of some of the intuitions presented in section 3.2.

PROPOSITION 5.1. *Let $F = FIP(\mathcal{Z},\mathcal{O})$ be a full-information agreement protocol, and define $F' = FIP(\mathcal{Z}',\mathcal{O}')$ and $F'' = FIP(\mathcal{Z}'',\mathcal{O}'')$, where*

$$\mathcal{Z}'_i = B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0) \quad \text{and} \quad \mathcal{O}'_i = B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0),$$

while

$$\mathcal{O}''_i = B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1) \quad \text{and} \quad \mathcal{Z}''_i = B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1).$$

Then F' and F'' are both agreement protocols that dominate F.

Proof. The case of $F'' = FIP(\mathcal{Z}'',\mathcal{O}'')$ is symmetric to that of $F' = FIP(\mathcal{Z}',\mathcal{O}')$; we prove for F'.

First note that $\models C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \Leftrightarrow \boxdot C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$, and hence $C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$ is a property of runs. The property $\mathcal{Z}'_i$ can hold at a point in a run only if the run satisfies $C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$, while $\mathcal{O}'_i$ can hold only if $\neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$ does. It follows that nonfaulty processors cannot reach different decisions in the same run of F', and so weak agreement holds. In addition, since $\models \mathcal{Z}'_i \Rightarrow \exists 0$ and $\models \mathcal{O}'_i \Rightarrow \exists 1$, weak validity is guaranteed. It follows that F' is an agreement protocol. It remains to show that F' dominates F.

Let r be a run of $\mathcal{R}_F$, and let r' be the corresponding run in $\mathcal{R}_{F'}$. First, suppose that $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(1) \wedge (i \in \mathcal{N})$. We want to show that processor i decides by time m in r'. If i decides in r' before time m we are done. Assume not. Clearly, either (a) $(\mathcal{R}_F, r, m) \models C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$ or (b) $(\mathcal{R}_F, r, m) \models \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$. If (a) holds, then $r_i(m) \in (\mathcal{N} \wedge \mathcal{O})$ and by Lemma 3.5(e) we have that

$$(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0).$$

Thus, by definition of $\mathcal{Z}'$,we have that $r_i(m) \in \mathcal{Z}'_i$. If (b) holds, a similar argument shows that $(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}\neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0$. In addition, since F is an agreement protocol, it follows from the fact that i decides 1 at (r, m) that $(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}\exists 1$. Therefore, $(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$, and so $r_i(m) \in \mathcal{O}'_i$. Thus, $r'_i(m) \in (\mathcal{Z}'_i \cup \mathcal{O}'_i)$ so that in either case processor i decides at $r'_i(m)$.

Now suppose that $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(0) \wedge (i \in \mathcal{N})$. Then by Proposition 4.3, we also have $(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$. Therefore, by definition, $r_i(m) \in \mathcal{Z}'_i$, and again it follows that processor i decides at least as soon in r' as in r. □

Proposition 5.1 suggests a way to construct an optimal agreement protocol. Namely, start with a full-information agreement protocol $F = \mathit{FIP}(\mathcal{Z}, \mathcal{O})$. Then construct a new protocol $F^1 = \mathit{FIP}(\mathcal{Z}^1, \mathcal{O}^1)$, where

$$\begin{aligned}\mathcal{Z}_i^1 &= B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0) \text{ and}\\ \mathcal{O}_i^1 &= B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0).\end{aligned}$$

Notice that the new protocol is completely determined by the formula decision set $\mathcal{O}$ of the original protocol. We then proceed to create a second protocol $F^2 = \mathit{FIP}(\mathcal{Z}^2, \mathcal{O}^2)$, where

$$\begin{aligned}\mathcal{Z}_i^2 &= B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}^1}\exists 1) \text{ and}\\ \mathcal{O}_i^2 &= B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}^1}\exists 1).\end{aligned}$$

Protocol F^2 is thus completely determined by $\mathcal{Z}^1$. We can, of course, continue in this fashion and define protocols $F^{2,1}, F^{2,2}, \ldots$. By Proposition 5.1, each new protocol we construct dominates the previous ones. By the observations in section 2.2, if we start with an EBA protocol, this process eventually terminates, giving us an optimal EBA protocol. However, it is far from clear how long it takes this process to terminate (or if it even terminates at all if we start with an arbitrary agreement protocol). We now show that, in fact, the process always terminates in two steps, so the resulting protocol F^2 is indeed optimal. (Of course we can also show that the analogous construction, exchanging the roles of $\mathcal{Z}$ and $\mathcal{O}$, results in an optimal protocol.)

THEOREM 5.2. *Let $F = \mathit{FIP}(\mathcal{Z}, \mathcal{O})$ be an agreement protocol, and let F^1 and F^2 be as described above. Then F^2 is an optimal agreement protocol. Moreover, if F is an EBA protocol, then F^2 is an optimal EBA protocol dominating F.*

Proof. Suppose that F^2 is not optimal and that F^3 dominates F^2. Proposition 5.1 implies that F^2 dominates both F and F^1. Therefore, F^3 must also dominate both F and F^1. By Lemma 5.3, if processor i decides 0 at the point (r', m) of $\mathcal{R}_{F^3}$, it must be the case that $(\mathcal{R}_{F^3}, r', m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$. Thus, by definition of $\mathcal{Z}_i^1$, processor i decides at the corresponding point of $\mathcal{R}_{F^1}$ (if it has not done so earlier in the run). Since F^2 dominates F^1, it must be the case that processor i has also decided by the corresponding point of $\mathcal{R}_{F^2}$. Similarly, since F^3 dominates F^1, if processor i decides 1 at the point (r', m), then $(\mathcal{R}_{F^3}, r', m) \models B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}^1}\exists 1)$. By definition of $\mathcal{O}_i^2$, it must be the case that processor i has decided by the corresponding point

in $\mathcal{R}_{F^2}$. We conclude that F^2 dominates F^3. Since F^3 was chosen as an arbitrary protocol dominating F^2, it follows that F^2 is indeed optimal.

Notice that, in general, F^2 is not necessarily an EBA protocol because not all the nonfaulty processors necessarily decide. However, if the protocol F we started out with is an EBA protocol, then every protocol dominating F must satisfy the decision condition of EBA. In this case, then, F^2 will in fact be an optimal EBA protocol dominating F. □

To complete the proof of Theorem 5.2, we need a technical lemma that relates the stopping conditions of two protocols F and F', where F' dominates F. It describes the state of knowledge that must hold in a run of F' in which the decision value is different from that of the corresponding run of F.

LEMMA 5.3. *Let $F = FIP(\mathcal{Z}, \mathcal{O})$ and $F' = FIP(\mathcal{Z}', \mathcal{O}')$ be full-information agreement protocols such that F' dominates F. Then we must have*

(a) $\mathcal{R}_{F'} \models decide_i(0) \Rightarrow B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} \exists 0)$;

(b) $\mathcal{R}_{F'} \models decide_i(1) \Rightarrow B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}} \exists 1)$.

Proof. The proofs of (a) and (b) are identical, so we prove only (a). The fact that $\mathcal{R}_F \models decide_i(0) \Rightarrow B_i^{\mathcal{N}} \exists 0$ follows from the fact that F' is an agreement protocol. Therefore, it suffices to show that $\mathcal{R}_{F'} \models decide_i(0) \Rightarrow B_i^{\mathcal{N}} C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}}(\exists 0)$. Proposition 4.3 implies that $\mathcal{R}_{F'} \models decide_i(0) \Rightarrow B_i^{\mathcal{N}} C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}'}(\exists 0)$. We need to show that the same result holds with $\mathcal{O}'$ replaced by $\mathcal{O}$.

Let $decide(0)$ denote the basic fact that is true at a point (r, m) in $\mathcal{R}_{F'}$ exactly if some nonfaulty processor decides 0 at some point in r. It is clearly the case that $\mathcal{R}_{F'} \models decide_i(0) \Rightarrow B_i^{\mathcal{N}} decide(0)$, since if $decide_i(0)$ holds, then i knows $decide_i(0)$, and hence in particular i knows that if i is nonfaulty, then $decide(0)$ holds. To complete the proof, it remains to prove the following.

Claim. $\mathcal{R}_{F'} \models decide(0) \Rightarrow C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} \exists 0$.

Proof. Since F' satisfies weak validity, it is easy to check that $\mathcal{R}_{F'} \models decide(0) \Rightarrow \exists 0$. It therefore suffices to prove that $\mathcal{R}_{F'} \models decide(0) \Rightarrow C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} decide(0)$. Using the induction rule (part (f) of Lemma 3.5), it suffices to show that

$$\mathcal{R}_{F'} \models decide(0) \Rightarrow E^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}} decide(0).$$

To see this, assume that $(\mathcal{R}_{F'}, r', m) \models decide(0)$ and $j \in (\mathcal{N} \wedge \mathcal{O})(r', m')$ for some point (r', m') in the run r'. We need to show that $(\mathcal{R}_{F'}, r', m') \models B_j^{\mathcal{N}} decide(0)$. Let r be the run of $\mathcal{R}_F$ that corresponds to r'. Since $F = FIP(\mathcal{Z}, \mathcal{O})$, if processor j decides in run r at all, it does so at the first time m such that $r_j(m) \in \mathcal{O}_j$. Since $j \in (\mathcal{N} \wedge \mathcal{O})(r', m')$, it follows that $r'_j(m') = r_j(m') \in \mathcal{O}_j$. Thus, processor j decides no later than at time m' in run r. Since F' dominates F, processor j must decide no later than at time m' in run r' either. Since j is nonfaulty in r', and the nonfaulty processors decide 0 in r', it follows that j decides 0 in r' at some time $m'' \leq m'$. Thus, we have $(\mathcal{R}_{F'}, r', m'') \models decide_j(0)$, from which it follows that $(\mathcal{R}_{F'}, r', m'') \models B_j^{\mathcal{N}} decide(0)$. Finally, since in a full-information protocol processors keep track of their history in their local state, it follows that once processor j believes that $decide(0)$ holds, it will continue to believe this at all points in the future. (Technically, the property of *no forgetting* or *perfect recall* holds in full-information protocols [10].) Thus, $(\mathcal{R}_{F'}, r', m') \models B_j^{\mathcal{N}} decide(0)$, as desired. □

To complete the section, we now show that, as claimed earlier, the necessary conditions of Proposition 4.3 actually characterize optimal (full-information) agreement protocols.

THEOREM 5.4. *Let $F = FIP(\mathcal{Z}, \mathcal{O})$ be a full-information agreement protocol. Then F is optimal iff both of the following conditions hold:*

(a) $\mathcal{R}_F \models i \in \mathcal{N} \Rightarrow (\mathit{decide}_i(0) \Leftrightarrow B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1)))$,
(b) $\mathcal{R}_F \models i \in \mathcal{N} \Rightarrow (\mathit{decide}_i(1) \Leftrightarrow B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{Z}}\exists 1 \wedge \neg \mathit{decide}_i(0)))$.

Proof. For the "only if" direction, we prove only (a) here; the proof of (b) is analogous. Since F is an agreement protocol, it follows from Proposition 4.3 that

$$\mathcal{R}_F \models \mathit{decide}_i(0) \Rightarrow B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1)).$$

Hence it is enough to prove that

$$\mathcal{R}_F \models i \in \mathcal{N} \Rightarrow (B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1)) \Rightarrow \mathit{decide}_i(0)).$$

Suppose that

$$(\mathcal{R}_F, r, m) \models i \in \mathcal{N} \wedge B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1)).$$

Let $\mathcal{Z}_i' = B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$ and $\mathcal{O}_i' = B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0)$, and consider the protocol F' defined by $F' = \mathit{FIP}(\mathcal{Z}', \mathcal{O}')$. Proposition 5.1 shows that F' is an agreement protocol that dominates F. Therefore, for F to be optimal, it must dominate F' as well.

Let (r', m) be the point of $\mathcal{R}_{F'}$ corresponding to (r, m). Since $(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1))$, we have that $r_i(m) \in \mathcal{Z}_i'$. From $r_i(m) = r_i'(m)$, it follows that $r_i'(m) \in \mathcal{Z}_i'$ so that processor i decides or has decided by (r', m) in protocol F'. Since F is optimal by assumption, we have that processor i must also decide by (r, m). Thus, $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(0) \vee \mathit{decide}_i(1)$. However, since $(\mathcal{R}_F, r, m) \models i \in \mathcal{N} \wedge B_i^{\mathcal{N}}\neg \mathit{decide}_i(1)$, it follows by Proposition 4.1(c) that $(\mathcal{R}_F, r, m) \models \neg \mathit{decide}_i(1)$, and hence $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(0)$ as desired.

For the "if" direction, suppose that both (a) and (b) hold. Proposition 2.3 implies that it is enough to show that F dominates any full-information agreement protocol F' that dominates it. Therefore, suppose F' dominates F. Let r' be a run of $\mathcal{R}_{F'}$, and let r be the corresponding run in $\mathcal{R}_F$. First, suppose that $(\mathcal{R}_{F'}, r', m) \models i \in \mathcal{N} \wedge \mathit{decide}_i(0)$. We want to show that processor i decides by time m in r. The proof that the same happens if $\mathit{decide}_i(1)$ holds is identical and hence is omitted. By Lemma 5.3, we have

$$(\mathcal{R}_{F'}, r', m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0).$$

Since (r, m) and (r', m) are corresponding points of $\mathcal{R}_F$ and $\mathcal{R}_{F'}$, we have that

$$(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0).$$

If, in addition, $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(1)$, then processor i decides by time m in run r. On the other hand, if $(\mathcal{R}_F, r, m) \models \neg \mathit{decide}_i(1)$, then Proposition 4.1 implies that $\mathcal{R}_F \models \neg \mathit{decide}_i(1) \Rightarrow B_i^{\mathcal{N}}\neg \mathit{decide}_i(1)$, so

$$(\mathcal{R}_F, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N}\wedge\mathcal{O}}\exists 0 \wedge \neg \mathit{decide}_i(1)).$$

From assumption (a), it follows that $(\mathcal{R}_F, r, m) \models \mathit{decide}_i(0)$, and thus we again get that processor i decides by time m in run r. □

6. Examples of optimal protocols. In this section, we apply the technique of Theorem 5.2 to construct some optimal agreement protocols. We start with an agreement protocol $F(\mathcal{Z}, \mathcal{O})$ and proceed as described in section 5. Though each protocol constructed in this way is optimal, different choices of $\mathcal{Z}$ and $\mathcal{O}$ yield optimal

protocols with different properties and different performance. Moreover, even if we start with the same basic protocol, the protocol we end up with depends on the failure mode considered. This stems from a subtle point that has not arisen in our treatment so far. Up until now, we have ignored the issue of whether we are dealing with crash failures or omission failures. Notice, however, that the sets $\mathcal{Z}_i^1$, $\mathcal{O}_i^1$, $\mathcal{Z}_i^2$, and $\mathcal{O}_i^2$ are defined in terms of what processors know at certain points. Of course, what they know at a given point will depend in part on whether we are dealing with crash failures or omission failures. Suppose, for example, we start with an agreement protocol F that works in the case of omission failure. Then it clearly also works in the case of crash failures. However, $\mathcal{R}_F^{cr}(n,t)$ and $\mathcal{R}_F^{om}(n,t)$ are different systems. Thus, we have different decision pairs $(\mathcal{Z}^{cr}, \mathcal{O}^{cr})$ and $(\mathcal{Z}^{om}, \mathcal{O}^{om})$, depending on whether we consider crash failures or omission failures. Because there are more runs in $\mathcal{R}_F^{om}$ than $\mathcal{R}_F^{cr}$, it follows that $\mathcal{Z}_i^{cr} \subseteq \mathcal{Z}_i^{om}$ and $\mathcal{O}_i^{cr} \subseteq \mathcal{O}_i^{om}$. However, once we apply the procedure of Proposition 5.1 to these sets, we may no longer have, for example, $(\mathcal{Z}_i')^{cr} \subseteq (\mathcal{Z}_i^1)^{om}$. Nevertheless, all the results from section 5 apply whether we are working with crash failures or omission failures, so we continue to suppress the failure mode here. However, as we shall see, the issue of the failure mode plays a major role in starting with the same protocol; our techniques lead to different protocols depending on the type of failures considered. Since many of the proofs in this section are lengthy and provide less insight than the ones in the previous sections, we defer them to the appendix.

6.1. A simple optimal protocol. A particularly trivial agreement protocol is one in which no processor ever decides. Let F^Λ be the full-information protocol in which no processor ever decides. That is, we define $\mathcal{Z}_i^\Lambda = \mathcal{O}_i^\Lambda = \emptyset$ for $i = 1, \ldots, n$, and let $F^\Lambda = FIP(\mathcal{Z}^\Lambda, \mathcal{O}^\Lambda)$. Suppose we apply our optimization technique to F^Λ.

The first step of the construction consists of having the processors decide 0 as soon as possible, given the criterion for deciding 1. Thus,

$$\mathcal{Z}_i^{\Lambda,1} = B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}^\Lambda} \exists 0) \text{ and}$$
$$\mathcal{O}_i^{\Lambda,1} = B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}^\Lambda} \exists 0).$$

Since in F^Λ nonfaulty processors never decide 1, we have that $\mathcal{R}_{F^\Lambda} \models \boxdot(\mathcal{N} \wedge \mathcal{O}^\Lambda = \emptyset)$, and hence also $\mathcal{R}_{F^\Lambda} \models C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}^\Lambda} \exists 0$. It follows that $\mathcal{Z}_i^{\Lambda,1} = B_i^{\mathcal{N}} \exists 0$ and $\mathcal{O}_i^{\Lambda,1} = B_i^{\mathcal{N}}(\exists 1 \wedge \mathit{false})$, which is equivalent to $B_i^{\mathcal{N}} \mathit{false}$.

The second step of the construction optimizes the decision on 1, given the definition of $\mathcal{Z}^{\Lambda,1}$. Following Proposition 5.1, we define

$$\mathcal{Z}_i^{\Lambda,2} = B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1) \text{ and}$$
$$\mathcal{O}_i^{\Lambda,2} = B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1).$$

We denote $FIP(\mathcal{Z}^{\Lambda,2}, \mathcal{O}^{\Lambda,2})$ by $F^{\Lambda,2}$.

The analysis we have performed so far applies equally well to both the crash and the omission failure mode. Thus, $F^{\Lambda,2}$ is an optimal agreement protocol in both cases. As we shall see, however, while F^Λ behaves in essentially the same way in both settings, the properties of $F^{\Lambda,2}$ are dramatically different in the two failure modes. The crux of the analysis will involve figuring out when $\mathcal{Z}^{\Lambda,2}$ and $\mathcal{O}^{\Lambda,2}$ hold in runs of $F^{\Lambda,2}$. This, in turn, will be determined by when $C^{\boxdot}_{\mathcal{N} \wedge B^{\mathcal{N}} \exists 0} \exists 1$ holds.

We begin with an analysis of $F^{\Lambda,2}$ in the crash failure mode. As the next theorem shows, $F^{\Lambda,2}$ is quite simple in this case.

THEOREM 6.1. *In the crash failure mode,* $F^{\Lambda,2} = FIP(\mathcal{Z}^{cr}, \mathcal{O}^{cr})$, *where* $\mathcal{Z}_i^{cr} = B_i^{\mathcal{N}} \exists 0$ *and* $\mathcal{O}_i^{cr} = B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{cr}) = \emptyset)$.

Proof. The proof can be found in the appendix. □

It is easy for processor i to compute when it should decide 0 according to $F^{\Lambda,2}$ in the crash failure mode: it decides 0 if it ever hears about a 0 from any processor. It also turns out to be easy for processor i to compute when to decide 1. According to Theorem 6.1, it should decide 1 when it believes that $\mathcal{N} \wedge \mathcal{Z}^{cr} = \emptyset$, that is, when it believes that no nonfaulty processor knows that some processor started with 0. The relevant conditions for this to be the case are precisely those given for the protocol $P0_{\text{opt}}$ described in the introduction. As we prove in Theorem 6.2, processor i decides 1 according to the protocol when it either knows that all initial values are 1 or it hears from the same set of processors in two consecutive rounds and still does not know that some processor had an initial value of 0. Thus, $P0_{\text{opt}}$ and $F^{\Lambda,2}$ are essentially equivalent protocols. The same decisions are made by all processors at corresponding points of the systems generated by the two protocols in the crash failure mode. (The two protocols are not identical because $F^{\Lambda,2}$ is a full-information protocol, while in $P0_{\text{opt}}$ the processors send much shorter messages.) It follows from this that not only is $F^{\Lambda,2}$ an optimal agreement protocol (this already follows from Theorem 5.2), but it is also an optimal EBA protocol.

THEOREM 6.2. *In the crash failure mode, the same decisions are made by nonfaulty processors at corresponding points of the protocols* $P0_{\text{opt}}$ *and* $F^{\Lambda,2}$. *Thus, both* $F^{\Lambda,2}$ *and* $P0_{\text{opt}}$ *are optimal EBA protocols for the crash failure mode.*

Proof. The proof can be found in the appendix. □

Since $P0_{\text{opt}}$ can be implemented using messages of linear size, $F^{\Lambda,2}$ gives us an efficient optimal EBA protocol in the crash failure mode. The situation is very different in the case of omission failures. While $F^{\Lambda,2}$ is still an optimal agreement protocol, and an analysis similar to that carried out for the case of crash failures can be used to show that it has an efficient implementation, it is no longer an EBA protocol. There are runs in which it never halts.

PROPOSITION 6.3. *If* $t > 1$ *and* $n \geq t + 2$, *then there are runs of* $F^{\Lambda,2}$ *in* $\mathcal{R}^{om}$ *in which the nonfaulty processors do not decide.*

Proof. The proof can be found in the appendix. □

6.2. Optimal EBA for omission failures. As the proof of Proposition 6.3 shows, the reason that $F^{\Lambda,2}$ does not necessarily terminate in the presence of omission failures is that there is no bound on the time at which a processor can learn that there exists an initial value of 0. Clearly, if we start out with a terminating protocol and find an optimal protocol that dominates it, then the optimal protocol we obtain will also be terminating. We now use a well-known approach to generate a (terminating) EBA protocol in the omission failure mode. Intuitively, we say that a processor *accepts* 0 in round m only if this value was transferred by a chain of $m - 1$ distinct processors (cf. [4]). Formally, we say that a 0-*chain* exists at the point (r, m) full-information F iff there is a sequence of m distinct processors $i_1, \ldots, i_m$, such that i_1 has initial value 0, i_{k+1} received a message from i_k at round k and $(\mathcal{R}_F, r, k) \models \neg B_{i_{k+1}}^{\mathcal{N}}(i_k \notin \mathcal{N})$, and i_m is nonfaulty. We say that $(\mathcal{R}_F, r, m) \models \exists 0^*$ if there is a 0-chain at some point (r, m') with $m' \leq m$.

Let $\mathcal{Z}_i^0 = B_i^{\mathcal{N}} \exists 0^*$ and $\mathcal{O}_i^0 = B_i^{\mathcal{N}} \neg \exists 0^*$, and consider the protocol $FIP(\mathcal{Z}^0, \mathcal{O}^0)$. It is easy to see that this is an agreement protocol. The following proposition shows that it is actually an EBA protocol.

PROPOSITION 6.4. *In a run* r *of* $FIP(\mathcal{Z}^0, \mathcal{O}^0)$ *in the omission failure mode where*

f processors actually fail, all nonfaulty processors decide by time $f+1$.

Proof. The proof can be found in the appendix. □

Clearly $FIP(\mathcal{Z}^0, \mathcal{O}^0)$ satisfies the validity and consistency conditions, so we immediately get the following corollary.

COROLLARY 6.5. *$FIP(\mathcal{Z}^0, \mathcal{O}^0)$ is an EBA protocol.*

We can now apply the construction of Theorem 5.2 to $FIP(\mathcal{Z}^0, \mathcal{O}^0)$ to get an optimal EBA protocol that dominates it. The description of the protocol that we get using the construction can be simplified somewhat. Let $\mathcal{Z}^* = B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}^0} \exists 0)$, $\mathcal{O}^* = B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{O}^0} \exists 0)$, and $F^* = FIP(\mathcal{Z}^*, \mathcal{O}^*)$.

PROPOSITION 6.6. *F^* is an optimal EBA in the omission failure mode that dominates $FIP(\mathcal{Z}^0, \mathcal{O}^0)$.*

Proof. The proof can be found in the appendix. □

Note that the decision rules in F^* involve explicit tests for knowledge. Although it is not hard to show that these tests can be implemented effectively (by which we mean that the tests are decidable in principle; in fact, this can be done in PSPACE [5]), we do not know if they can be implemented efficiently, even in the case of crash failures. The question of whether there exists a polynomial-time protocol that is optimal for EBA in the case of omission failures is still open; we conjecture that such a protocol does exist. (We remark that in [5, 14], polynomial time protocols that are optimum for SBA in the case of crash and omission failures are given.)

7. Conclusions. This paper completely characterizes optimal EBA protocols in the case of crash and omission failures. It involves a new variant of common knowledge—*continual common knowledge*—in an essential way. Interestingly, continual common knowledge is a stronger state of knowledge than common knowledge. By contrast, all other variants of common knowledge considered in the literature (cf. [8, 16]) are weakenings of common knowledge. The characterization is far from trivial. While it would in principle be possible to characterize and prove these properties without using knowledge, we conjecture that it would be rather difficult.

Our results can be extended in a number of ways. First, although we have considered here only crash and omission failures, we believe that our results should be extendible to the case of Byzantine failures, once a reasonable notion of corresponding runs and protocols is given for that model. Second, although we assumed here that the processors are synchronous, our knowledge analysis is also valid for asynchronous systems (except for the implementations in section 6, of course). Third, although this paper focuses on EBA and agreement protocols, it is straightforward to extend our results to general coordination problems along the lines of [14], including ones in which *all* processors (and not only the nonfaulty ones) are required to act consistently. (Problems of the latter type are said to require *uniform agreement*.) Neiger and Bazzi [15] consider these issues, using an approach based on ours. They consider agreement in synchronous and asynchronous systems, and study both uniform and nonuniform agreement. In addition, they look at cases where explicit termination, as well as agreement, is required. To handle termination and agreement, they define a notion of *extended knowledge*, which is essentially a combination of continual common knowledge and eventual common knowledge.

An interesting question left open is whether our two-step optimizing process can be done in a computationally efficient manner. Our examples in section 6 show that in some cases we can obtain efficient optimal EBA protocols. However, a naive evaluation of the formulas used in the optimization process does not guarantee computationally efficient results.

Appendix. Proofs. In this appendix, we prove the results in section 6. For the convenience of the reader, we repeat the statements of the results.

THEOREM 6.1. *In the crash failure mode,* $F^{\Lambda,2} = FIP(\mathcal{Z}^{cr}, \mathcal{O}^{cr})$, *where* $\mathcal{Z}_i^{cr} = B_i^{\mathcal{N}}\exists 0$ *and* $\mathcal{O}_i^{cr} = B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{cr}) = \emptyset)$.

Theorem 6.1 follows immediately from the next two lemmas. The first shows that $\mathcal{Z}_i^{\Lambda,2} = B_i^{\mathcal{N}}\exists 0$, and the second shows that $\mathcal{O}_i^{\Lambda,2} = B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{cr}) = \emptyset)$. This shows that $\mathcal{Z}^{\Lambda,2} = \mathcal{Z}^{cr}$ and $\mathcal{O}^{\Lambda,2} = \mathcal{O}^{cr}$, proving the theorem. To simplify notation in the proofs, we denote the system $\mathcal{R}^{cr}_{F^{\Lambda,2}}$ obtained by running protocol $F^{\Lambda,2}$ in the crash failure mode by $\mathcal{R}^{cr}$.

LEMMA A.1. *In* $\mathcal{R}^{cr}$, $\mathcal{Z}_i^{\Lambda,2} = B_i^{\mathcal{N}}\exists 0$.

Proof. Recall that $\mathcal{Z}_i^{\Lambda,2} = B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1)$ and that $\mathcal{Z}_i^{\Lambda,1} = B_i^{\mathcal{N}}\exists 0$. Clearly $\mathcal{Z}_i^{\Lambda,2} \subseteq B_i^{\mathcal{N}}\exists 0$. For the opposite containment, suppose $(\mathcal{R}^{cr}, r, m) \models B_i^{\mathcal{N}}\exists 0$. We want to show that $(\mathcal{R}^{cr}, r, m) \models B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1)$. Therefore, suppose that $r_i(m) = r'_i(m)$ and i is nonfaulty in r'. We must show that $(\mathcal{R}^{cr}, r', m) \models \neg C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1$. Since $(\mathcal{R}^{cr}, r', m) \models i \in \mathcal{N} \wedge B_i^{\mathcal{N}}\exists 0$, it certainly suffices to show that

$$\mathcal{R}^{cr} \models (i \in \mathcal{N} \wedge B_i^{\mathcal{N}}\exists 0) \Rightarrow \neg C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1.$$

To show this, we prove by induction on k that for all runs r' of $\mathcal{R}^{cr}$ and processors i that are nonfaulty in r', if $(\mathcal{R}^{cr}, r', k) \models B_i^{\mathcal{N}}\exists 0$, then there is a point $(\hat{r}, k)$ that is $(\mathcal{N} \wedge B^{\mathcal{N}}\exists 0)$-$\boxdot$-reachable from (r', k) such that $(\mathcal{R}^{cr}, \hat{r}, k) \models \neg\exists 1$.

Suppose that $k = 0$. Since i does not believe it is faulty, $B_i^{\mathcal{N}}\exists 0$ holds only if i's initial value is 0. Let $\hat{r}$ be the run where all processors have initial values 0 and no processor fails. Clearly, $(\hat{r}, 0)$ has the desired properties.

For the inductive step, let $k > 0$ and assume that the claim has been shown for all $k' < k$. If $(\mathcal{R}^{cr}, r', k') \models B_i^{\mathcal{N}}\exists 0$ for some time $k' < k$, then the point (r', k') is clearly $(\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1})$-$\boxdot$-reachable from (r', k), and the claim follows from the induction hypothesis for (r', k') and the transitivity of reachability.

Now suppose that $(\mathcal{R}^{cr}, r', k') \not\models B_i^{\mathcal{N}}\exists 0$ for all $k' < k$. Since $(\mathcal{R}^{cr}, r', k) \models B_i^{\mathcal{N}}\exists 0$, i must have received a value of 0 in one of the messages sent to it in round k, say, from processor j. Since we are in the crash failure mode, if processor i receives a round k message from j, then all other processors received messages from j in all rounds preceding round k. Thus, i does not know at the point (r', k) that j is faulty. Hence, there must be a run r'' in which both i and j are nonfaulty such that $r'_i(k) = r''_i(k)$ and $r'_j(k-1) = r''_j(k-1)$. It follows that $(\mathcal{R}^{cr}, r'', k-1) \models (j \in \mathcal{N}) \wedge B_j^{\mathcal{N}}\exists 0$. Moreover, by construction, $(r'', k-1)$ is $(\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1})$-$\boxdot$-reachable from (r', k) (since $\mathcal{Z}_j^{\Lambda,1} = B_j^{\mathcal{N}}\exists 0$). The claim now follows from the inductive hypothesis for $k-1$ (with respect to r'' and j), and (as before) from the transitivity of reachability. ∎

We now consider $\mathcal{O}^{\Lambda,2}$.

LEMMA A.2. *In* $\mathcal{R}^{cr}$, $\mathcal{O}_i^{\Lambda,2} = B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{cr}) = \emptyset)$.

Proof. Recall that $\mathcal{O}_i^{\Lambda,2} = B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1)$. Observe that $\mathcal{R}^{cr} \models C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1 \Leftrightarrow \boxdot((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$. The $\Leftarrow$ direction is immediate, since $\boxdot(\mathcal{S} = \emptyset) \Rightarrow C^{\boxdot}_{\mathcal{S}}\varphi$ is valid for any nonrigid set $\mathcal{S}$ and formula φ. The proof of Lemma A.1 shows that $\mathcal{R}^{cr} \models C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1 \Rightarrow ((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$. It is immediate that $\mathcal{R}^{cr} \models \boxdot C^{\boxdot}_{\mathcal{N}\wedge \mathcal{Z}^{\Lambda,1}}\exists 1 \Rightarrow \boxdot((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$. Since $C^{\boxdot}_{\mathcal{S}}\varphi \Rightarrow \boxdot C^{\boxdot}_{\mathcal{S}}\varphi$ is valid for all nonrigid sets $\mathcal{S}$ and formulas φ (Lemma 3.5), the result follows.

To complete the proof it suffices to show that

$$\mathcal{R}^{cr} \models B_i^{\mathcal{N}}(\exists 1 \wedge \boxdot((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)) \Leftrightarrow B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset).$$

Setting $\psi = ((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$, this claim has the form

$$\mathcal{R}^{cr} \models B_i^{\mathcal{N}}(\exists 1 \wedge \boxdot \psi) \Leftrightarrow B_i^{\mathcal{N}} \psi.$$

The $\Rightarrow$ direction is straightforward, following from the validity $\models \boxdot \psi \Rightarrow \psi$ and the monotonicity of $B_i^{\mathcal{N}}$. The other direction uses properties of the crash failure mode.

Assume that i is nonfaulty and that $(\mathcal{R}^{cr}, r, m) \models B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$. Since $\mathcal{Z}_i^{\Lambda,1} = B_i^{\mathcal{N}} \exists 0$, this clearly implies that $(\mathcal{R}^{cr}, r, m) \models \neg B_i^{\mathcal{N}} \exists 0$, and hence i's own initial value could not have been 0; thus, $B_i^{\mathcal{N}} \exists 1$ holds. It remains to show that $(\mathcal{R}^{cr}, r, m) \models B_i^{\mathcal{N}} \boxdot \psi$. Therefore, suppose $r_i(m) = r'_i(m)$ and i is nonfaulty in both r and r'. Note that $(\mathcal{R}^{cr}, r', m) \models B_i^{\mathcal{N}} \psi$. We must show that $(\mathcal{R}^{cr}, r', m) \models \boxdot \psi$. Clearly $(\mathcal{R}^{cr}, r', m') \models \psi$ for $m' < m$, for if a nonfaulty processor j believed $\exists 0$ at a time $m' < m$, this fact would have appeared in j's message to i in round $m' + 1 \leq m$, and hence i would already know about the 0 at time m in r', contradicting the fact that $(\mathcal{R}^{cr}, r', m) \models \psi$. Since $(\mathcal{R}^{cr}, r', m) \models B_i^{\mathcal{N}} \psi$ and $i \in \mathcal{N}$, it follows that $(\mathcal{R}^{cr}, r', m) \models \psi$. Notice that, since we are in the crash failure mode, a processor that is known by i to be faulty at time m cannot send messages after round m. Moreover, for i to believe ψ at (r', m), it must be the case that no processor not known by i to be faulty knows $\exists 0$ at (r', m). (The formal proof of this is identical to that of the proof of Theorem 5 in [5].) It immediately follows that, because we are in the crash failure mode, no nonfaulty processor will know $\exists 0$ at any later time $m' > m$ in r'. □

THEOREM 6.2. *In the crash failure mode, the same decisions are made by nonfaulty processors at corresponding points of the protocols* $P0_{\text{opt}}$ *and* $F^{\Lambda,2}$. *Thus, both* $F^{\Lambda,2}$ *and* $P0_{\text{opt}}$ *are optimal EBA protocols for the crash failure mode.*

Proof. Recall that information about the existence of a value of 0 and about the identity of processors that started out with 1 are forwarded in $P0_{\text{opt}}$ as fast as in a full-information protocol. Moreover, in $P0_{\text{opt}}$ every processor sends a message to all other processors in every round. As a result, a straightforward inductive argument shows that for every pair of corresponding runs r of $\mathcal{R}^{cr}_{P0_{\text{opt}}}$ and r' of $\mathcal{R}^{cr}_{FL^2}$, every processor i nonfaulty in these runs, and every time $m \geq 0$, we have that (1) $B_i^{\mathcal{N}} \exists 0$ holds at (r, m) exactly if it holds at (r', m), (2) $B_i^{\mathcal{N}} \neg \exists 0$ holds at (r, m) exactly if it holds at (r', m), and (3) i receives messages from the same set of processors in round m of r and in round m of r'.

Since in both protocols a nonfaulty processor i decides 0 exactly when $B_i^{\mathcal{N}} \exists 0$ first holds, it follows that decisions on 0 take place at corresponding points of the two systems. To complete the proof, we need to show that nonfaulty processors decide 1 at corresponding points of the two systems as well. Notice that the condition $\mathcal{O}_i^{\Lambda,2} = B_i^{\mathcal{N}}((\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}) = \emptyset)$ holds exactly if i knows that no nonfaulty processor can know of a value of 0. It remains to show that this holds at a point of $\mathcal{R}^{cr}_{F^{\Lambda,2}}$ exactly if one of the conditions (a) and (b) defining the decision on 1 in $P0_{\text{opt}}$ holds in the corresponding point of $\mathcal{R}^{cr}_{P0_{\text{opt}}}$.

We first argue that if the nonfaulty processor i does not decide 1 at the point (r, m) of $\mathcal{R}^{cr}_{P0_{\text{opt}}}$, then it does not do so in the corresponding point (r', m) of $\mathcal{R}^{cr}_{F^{\Lambda,2}}$. The case $m = 0$ is straightforward, since processor i has no information about values of other processors. Thus, it must consider it possible that some nonfaulty processor i_1 has initial value 0, so $\mathcal{O}_i^{\Lambda,2}$ cannot hold at $(r', 0)$ (since $\mathcal{Z}_i^{\Lambda,1} = B_i^{\mathcal{N}} \exists 0$). Now suppose that $m \geq 1$. Notice that if neither (a) nor (b) holds for i at all points (r, m') with $m' \leq m$, then there is a sequence $i_1, i_2, \ldots, i_m$ of processors such that, for all $k < m$, in round k processor i_k crashes and sends no messages to processors that survive round k, except possibly to i_{k+1}. In addition, processor i_m crashes in round m without sending

a message to i. It follows that there is a run r'' in which (i) i_1 has initial value 0, (ii) for all $k < m$ we have that i_k sends a message only to i_{k+1} in round k, (iii) the same processors are nonfaulty in r and in r'', (iv) $r_j(k) = r''_j(k)$ for all $k < m$ and all processors j nonfaulty in r (and hence also in r''), and finally (v) i_m sends a message in round m of r'' to all nonfaulty processors $j' \neq i$. Thus, $r''_i(m) = r_i(m)$ and at (r'', m) there are nonfaulty processors that know $\exists 0$. Since r' is a run of a full-information protocol in which the same processors are nonfaulty as in r, it follows that $\mathcal{O}_i^{\Lambda,2}$ will not hold at (r', m) as well, so i does not decide 1 at (r', m).

To complete the proof, we argue that if i decides 1 at the point (r, m) of $P0_{\mathrm{opt}}$, then it also does so at the corresponding point (r', m) of $F^{\Lambda,2}$. By fact (1) above, if condition (a) holds, so that i knows at (r, m) that all initial values were 1, then it knows this fact at (r', m) as well; as a result, i knows nobody can know $\exists 0$, so $\mathcal{O}_i^{\Lambda,2}$ holds.

Suppose that condition (b) holds at (r, m); we need to show that in (r', m) process i believes that $\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1} = \emptyset$. Since condition (b) holds at (r, m), then i does not know $\exists 0$ at (r, m), and i received messages from the same set G of processors in rounds $m-1$ and m of r. By facts (1) and (3) above, the same is true for r' as well. In the crash failure mode, a processor whose round $m-1$ message to i is not delivered is definitely silent from round m on. Hence, no processor other than those in G sends (or receives) messages in round m of r'. Since $F^{\Lambda,2}$ is a full-information protocol, it follows that if the set of messages sent by the processors in G in round m does not contain information about the existence of a value of 0, then nobody knows $\exists 0$ at time (r', m). Finally, by assumption, processor i receives all messages from processors in G in round m and still does not know that $\exists 0$ holds there. We conclude that $\mathcal{O}_i^{\Lambda,2}$ holds at (r', m), and we are done. □

We next want to prove that in the omission failure mode, $F^{\Lambda,2}$ does not hold. We first need to get some insight into $\mathcal{Z}^{\Lambda,1}$ in the case of omission failures. The following is a variant of Lemma A.1 modified to suit the omission failure model.

LEMMA A.3. *Suppose that $n \geq t+2$, r is a run of $\mathcal{R}^{om}$ in which fewer than t processors fail, and i is nonfaulty in r. Then for all times $m \geq 0$ we have that $(\mathcal{R}^{om}, r, m) \models B_i^{\mathcal{N}} \exists 0 \Rightarrow \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$.*

Proof. We proceed by induction on k to show that if k is the first time in r such that $(\mathcal{R}^{om}, r, k) \models B_i^{\mathcal{N}} \exists 0$, then $(\mathcal{R}^{om}, r, k) \models \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$. This suffices since if $\neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$ is true at (r, k), it is true at (r, m) for all m by part (g) of Lemma 3.5. If $k = 0$, it must be the case that i has initial value 0, and hence its state at $(r, 0)$ is the same as at $(r', 0)$, where r' is the run in which no processor started with initial value 1 and all processors are nonfaulty. Since $\exists 1$ does not hold at $(r', 0)$ and i is nonfaulty at both $(r, 0)$ and $(r', 0)$, then by Proposition 3.3 it follows that $(\mathcal{R}^{om}, r, 0) \models \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$.

Now suppose that $k > 1$ and the induction hypothesis holds for $k-1$. Then i must have received a 0 from some processor j in round k of r. If i does not know that j is faulty, then let r' be a run with the same failure pattern as r, except that processor j is nonfaulty. (In particular, $r' = r$ if j is nonfaulty in r.) Since $(\mathcal{R}^{om}, r', k') \models B_j^{\mathcal{N}} \exists 0$ for some $k' < k$, by the induction hypothesis $(\mathcal{R}^{om}, r', k') \models \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$. The claim follows since (r, k) is $(\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1})$-$\boxdot$-reachable from (r', k'). If i does consider j to be faulty at (r, k), consider a run r' that is identical to r up to time k, except that there is a processor j'' faulty in r' that fails for the first time in round k of r and sends messages to all processors in round k, except some nonfaulty processor $j' \neq i$. Note that this means that $r_i(k) = r'_i(k)$. (Such processors j' and j'' exist because,

by assumption, $n \geq t+2$, and less than t processors fail in r.) Finally, consider a run r'' which is identical to r' up to time k, except that whichever processor sent j a message saying $\exists 0$ in r at round $k-1$ also sends it to j', and at round k, and all processors send the same messages in r' and r'', except that j' sends i a message saying $\exists 0$. Note that $r'_j(k) = r''_j(k)$. Now, by construction, i learns about $\exists 0$ in r'' from a processor which it does not know to be faulty. Thus, by the earlier argument, $(\mathcal{R}^{om}, r'', k) \models \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$. The induction hypothesis carries us through as above, and we are done. □

PROPOSITION 6.3. *If $t > 1$ and $n \geq t+2$, then there are runs of $F^{\Lambda,2}$ in $\mathcal{R}^{om}$ in which the nonfaulty processors do not decide.*

Proof. Consider the run r where all the processors start with 1, processor 1 is faulty and never sends messages to any processor, and no other processors fail. Clearly, the weak validity condition implies that any nonfaulty processor which decides in r must decide 1. We now show that no nonfaulty processor can ever decide 1 in r. Assume that a nonfaulty processor i decides 1 at (r, m) for some m. Then, by definition, $\mathcal{O}^2_i$ holds at (r, m). Let r' be a run of R^{om} in which processor 1 is the only faulty processor and where the first m rounds of r' differ from those of r only in the following two ways: (a) processor 1 has initial value 0, and (b) exactly one message sent by processor 1 is delivered in r'; it is a message sent in round m to some nonfaulty processor $j \neq i$. Clearly, $r_i(m) = r'_i(m)$. It follows that $\mathcal{O}^2_i$, which depends only on i's local state, holds at (r', m) as well. By definition of $\mathcal{O}^2_i$, we thus have that $(\mathcal{R}^{om}, r', m) \models B^{\mathcal{N}}_i C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$. Since i is nonfaulty in r', we have that

$$(R^{om}, r', m) \models C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1.$$

Notice, however, that in r' the nonfaulty processor $j \neq i$ receives a message reporting a value of 0 from processor 1 in round m. As a result, we have that $(R^{om}, r', m) \models B^{\mathcal{N}}_j \exists 0$. Applying Lemma A.3 with respect to r' and j now yields $(R^{om}, r', m) \models \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^{\Lambda,1}} \exists 1$, giving us a contradiction. □

PROPOSITION 6.4. *In a run r of FIP($\mathcal{Z}^0, \mathcal{O}^0$) in the omission failure mode where f processors actually fail, all nonfaulty processors decide by time $f+1$.*

Proof. Let $F = FIP(\mathcal{Z}^0, \mathcal{O}^0)$. Suppose that r is a run in $\mathcal{R}^{om}_F$ in which f processors fail. Then, for each nonfaulty processor i, there is a round $m \leq f+1$ such that the only processors from which i does not receive a message in round m are ones that did not send it a message in some earlier round. Since a nonfaulty processor sends its state to everybody at each round, if processor j does not send a message to i in round $m' < m$, in round m (in fact, in all rounds between m' and m) i tells all processors that j is faulty. Thus, for each such j, we have

$$(\mathcal{R}^{om}_F, r, m) \models B_i(E_{\mathcal{N}}(j \notin \mathcal{N})).$$

We claim that $(\mathcal{R}^{om}_F, r, m) \models B^{\mathcal{N}}_i \exists 0^* \vee B^{\mathcal{N}}_i \neg \exists 0^*$, so that i decides at (r, m). Clearly, if i receives a message in round m from some processor j that is not known by i to be faulty which implies $\exists 0^*$, then $(\mathcal{R}^{om}_F, r, m) \models B^{\mathcal{N}}_i \exists 0^*$, and i can decide 0. Otherwise, we claim that $(\mathcal{R}^{om}_F, r, m) \models B^{\mathcal{N}}_i \neg \exists 0^*$. Suppose by way of contradiction that there exists a run r' such that $r'_i(m) = r_i(m)$, i is nonfaulty in r', and $(\mathcal{R}^{om}_F, r', m) \models \exists 0^*$. Thus, there must be a 0-chain in r' ending with some nonfaulty processor j at some time $m' \leq m$. Thus, j hears $\exists 0^*$ at some round $m' \leq m$. If $m' < m$, since j is nonfaulty in r', it will tell i $\exists 0^*$ in round $m'+1$ of r', so there will be a 0-chain ending with i at $(r', m'+1)$. Thus, i decides 0 at $(r', m'+1)$. Since $r_i(m) = r'_i(m)$ and F is a full-information protocol, we must have $r_i(m'+1) = r'_i(m'+1)$, and i decides 0

at $(r, m'+1)$, contradicting our assumption. Therefore, suppose that j hears $\exists 0^*$ in round m from j'. Thus, j considers j' to be nonfaulty at (r', m). However, this means that i must consider j' to be nonfaulty at $(r', m-1)$ (and hence at $(r, m-1)$), since otherwise i would tell j that j' was faulty in round m of r'. By choice of (r, m), i must also hear from j' in round m of r. Since $r_i(m) = r'_i(m)$, i also hears from j' in round m of r' and gets the same message in both r and r'. Again, this means that i hears $\exists 0^*$ in r, a contradiction. □

PROPOSITION 6.6. *F^* is an optimal EBA in the omission failure mode that dominates FIP($\mathcal{Z}^0, One^0$).*

Proof. Applying the first step of our construction, optimizing the decision on 1, given the rule in $F(\mathcal{Z}^0, \mathcal{O}^0)$ for deciding 0, we get

$$\mathcal{Z}_i^1 = B_i^{\mathcal{N}}(\exists 0 \wedge \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0} \exists 1) \text{ and}$$
$$\mathcal{O}_i^1 = B_i^{\mathcal{N}}(\exists 1 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0} \exists 1).$$

We claim that $\mathcal{Z}^1 = \mathcal{Z}^1$ and $\mathcal{O}^1 = \mathcal{O}^0$. This follows from the next two lemmas.

LEMMA A.4. *Let F be any full-information protocol. Then*

$$\mathcal{R}_F^{om} \models C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0} \exists 1 \Leftrightarrow \boxdot(\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset.$$

Proof. The "if" direction follows immediately from the definition, so we prove the "only if" direction. Assume that $(\mathcal{R}_F^{om}, r, m) \models \neg \boxdot(\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset$. Then there is a first time $l = l(r)$ at which we have $(\mathcal{R}_F^{om}, r, l(r)) \models (\mathcal{N} \wedge \mathcal{Z}^0) \neq \emptyset$. We prove by induction on $l(r)$ that $C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0} \exists 1$ does not hold in r. If $l(r) = 0$, since $\mathcal{Z}_i^0 = B_i^{\mathcal{N}} \exists 0^*$, it must be the case that all the nonfaulty processors in r have initial value 0. Thus, all the nonfaulty processors in r initially consider it possible that $\neg \exists 1$ holds; the result follows immediately.

Now suppose that $l(r) > 0$ and the induction hypothesis holds for $l(r)-1$. Suppose $B_i^{\mathcal{N}} \exists 0^*$ holds at $(r, l(r))$ for some nonfaulty processor i. Then there must be some processor j that is not known to i as faulty at (r, l) that tells i $B_j^{\mathcal{N}} \exists 0^*$ in round l of r. Let r' be the run with the same failure pattern and initial states as r, except that j is nonfaulty in r'. It is easy to see that $r_i(l(r)) = r'_i(l(r))$. However, $l(r') = l(r) - 1$, and hence by the induction hypothesis $C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0} \exists 1$ does not hold in r'. By Proposition 3.3, it follows that $C^{\boxdot}_{\mathcal{N} \wedge B^{\mathcal{N}} \exists 0^*} \exists 1$ does not hold in r either, and we are done. □

Lemma A.4 implies that $\mathcal{Z}^1, \mathcal{O}^1$ reduce to

$$\mathcal{Z}_i^1 \equiv B_i^{\mathcal{N}}(\exists 0 \wedge \neg \boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)) \text{ and}$$
$$\mathcal{O}_i^1 \equiv B_i^{\mathcal{N}}(\exists 1 \wedge \boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)).$$

As the following lemma shows, even further simplification is possible.

LEMMA A.5. *Let F be any full-information protocol. Then*

$$\mathcal{R}_F^{om} \models B_i^{\mathcal{N}}(\exists 0 \wedge \neg \boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)) \Leftrightarrow B_i^{\mathcal{N}}(\exists 0^*)$$

and

$$\mathcal{R}_F^{om} \models B_i^{\mathcal{N}}(\exists 1 \wedge \boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)) \Leftrightarrow B_i^{\mathcal{N}}(\neg \exists 0^*).$$

Proof. We first need the following claim.

Claim. $\mathcal{R}_F^{om} \models (\mathcal{N} \wedge \mathcal{Z}^0) \neq \emptyset \Leftrightarrow \exists 0^*$.

Clearly, if $\exists 0^*$ holds at (r, m), then there is a 0-chain ending at some nonfaulty processor j at some point (r, m') with $m' \leq m$. Then $j \in (\mathcal{N} \wedge \mathcal{Z}^0)(r, m'')$ for all

$m'' \geq m'$. Conversely, if $j \in (\mathcal{N} \wedge \mathcal{Z}^0)(r,m)$, then $B_j^{\mathcal{N}}(\exists^* 0)$ holds at (r,m), and j is nonfaulty in r. Thus, $\exists^* 0$ holds at (r,m) too. This completes the proof of the claim.

For the first half of the lemma, $B_i^{\mathcal{N}}(\exists 0^*)$ clearly implies $B_i^{\mathcal{N}}(\exists 0)$. It is immediate from the claim that $\mathcal{R}_F^{om} \models B_i^{\mathcal{N}}(\exists 0^*) \Rightarrow B_i^{\mathcal{N}}(\neg\boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset))$. For the converse, the claim implies that $\mathcal{R}_F^{om} \models B_i^{\mathcal{N}}(\neg\boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)) \Rightarrow B_i^{\mathcal{N}}(\neg\boxdot\neg\exists^* 0)$. However, it should be clear that the only way processor i can believe that $\exists 0^*$ holds at some point in a run r is if i currently believes $\exists 0^*$ holds.

For the second half of the lemma, note that it is immediate from the claim that $\mathcal{R}_F^{om} \models B_i^{\mathcal{N}}(\exists 1 \wedge \boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset)) \Rightarrow B_i^{\mathcal{N}}(\neg\exists 0^*)$. For the converse, suppose that $(\mathcal{R}_F^{om}, r, m) \models B_i^{\mathcal{N}}(\neg\exists 0^*)$. Clearly, this means that processor i must have an initial value of 1 in r, so $(\mathcal{R}_F^{om}, r, m) \models B_i^{\mathcal{N}}\exists 1$. To see that $(\mathcal{R}_F^{om}, r, m) \models B_i^{\mathcal{N}}(\boxdot((\mathcal{N} \wedge \mathcal{Z}^0) = \emptyset))$, suppose not. Then, by the claim, there is some time m' such that $(\mathcal{R}_F^{om}, r, m') \models B_i^{\mathcal{N}}(\exists 0^*)$. Since $(\mathcal{R}_F^{om}, r, m) \models B_i^{\mathcal{N}}(\neg\exists 0^*)$, we must have $m' > m$. However, this means that i believes at (r,m') there is a 0-chain with m' processors. Let i_m and i_{m+1} be the mth and $(m+1)$st processors in that chain. Processor i must believe at time m that i_m is faulty, otherwise we would not have $(\mathcal{R}_F^{om}, r, m) \models B_i^{\mathcal{N}}(\neg\exists 0^*)$. However, this means that i would tell i_{m+1} in round $m+1$ that i_m is faulty, contradicting the existence of the 0-chain. This completes the proof of the lemma. □

From Lemmas A.4 and A.5, we have that

$$\mathcal{Z}_i^1 \equiv B_i^{\mathcal{N}}\exists 0^* \text{ and}$$
$$\mathcal{O}_i^1 \equiv B_i^{\mathcal{N}}\neg\exists 0^*.$$

Now we apply the second step of our construction and obtain

$$\mathcal{Z}_i^2 = B_i^{\mathcal{N}}(\exists 0 \wedge C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0}\exists 0) \text{ and}$$
$$\mathcal{O}_i^2 = B_i^{\mathcal{N}}(\exists 1 \wedge \neg C^{\boxdot}_{\mathcal{N} \wedge \mathcal{Z}^0}\exists 0).$$

Thus, $F^* = FIP(\mathcal{Z}^*, \mathcal{O}^*)$. By Theorem 5.2, F^* dominates $FIP(\mathcal{Z}^0, \mathcal{O}^0)$. □

Acknowledgments. IBM's support is gratefully acknowledged. The paper greatly benefitted from the comments of two very thorough referees. We thank them for their efforts.

REFERENCES

[1] B. F. Chellas, *Modal Logic*, Cambridge University Press, Cambridge, UK, 1980.
[2] B. Coan, *A communication-efficient canonical form for fault-tolerant distributed protocols*, in Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, 1986, pp. 63–72.
[3] D. Dolev, R. Reischuk, and H. R. Strong, *Early stopping in Byzantine agreement*, J. ACM, 34 (1990), pp. 720–741.
[4] D. Dolev and H. R. Strong, *Requirements for agreement in a distributed system*, in Distributed Data Bases, H. J. Schneider, ed., North-Holland, Amsterdam, 1982, pp. 115–129.
[5] C. Dwork and Y. Moses, *Knowledge and common knowledge in a Byzantine environment: Crash failures*, Inform. and Comput., 88 (1990), pp. 156–186.
[6] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning About Knowledge*, MIT Press, Cambridge, MA, 1995.
[7] M. J. Fischer, *The consensus problem in unreliable distributed systems*, in Foundations of Computation Theory, M. Karpinski, ed., Lecture Notes in Comput. Sci. 185, Springer-Verlag, Berlin, 1983, pp. 127–140.
[8] J. Y. Halpern and Y. Moses, *Knowledge and common knowledge in a distributed environment*, J. ACM, 37 (1990), pp. 549–587.
[9] J. Y. Halpern and Y. Moses, *A guide to completeness and complexity for modal logics of knowledge and belief*, Artificial Intelligence, 54 (1992), pp. 319–379.

[10] J. Y. Halpern and M. Y. Vardi, *The complexity of reasoning about knowledge and time,* I: *Lower bounds*, J. Comput. System Sci., 38 (1989), pp. 195–237.
[11] L. Lamport and M. J. Fischer, *Byzantine Generals and Transactions Commit Protocols*, Tech. Report Opus 62, SRI International, Menlo Park, CA, 1982.
[12] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Vol. 1, Springer-Verlag, New York, 1992.
[13] Y. Moses and Y. Shoham, *Belief as defeasible knowledge*, Artificial Intelligence, 64 (1993), pp. 299–322.
[14] Y. Moses and M. R. Tuttle, *Programming simultaneous actions using common knowledge*, Algorithmica, 3 (1988), pp. 121–169.
[15] G. Neiger and R. A. Bazzi, *Using knowledge to optimally achieve coordination in distributed systems*, Theoret. Comput. Sci., 220 (1999), pp. 31–65.
[16] P. Panangaden and S. Taylor, *Concurrent common knowledge: Defining agreement for asynchronous systems*, Distrib. Comput., 6 (1992), pp. 73–93.
[17] M. Pease, R. Shostak, and L. Lamport, *Reaching agreement in the presence of faults*, J. ACM, 27 (1980), pp. 228–234.
[18] K. Perry and S. Toueg, *Distributed agreement in the presence of processor and communication faults*, IEEE Trans. Software Engrg., 12 (1986), pp. 477–481.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 866–875

EVASIVENESS OF SUBGRAPH CONTAINMENT AND RELATED PROPERTIES*

AMIT CHAKRABARTI†, SUBHASH KHOT†, AND YAOYUN SHI†

Abstract. We prove new results on evasiveness of monotone graph properties by extending the techniques of Kahn, Saks, and Sturtevant [*Combinatorica*, 4 (1984), pp. 297–306]. For the property of containing a subgraph isomorphic to a fixed graph, and a fairly large class of related n-vertex graph properties, we show evasiveness for an arithmetic progression of values of n. This implies a $\frac{1}{2}n^2 - O(n)$ lower bound on the decision tree complexity of these properties.

We prove that properties that are preserved under taking graph minors are evasive for all sufficiently large n. This greatly generalizes a theorem due to Best, van Emde Boas, and Lenstra [*A Sharpened Version of the Aanderaa–Rosenberg Conjecture*, Report ZW 30/74, Mathematisch Centrum, Amsterdam, The Netherlands, 1974] which states that planarity is evasive. We prove a similar result for bipartite subgraph containment.

Key words. decision tree complexity, monotone graph properties, evasiveness, topological method, graph property testing

AMS subject classifications. 68Q17, 68Q25

PII. S0097539700382005

1. Introduction.

Suppose we have an input graph G and are required to decide whether or not it has a certain (isomorphism invariant) property P. The graph is given by an oracle which answers queries of the form "Is (x, y) an edge of G?" A *decision tree algorithm* for P is a strategy that specifies a sequence of such queries to the oracle, where each query may depend upon the outcomes of the previous ones, terminating when sufficient information about G has been obtained to decide whether or not P holds for G. The *cost* of such a decision tree algorithm is the worst case number of queries that it makes. The *decision tree complexity* of P is the minimum cost of any decision tree algorithm for P.

Since an n-vertex graph has $\frac{1}{2}n(n-1)$ vertex pairs each of which could either be an edge or not, it is clear that any property of n-vertex graphs has complexity at most $\frac{1}{2}n(n-1)$. If a property happens to have complexity *exactly* $\frac{1}{2}n(n-1)$, then it is said to be *evasive*.[1]

A property of n-vertex graphs is said to be *monotone* if, starting with a graph which has the property, the addition of edges does not destroy the property. It is said to be *nontrivial* if there exists an n-vertex graph which has the property and one which does not. Connectedness, nonplanarity, non-k-colorability, and the property of containing a perfect matching are all examples of nontrivial monotone properties (for sufficiently large n). Rosenberg [7] attributes to Karp the following conjecture which, remarkably, remains open even today.

KARP CONJECTURE. *Every nontrivial monotone graph property is evasive.*

*Received by the editors December 5, 2000; accepted for publication (in revised form) August 30, 2001; published electronically January 11, 2002. A preliminary version of this paper appeared in *Proceedings of the 18th International Symposium on Theoretical Aspects of Computer Science*, Dresden, 2001. This work was supported in part by NSF grant CCR-96-23768, NSF grant CCR-98-20855, and ARO grant DAAH04-96-1-0181.

http://www.siam.org/journals/sicomp/31-3/38200.html

†Department of Computer Science, Princeton University, Princeton, NJ 08544 (amitc@cs.princeton.edu, khot@cs.princeton.edu, shiyy@cs.princeton.edu).

[1]Some authors call such properties "elusive" instead of evasive.

As a first step towards a resolution of this conjecture, Rivest and Vuillemin [6] proved that such properties have complexity at least $n^2/16$, thereby settling the Aanderaa–Rosenberg conjecture [7] of an $\Omega(n^2)$ complexity lower bound. The next big advance was the work of Kahn, Saks, and Sturtevant [4], where an interesting *topological approach* was used to prove that the Karp Conjecture holds whenever n is a prime power. Triesch [8, 9] used this approach, together with complicated algebraic constructions, to prove the evasiveness of some special classes of properties: specifically, these papers established evasiveness of graph properties that are always false when (i) the graph contains either a 3-cycle or a 4-cycle, and when (ii) the graph is not bipartite, respectively. Similar topological ideas were used by Yao [10] to prove a related result, namely, that nontrivial monotone *bipartite* graph properties are always evasive. Prior to the work of Kahn, Saks, and Sturtevant [4], adversarial strategies had been devised to prove the evasiveness, for all n, of certain specific graph properties (see, e.g., [1], [5], and [3, Ch. 8]). These strategies worked for the properties of acyclicity, connectedness, 2-connectedness, planarity, and simple variants on these. The most sophisticated of these adversarial strategies was one used by Bollobás [2] to prove the evasiveness of the property of containing a k-clique, for any k, $2 \leq k \leq n$.

Let H be any fixed graph. For n-vertex graphs, let Q_n^H denote the property of containing H as a subgraph (not necessarily as an induced subgraph). From the work of Bollobás [2] we know that Q_n^H is evasive for all n in the special case when H is a complete graph. This raises the natural question "What can we say about general H?"

In this paper, we study this question and some related ones, extending the topological approach of [4] to a fairly general class of graph properties. For each of these properties, we draw stronger inferences than [4]. Our main theorem is stated below.

THEOREM 1.1 (main theorem). *For any fixed graph H there exists an integer r_0 with the following property. Suppose $n = \sum_{i=1}^{r} q^{\alpha_i}$, where q is a prime power, $q \geq |H|$, each $\alpha_i \geq 1$, and $r \equiv 1 \pmod{r_0}$. Then Q_n^H is evasive.*

In order to understand the significance and strength of this theorem, consider the following statements (proven in this paper). Each of these statements follows either from the main theorem or from the techniques used in proving it.

1. For any graph H, there is an *arithmetic progression* such that Q_n^H is evasive for all n in the progression. Note that this is a much stronger inference than can be drawn by applying the results of [4].

2. The decision tree complexity of Q_n^H is $\frac{1}{2}n^2 - O(n)$. This bound does not follow from the results of [4].

3. If the graph H is *bipartite*, then Q_n^H is evasive for large enough n.

4. Any n-vertex nontrivial graph property that is preserved under taking graph minors is evasive for large enough n. This includes lots of very natural graph properties such as embeddability on any surface, outerplanarity, linkless embeddability in $\mathbf{R}^3$, the property of being a series-parallel graph, etc. Thus, our result generalizes a result of Best, van Emde Boas, and Lenstra [1], who show that planarity is evasive.

5. Any monotone boolean combination of the properties Q_n^H for several different graphs H still satisfies our main theorem. Thus, for example, if H_1, H_2, and H_3 are fixed graphs, then the property of containing as subgraph either H_1 or both of H_2 and H_3 is still evasive for those n which satisfy the conditions of the main theorem.

The remainder of the paper is organized as follows. In section 2 we review the basics of the topological approach of Kahn, Saks, and Sturtevant [4], establishing a connection between proving evasiveness of monotone properties and computing Euler

characteristics of abstract complexes. Then in section 3 we define a certain auxiliary property of graphs and prove a technical result (called the main lemma) about this property. This result is then used in section 4 to prove our main theorem. In section 5, we provide proofs for the additional results itemized above. We end with some concluding remarks in section 6.

Notations, terminology, and conventions. We call a graph *trivial* if it has no edges. Throughout this paper, all graphs will be assumed to be nontrivial, to have no loops, and to have no parallel edges. For a graph G, $|G|$ will denote the number of vertices in G, also called the *size* of G, $V(G)$ will denote its vertex set, $E(G)$ its edge set, $\mathrm{chr}(G)$ its chromatic number, and $\mathrm{clq}(G)$ the size of its largest clique. Graphs which occur as "input graphs" on which boolean properties are to be tested are assumed to always be vertex-labeled. All other graphs are assumed to be unlabeled, unless otherwise specified. When we speak of an "edge" in an input graph, we really mean an unordered vertex pair which may or may not be an edge.

2. Review of the topological approach. A property of m boolean variables $x_1, \ldots, x_m$ is a function $P : \{0,1\}^m \to \{0,1\}$; we say that the m-tuple $(x_1, \ldots, x_m)$ has (or satisfies) property P if $P(x_1, \ldots, x_m) = 1$. We say that P is *monotone* if for every m-tuple $(x_1, \ldots, x_m)$ that satisfies P, increasing any x_i from 0 to 1 yields an m-tuple that also satisfies P. We say that P is *evasive* if any decision tree algorithm for P has cost m. In our study of graph properties, the variables will be unordered pairs of vertices (i.e., potential edges of the graph) and P will be required to be invariant under relabelings of the graph.

Let $[m]$ denote the set $\{1, 2, \ldots, m\}$ and consider the collection of subsets $S \subseteq [m]$ with the following property: setting the variables indexed by S to 1 and those indexed by $[m] \setminus S$ to 0 yields an m-tuple which *does not satisfy* P. Since P is monotone, this collection of sets is downward closed under set inclusion. Recall that such a downward closed collection of sets is called an *abstract complex* and that the sets in this collection are called the *faces* of the complex. This observation motivates the following definition.

DEFINITION 2.1. *If P is monotone, then the abstract complex associated with P, denoted $\Delta(P)$, is defined as follows:*

$$\Delta(P) = \{S \subseteq [m] \,:\, \text{If } x_i = 1 \iff i \in S, \text{ then } (x_1, \ldots, x_m) \text{ does not satisfy } P\}.$$

Associated with an abstract complex Δ is a topologically important number called its *Euler characteristic* which is denoted $\chi(\Delta)$ and is defined as follows:

$$\chi(\Delta) = \sum_{\emptyset \neq F \in \Delta} (-1)^{|F|-1}. \tag{1}$$

Kahn, Saks, and Sturtevant [4] showed that nonevasiveness of P has topological consequences for $\Delta(P)$. The following theorem is implicit in their work.

THEOREM 2.2 (Kahn, Saks, and Sturtevant [4]). *If the monotone property P is not evasive, then $\chi(\Delta(P)) = 1$.*

For our result, we shall need to use a stronger theorem which can also be found in [4]. Let Δ be an abstract complex defined on $[m]$ and let Γ be a finite group which acts on the set $[m]$, preserving the faces of Δ. The action partitions $[m]$ into orbits, say $A_1, \ldots, A_k$. We use the action of Γ to define another abstract complex Δ_Γ on $[k]$ as follows:

$$\Delta_\Gamma = \left\{ S \subseteq [k] \,:\, \bigcup_{i \in S} A_i \in \Delta \right\}. \tag{2}$$

Sometimes, as is the case with our work, it is not easy to say much about $\Delta(P)$ for a monotone property P. However, it is possible to find some group Γ such that its action produces a more understandable abstract complex $(\Delta(P))_\Gamma$. The next theorem, the most important tool in [4], says that if Γ has certain rather restrictive properties, then nonevasiveness of P has a topological consequence on this new complex.

THEOREM 2.3 (Kahn, Saks, and Sturtevant [4]). *Suppose Γ has a normal subgroup Γ_1 which is such that $|\Gamma_1|$ is a prime power and the quotient group Γ/Γ_1 is cyclic. Then if P is not evasive, we have $\chi((\Delta(P))_\Gamma) = 1$.*

An application of this result leads to the following theorem which is the main result of [4].

THEOREM 2.4 (Kahn, Saks, and Sturtevant [4]). *Let P_n be a nontrivial monotone property of n-vertex graphs. If n is a prime power, then P_n is evasive.*

In order to derive Theorem 2.4 from Theorem 2.3, Kahn, Saks, and Sturtevant [4] construct a group which acts on the vertices of the input graph and thus, indirectly, on the edges. The number theoretic constraint on n is a consequence of the fact that this action depends crucially on being able to view the vertices of the graph as elements of a finite field. Our approach to proving evasiveness for more general n will be to devise a more sophisticated group action. Before we do so, we will need an auxiliary result which we shall establish in the next section.

3. The main lemma. Consider the following operation on a graph G. Let the vertices of G be colored, using *all* the colors in some set C, so that no two adjacent vertices get the same color. Let G' be a graph with vertex set C where two distinct vertices $c_1, c_2 \in C$ are adjacent iff the coloring assigns colors c_1 and c_2 to the end points of some edge in G. We shall call G' a *compression of graph G induced by coloring C*. If there exists a C which induces a compression G' of G, we shall write $G' \lhd G$.

DEFINITION 3.1. *A family $\mathcal{F}$ of graphs is said to be closed under compression if for graphs G, H such that $G \in \mathcal{F}$ and $H \lhd G$ we have $H \in \mathcal{F}$.*

Let $\mathcal{F}$ be a nonempty finite family of (nontrivial) graphs that is closed under compression. The property $P_n^{\mathcal{F}}$ that an input graph G on n vertices contains some member of $\mathcal{F}$ as a subgraph is clearly nontrivial, for n large enough, and monotone. Let $\Delta_n^{\mathcal{F}}$ be the abstract complex associated with this property and let $\chi_n = \chi(\Delta_n^{\mathcal{F}})$ be the Euler characteristic of this complex.

The purpose of this section is to establish that for any such family $\mathcal{F}$, we have $\chi_n \neq 1$ infinitely often. Let us set

$$T = 2^{2^t}, \text{ where } t \text{ is the smallest integer such that } T \geq \min_{F \in \mathcal{F}} |F|. \tag{3}$$

We shall prove the following lemma.

LEMMA 3.2 (main lemma). *If $n \equiv 1 \pmod{T-1}$, then $\chi_n \equiv 0 \pmod 2$.*

Since we care only about $\chi_n \bmod 2$, we can use the fact that addition and subtraction are equivalent mod 2 in (1) to get[2]

$$\chi_n \equiv \#\{G \,:\, G \text{ is nontrivial and does not satisfy } P_n^{\mathcal{F}}\} \pmod 2. \tag{4}$$

[2]Note that we are counting not graphs, but labeled graphs.

Consider n-vertex input graphs with vertices labeled with integers from 0 to $n-1$. For $n > T$, let us define a group action on such graphs as follows. For $a, b \in \{0, 1, 2, \ldots, T-1\}$ and a odd, let permutation $\phi_{a,b}$ be defined by mapping vertex i to vertex $(ai + b) \bmod T$ for $i \in \{0, 1, \ldots, T-1\}$. The other $n - T$ vertices are left fixed. It is routine to check that the set of all these permutations forms a group Φ under composition, thereby defining a group action on the labeled vertices. This action induces an action on graphs in the obvious manner, thereby partitioning the set of all labeled n-vertex graphs into orbits. Since the order $|\Phi|$ of the group is $T^2/2$, a power of 2, each orbit has size a power of 2. Therefore, (4) can be modified to

$$\chi_n \equiv \#\left\{G : \begin{array}{c} G \text{ is nontrivial, invariant under} \\ \Phi \text{ and does not satisfy } P_n^{\mathcal{F}} \end{array}\right\} \pmod 2. \tag{5}$$

The action of Φ on the vertices also induces an action on edges (or rather, on unordered pairs of distinct vertices, each of which may or may not be an edge), not to be confused with the action on labeled graphs mentioned above. Therefore the set of edges amongst vertices $0, 1, \ldots, T-1$ is partitioned into orbits. Since any odd integer is invertible mod T, we get 2^t orbits $E_0, E_1, \ldots, E_{2^t-1}$, where

$$E_i = \{(x, y) : 0 \le x < y < T, \;\; y - x = 2^i k \text{ for some odd number } k\} \; . \tag{6}$$

Let G be an invariant graph. From now on, let us refer to the vertices $0, 1, \ldots, T-1$ as *left vertices* and the rest as *right vertices*. Let G_{left} and G_{right} denote the subgraphs of G induced by the left and right vertices, respectively. By invariance of G, the set of right vertices adjacent to any left vertex is the same for each left vertex; let $\mathcal{R}(G)$ denote this set. Also, the set of edges $E(G_{\text{left}})$ is the union of a certain number of the orbits E_i; let $\text{orb}(G)$ denote this number. We shall show that whether or not G has the property $P_n^{\mathcal{F}}$ is completely determined once $G_{\text{right}}, \mathcal{R}(G)$ and this number $\text{orb}(G)$ are fixed; the specific G_{left} does not matter.

LEMMA 3.3. *For any invariant G, we have* $\text{chr}(G_{\text{left}}) = \text{clq}(G_{\text{left}}) = 2^{\text{orb}(G)}$.

Proof. Let $I \subseteq \{0, 1, \ldots, 2^t - 1\}$ be such that $E(G_{\text{left}}) = \bigcup_{i \in I} E_i$; then we have $|I| = \text{orb}(G)$. Consider two vertices x, y of G_{left}. If their binary representations agree on the bit positions indexed by I, then $x - y = \sum_{i \in I'} \pm 2^i$ for some set I' disjoint from I. By (6), this implies $(x, y) \notin E(G_{\text{left}})$. Therefore, the vertices of G_{left} can be partitioned into $2^{|I|}$ independent sets; thus $\text{chr}(G_{\text{left}}) \le 2^{\text{orb}(G)}$. On the other hand, if x, y are such that the bits in positions outside I are all zero, then $x - y = \sum_{i \in I''} \pm 2^i$ for some $I'' \subseteq I$, which by (6) implies that $(x, y) \in E(G_{\text{left}})$. Therefore, G_{left} has a clique of size $2^{|I|} = 2^{\text{orb}(G)}$. The lemma follows. □

LEMMA 3.4. *Let G_1, G_2 be two invariant n-vertex labeled graphs with $G_{1,\text{right}} = G_{2,\text{right}}$, $\mathcal{R}(G_1) = \mathcal{R}(G_2)$, and $\text{orb}(G_1) = \text{orb}(G_2)$. Then G_1 has property $P_n^{\mathcal{F}}$ iff G_2 does.*

Proof. Suppose G_1 has property $P_n^{\mathcal{F}}$; we shall show that G_2 does too. Suppose G_1 contains $F \in \mathcal{F}$ as a subgraph. We fix a particular occurrence of F within G_1 so that we can talk about $F_{\text{left}}, F_{\text{right}}$ and $\mathcal{R}(F) := \mathcal{R}(G_1) \cap V(F)$.

Using Lemma 3.3 and the hypothesis, we obtain $\text{chr}(F_{\text{left}}) \le \text{chr}(G_{1,\text{left}}) = \text{clq}(G_{2,\text{left}})$. Let $h = \text{chr}(F_{\text{left}})$; from the above inequality it is clear that $G_{2,\text{left}}$ contains K_h as a subgraph. Fix a particular occurrence of K_h and, starting with the graph F_{right}, connect each of the h left vertices in this occurrence to each vertex in $\mathcal{R}(F)$. Let F' be the resulting graph. Since $\mathcal{R}(F) \subseteq \mathcal{R}(G_1) = \mathcal{R}(G_2)$ and since F_{right} is a subgraph of $G_{1,\text{right}} = G_{2,\text{right}}$, it follows that F' is a subgraph of G_2.

Consider the following coloring of the graph F: we use h colors for its left vertices and color each right vertex with a distinct color, never using any of these h colors. Let $F'' \lhd F$ be the compression of F induced by this coloring. It is not hard to see that F'' is a subgraph of F' and therefore of G_2. Since $\mathcal{F}$ is closed under compression, $F'' \in \mathcal{F}$. Therefore G_2 has property $P_n^{\mathcal{F}}$. □

LEMMA 3.5. *For $n \geq T = 2^{2^t}$, we have $\chi_n \equiv \chi_{n-T+1} \pmod 2$.*

Proof. Let k be a fixed integer with $0 \leq k \leq 2^t$. Recall that the group action induced on the edges creates 2^t orbits. Consider the family of all n-vertex invariant graphs G with $\mathcal{R}(G)$ and G_{right} fixed, and $\text{orb}(G) = k$. By Lemma 3.4, either all graphs in this family have property $P_n^{\mathcal{F}}$ or none of them does. The size of this family is $\binom{2^t}{k}$ which is even if $k \neq 0$ and $k \neq 2^t$. If $k = 2^t$, G_{left} is a complete graph, and so G contains a clique of size T. From (3), we see that G has property $P_n^{\mathcal{F}}$. Therefore, by (5),

$$\chi_n \equiv \#\left\{ G : \begin{array}{c} \text{orb}(G) = 0 \text{ and } G \text{ is nontrivial,} \\ \text{invariant, and does not satisfy } P_n^{\mathcal{F}} \end{array} \right\} \pmod 2. \tag{7}$$

Suppose we take such a G with $\text{orb}(G) = 0$ and collapse all its left vertices into one vertex which we connect to every vertex in $\mathcal{R}(G)$ and to no others, thereby yielding a graph $\hat{G}$. This gives a bijection from n-vertex invariant graphs G with $\text{orb}(G) = 0$ to $(n - T + 1)$-vertex graphs.

It is clear that if $\hat{G}$ has property $P_{n-T+1}^{\mathcal{F}}$, then G has property $P_n^{\mathcal{F}}$. Now suppose G has property $P_n^{\mathcal{F}}$ and let $F \in \mathcal{F}$ be a subgraph of G. Since $\text{orb}(G) = 0$, the vertices in F_{left} form an independent set; thus we may color them all with one color and then color each remaining vertex of F with a distinct color different from the one just used. This coloring produces a compression $\hat{F} \lhd F$ which clearly is a subgraph of $\hat{G}$. Since $\mathcal{F}$ is closed under compression, we have $\hat{F} \in \mathcal{F}$ and so $\hat{G}$ has property $P_{n-T+1}^{\mathcal{F}}$. Thus our bijection respects the relevant property and this completes the proof. □

We now have all the pieces needed for the following proof.

Proof of Lemma 3.2. Set $n = T = 2^{2^t}$. The only way for an n-vertex graph to have $\text{orb}(G) = 0$ is for it to have no edges. Using (7), this implies $\chi_T \equiv 0 \pmod 2$. Invoking Lemma 3.5 completes the proof. □

4. Proof of the main theorem. We now return to proving Theorem 1.1. According to the theorem's hypotheses

$$n = \sum_{i=1}^{r} q^{\alpha_i}, \tag{8}$$

where q is a prime power, $q \geq |H|$, each $\alpha_i \geq 1$, and $r \equiv 1 \pmod{r_0}$. Our goal is to show that Q_n^H is evasive under these hypotheses for some choice of r_0.

The chief difficulty in applying the topological approach outlined in section 2 lies in having to construct a group action natural enough for the property under consideration and satisfying the stringent conditions on the underlying group necessary for Theorem 2.3 to apply. In this section we shall come up with a group action that allows us to "merge together" big clusters of vertices in our graph, in the process changing the property under consideration from Q_n^H to $P_r^{\mathcal{F}}$ for some family $\mathcal{F}$ of graphs, r being as in (8).

We partition the vertex set of our n-vertex graph into clusters $V_1, \ldots, V_r$, with $|V_i| = q^{\alpha_i}$, and identify vertices in V_i with elements of the finite field $\mathbb{F}_{q^{\alpha_i}}$. Define a

permutation group Γ on the vertices as follows:

$$\Gamma = \{\langle a, b_1, b_2, \ldots, b_r\rangle \ : \ a \in \mathbb{F}_q^*,\ b_i \in \mathbb{F}_{q^{\alpha_i}}\}\ , \tag{9}$$

where $\langle a, b_1, b_2, \ldots, b_r\rangle$ denotes a permutation which sends $x \in V_i = \mathbb{F}_{q^{\alpha_i}}$ to $ax + b_i \in V_i$. Let $\Gamma_1 = \{\langle 1, b_1, \ldots, b_r\rangle \ : \ b_i \in \mathbb{F}_{q^{\alpha_i}}\}$. It is easy to check that Γ_1 is a normal subgroup of Γ, $|\Gamma_1| = q^{\alpha_1+\cdots+\alpha_r}$, a prime power, and $\Gamma/\Gamma_1 \cong \mathbb{F}_q^*$, a cyclic group. Thus Γ satisfies the hypotheses of Theorem 2.3.

As in section 3, the action of Γ induces a group action on the edges and thus partitions the edges into orbits. Let $\mathcal{A}$ denote the set of these orbits and let $\Delta = \Delta(Q_n^H)$ denote the abstract complex associated with property Q_n^H. Define a complex Δ_Γ on $\mathcal{A}$ as in (2):

$$\Delta_\Gamma = \left\{\mathcal{D} \subseteq \mathcal{A} \ : \ \bigcup_{A\in\mathcal{D}} A \ \in \Delta\right\}. \tag{10}$$

Our intention is to show that the Euler characteristic $\chi(\Delta_\Gamma) \neq 1$. By Theorem 2.3, evasiveness of Q_n^H will follow. To this end, let us investigate what the faces of Δ_Γ look like. Call an edge an *intracluster edge* if both its end points lie in the same V_i for some i; otherwise, call the edge an *intercluster edge*.

LEMMA 4.1. *An orbit containing an intracluster edge is not contained in any face of* Δ_Γ.

Proof. Let $A \in \mathcal{A}$ be the orbit of the intracluster edge (u, v), $u, v \in V_i$. Then $A = \{(au + b, av + b) \ : \ b \in \mathbb{F}_{q^{\alpha_i}},\ a \in \mathbb{F}_q^*\}$. Set $w = v - u$. Then $(0, w) \in A$. Consider the set of vertices $X = \{wz \ : \ z \in \mathbb{F}_q\}$. For $0 \neq x \in X$ we clearly have $(0, x) \in A$. Thus for any pair of distinct vertices $x_1, x_2 \in X$, we have $(0, x_2 - x_1) \in A$, whence $(x_1, x_2) \in A$. So A contains all edges among vertices in X. Since $|X| = q \geq |H|$, the orbit A contains H as a subgraph. By definition, Δ cannot contain a face that includes A and so no face of Δ_Γ can contain A. ∎

If $u \in V_i, v \in V_j, i < j$, then the orbit of the intercluster edge (u, v) is the set E_{ij} of all edges between V_i and V_j. Let $\mathcal{E} = \{E_{ij} |\ i < j\} \subseteq \mathcal{A}$. From the preceding lemma and (10) it is clear that

$$\Delta_\Gamma = \left\{\mathcal{D} \subseteq \mathcal{E} \ : \ \bigcup_{A\in\mathcal{D}} A \ \in \Delta\right\}. \tag{11}$$

Let $\mathcal{D}$ be any subset of $\mathcal{E}$. Then $G_\mathcal{D} = \bigcup_{A\in\mathcal{D}} A$ is a graph on n vertices with no intracluster edges and such that if $i \neq j$, the edges between V_i and V_j are either all present or all absent. Define a graph $\hat{G}_\mathcal{D}$ on r vertices $v_1, \ldots, v_r$ such that (v_i, v_j) is an edge iff all edges between V_i, V_j are present in $G_\mathcal{D}$.

Let $\mathcal{T}_H$ denote the family of all graphs $\hat{H}$ such that $\hat{H} \lhd H$. It is easy to check that $\mathcal{T}_H$ is closed under compression (refer to Definition 3.1). The following lemma is simple to prove and connects this section with section 3.

LEMMA 4.2. *H is a subgraph of $G_\mathcal{D}$ iff there is a $\hat{H} \in \mathcal{T}_H$ such that $\hat{H}$ is a subgraph of $\hat{G}_\mathcal{D}$. In other words, $G_\mathcal{D}$ satisfies Q_n^H iff $\hat{G}_\mathcal{D}$ satisfies $P_r^{\mathcal{T}_H}$.*

Proof. Suppose H is a subgraph of $G_\mathcal{D}$. Consider the following coloring of $G_\mathcal{D}$: all vertices in a cluster are colored the same and no two clusters use the same color. This is a valid coloring since each cluster of vertices is an independent set. This coloring induces a coloring of H which in turn induces a compression $\hat{H} \lhd H$. Clearly, this $\hat{H}$ is a subgraph of $\hat{G}_\mathcal{D}$.

Now suppose $\hat{H} \lhd H$ is a subgraph of $\hat{G}_{\mathcal{D}}$. Consider the graph H_1 with vertices in $\cup_{i=1}^{r} V_i$ formed by taking all edges in E_{ij} whenever v_i and v_j are adjacent in $\hat{H}$. Since each $|V_i| \geq q \geq |H|$, a straightforward argument shows that H is a subgraph of H_1, and therefore of $G_{\mathcal{D}}$. □

We are ready to prove our main theorem.

Proof of Theorem 1.1. Suppose Q_n^H is not evasive. From Theorem 2.3, we have $\chi(\Delta_\Gamma) = 1$. If $r = 1$, there is only one cluster, so by Lemma 4.1 we have $\Delta_\Gamma = \{\emptyset\}$, whence $\chi(\Delta_\Gamma) = 0$, a contradiction. Therefore $r > 1$. Equation (11) and Lemma 4.2 imply that there is a one-to-one correspondence between faces of Δ_Γ and nontrivial r-vertex graphs not satisfying property $P_r^{\mathcal{T}_H}$. Hence the abstract complex Δ_Γ is the same as the abstract complex $\Delta_r^{\mathcal{T}_H}$ defined in section 3. It follows from the definition of compression that $\mathcal{T}_H$ contains the complete graph on $\mathrm{chr}(H)$ vertices and contains no smaller graph. Therefore, (3) yields $t = \lceil \lg \lg \mathrm{chr}(H) \rceil$. Setting $r_0 = 2^{2^t} - 1$ and applying Lemma 3.2 we have $\chi(\Delta_r^{\mathcal{T}_H}) \neq 1$ and so $\chi(\Delta_\Gamma) \neq 1$, a contradiction. □

5. Consequences and extensions. Our techniques enable us to prove certain results with "cleaner" statements than our main Theorem 1.1; we prove four such results below. The first two are simple corollaries of Theorem 1.1 while the other two can easily be proved using the machinery of its proof. Finally, we present an interesting generalization of our main theorem.

THEOREM 5.1. *For any graph H there exist infinitely many primes p with the following property: for all sufficiently large n divisible by p, the property Q_n^H is evasive.*

Remark. Note that this establishes the evasiveness of Q_n^H for an arithmetic progression of values of n.

Proof. Choose an integer t such that $T = 2^{2^t}$ is at least $|H|$. By Dirichlet's theorem there exist infinitely many primes p such that $p \equiv 2 \pmod{T-1}$. Fix one such $p \geq T$ and pick any $n \geq p^2(T-1)$ divisible by p. Now $p-1$ is relatively prime to $T-1$; therefore there is an integer x such that $x(p-1) \equiv n/p - 1 \pmod{T-1}$ and $0 \leq x < T-1$. From the lower bound on n we have $n/p - px > 0$. Therefore we can write

$$n = \sum_{i=1}^{x} p^2 + \sum_{i=1}^{n/p-px} p,$$

which is an expression of n as a sum of powers of p. The number of summands in this expression is $x + n/p - px \equiv 1 \pmod{T-1}$. Since $p \geq T \geq |H|$, we can apply Theorem 1.1 to conclude that Q_n^H is evasive. □

COROLLARY 5.2. *For any graph H there exists a constant $c = c(H)$ such that for all sufficiently large n, the decision tree complexity of Q_n^H is at least $\frac{1}{2}n^2 - cn$.*

THEOREM 5.3. *If the graph H is bipartite, then Q_n^H is evasive for all sufficiently large n.*

Proof. Since $\mathrm{chr}(H) = 2$, in the proof of Theorem 1.1, using the notation of that proof, we may take $t = 0$ which gives $r_0 = 1$. The condition $r \equiv 1 \pmod{r_0}$ is now trivially satisfied. The condition on n becomes a simple requirement that n be divisible by a prime power $q \geq |H|$. However, if n is sufficiently large, then it clearly satisfies this condition. □

THEOREM 5.4. *Let $\mathcal{M}$ be an infinite minor-closed family of graphs that does not include all graphs. For n-vertex graphs, let $R_n^{\mathcal{M}}$ be the property of being in $\mathcal{M}$. Then $R_n^{\mathcal{M}}$ is evasive for all sufficiently large n.*

Remark. Planarity was already known to be evasive [1]. This result is a major generalization. However, it is not the strongest possible generalization to "minor-closed" properties, since planarity has been proven evasive whenever n is large enough for it to be a nontrivial property (i.e., $n \geq 5$).

Proof. Let H be a graph not in $\mathcal{M}$ with minimum size and let $h = |H|$. Then H is a minor of both the complete graph K_h and the complete bipartite graph $K_{h,h}$; therefore no graph in $\mathcal{M}$ can contain either K_h or $K_{h,h}$ as a subgraph.

Suppose n is divisible by a prime power $q \geq h$, a condition that always holds if n is sufficiently large. Following the argument of section 4 we divide the labeled vertices of the candidate graph G into clusters of size q and consider the orbits of the edges created by the action of the group Γ described there. Let Δ be the abstract complex associated *with the negation*[3] *of* $R_n^{\mathcal{M}}$. An orbit containing an intracluster edge cannot be included in a face of Δ_Γ because its edges, if present, would create a K_q subgraph. An orbit containing an intercluster edge cannot be included either because its edges, if present, would create a $K_{q,q}$ subgraph. Thus, $\Delta_\Gamma = \{\emptyset\}$ and so $\chi(\Delta_\Gamma) = 0 \neq 1$. By Theorem 2.3, the negation of $R_n^{\mathcal{M}}$ is evasive and therefore so is $R_n^{\mathcal{M}}$. □

The next theorem generalizes our main theorem and can be proved essentially using the same argument as that for the main theorem.

THEOREM 5.5. *Let $f : \{0,1\}^k \to \{0,1\}$ be a nontrivial monotone boolean function and let $H_1, \ldots, H_k$ be arbitrary graphs. Define the composite property $Q_n = f(Q_n^{H_1}, \ldots, Q_n^{H_k})$. Then there exists an integer r_0 with the following property. Suppose $n = \sum_{i=1}^{r} q^{\alpha_i}$, where q is a prime power, $q \geq \max_{1 \leq i \leq k} |H_i|$, each $\alpha_i \geq 1$, and $r \equiv 1 \pmod{r_0}$. Then Q_n is evasive.*

Remark. This theorem shows, for instance, that properties like "G either contains H_1 as a subgraph or else contains both H_2 and H_3 as subgraphs" are evasive for several values of n. This theorem has corollaries similar to Theorem 5.1 and Corollary 5.2.

6. Concluding remarks. The major open question in the area of decision tree complexity of graph properties is to settle the Karp Conjecture. The pioneering work of Kahn, Saks, and Sturtevant [4] has given us a possible direction to follow in attempting to settle this conjecture. Our work takes steps in this direction by extending their topological approach to prove stronger results for a fairly general class of graph properties.

An obvious open question raised by our work is "How far can one enlarge the set of values of n for which our results hold?" We conjecture that in the notation of section 3, we have $\chi_n \neq 1$ for large enough n. If proved true, this conjecture would remove all number theoretic restrictions in the main theorem.

Acknowledgments. We would like to express our sincere thanks to Professors Sanjeev Arora, Bernard Chazelle, and Andrew Yao for their valuable comments and suggestions.

REFERENCES

[1] M.R. BEST, P. VAN EMDE BOAS, AND H.W. LENSTRA, JR., *A Sharpened Version of the Aanderaa-Rosenberg Conjecture*, Report ZW 30/74, Mathematisch Centrum, Amsterdam, The Netherlands, 1974.

[2] B. BOLLOBÁS, *Complete subgraphs are elusive*, J. Combin. Theory Ser. B, 21 (1976), pp. 1–7.

[3] B. BOLLOBÁS, *Extremal Graph Theory*, Academic Press, London, New York, 1978.

[3] Notice that the property $R_n^{\mathcal{M}}$ is not monotone. However, its negation is monotone. Clearly, a property is evasive if its negation is.

[4] J. Kahn, M. Saks, and D. Sturtevant, *A topological approach to evasiveness*, Combinatorica, 4 (1984), pp. 297–306.
[5] E.C. Milner and D.J.A. Welsh, *On the computational complexity of graph theoretical properties*, in Proceedings of the 5th British Combinatorial Conference, Aberdeen, Scotland, 1975, C. St. J.A. Nash-Williams and J. Sheehan, eds., 1976, pp. 471–487.
[6] R.L. Rivest and J. Vuillemin, *On recognizing graph properties from adjacency matrices*, Theoret. Comput. Sci., 3 (1976), pp. 371–384.
[7] A.L. Rosenberg, *On the time required to recognize properties of graphs: A problem*, SIGACT News, 5 (1973), pp. 15–16.
[8] E. Triesch, *Some results on elusive graph properties*, SIAM J. Comput., 23 (1994), pp. 247–254.
[9] E. Triesch, *On the recognition complexity of some graph properties*, Combinatorica, 16 (1996), pp. 259–268.
[10] A.C.-C. Yao, *Monotone bipartite graph properties are evasive*, SIAM J. Comput., 17 (1988), pp. 517–520.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 876–886

COMPRESSIBILITY AND RESOURCE BOUNDED MEASURE*

HARRY BUHRMAN† AND LUC LONGPRÉ‡

Abstract. We give a new definition of resource bounded measure based on compressibility of infinite binary strings. We prove that the new definition is equivalent to the one commonly used. This new characterization offers us a different way to look at resource bounded measure, shedding more light on the meaning of measure zero results and providing one more tool to prove such results. The main contribution of the paper is the new definition and the proofs leading to the equivalence result. We then show how this new characterization can be used to prove that the class of linear autoreducible sets has p- measure 0. We also prove that the class of sets that are truth-table reducible to a p-selective set has p-measure 0 and that the class of sets that Turing reduce to a subpolynomial dense set has p-measure 0. This strengthens various results.

Key words. resource bounded measure, Kolmogorov complexity, compressibility

AMS subject classifications. 28A99, 68Q15, 68Q30

PII. S0097539797317123

1. Introduction.

While Lebesgue measure has been used in mathematics since the previous century, and while it has been used earlier this century to study randomness in infinite strings [19], its notable appearance in complexity theory is in the formalization of "random oracles." Bennett and Gill [5] showed, for example, that relative to a random oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A$. The statement about the random oracle is formalized as follows: the class of sets A such that $\mathrm{P}^A = \mathrm{NP}^A$ has Lebesgue measure zero.

When dealing with uncountable classes, measure zero is an intuitively appealing concept to formalize the idea that sets having a certain property are rare. But the concept seems to fall apart when dealing with countable classes, since all of these have Lebesgue measure zero. For example, how would we formalize the statement "most recursive oracles separate P from NP"?

In order to formalize this kind of statement, Lutz introduced resource bounded measure [14]. A more useful definition based on resource bounded martingales appeared in [15]. With resource bounded measure, one is able to formally state results of the type "most languages in class $\mathcal{C}$ have property P." These notions turned out to be quite useful in complexity, as witnessed by a stream of results in recent years (for example, see [9, 10, 16, 18, 17, 3, 4, 1]).

In this paper, we offer a different definition of resource bounded measure, equivalent to that of Lutz in [15]. We say that a set is $t(n)$-compressible if its characteristic sequence can be compressed and uncompressed in time $t(n)$. The precise definition is given in section 4. A class of sets has p-measure zero if all the sets in the class are n^k-compressible for some fixed k. This new characterization has several advantages.

*Received by the editors March 3, 1997; accepted for publication (in revised form) November 20, 2000; published electronically January 11, 2002.

http://www.siam.org/journals/sicomp/31-3/31712.html

†CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands (buhrman@cwi.nl). This author was partially supported by the Dutch foundation for scientific research (NWO) through NFI Project ALADDIN, under contract number NF 62-376.

‡Computer Science Department, University of Texas at El Paso, El Paso, TX 79968 (longpre@cs.utep.edu). This author was supported in part by NSF grant CCR-9211174 and NSF grant INT-9123551.

(i) This new characterization may allow more intuitive proofs of results about resource bounded measure.

(ii) A result claiming that a class of set does not have p-measure zero is usually seen as an abundance result. How should this abundance be interpreted? The new characterization explains in a precise way what is meant: the sets in the class cannot all be compressed with a fixed polynomial-time bound.

(iii) While the martingale characterization was not directly applicable to classes below E, Mayordomo [21] gave a definition applicable to PSPACE, and Allender and Strauss [1] gave a definition applicable to P and other subexponential classes. Further related work on this can be found in [24, 8]. Although similar technical problems arise with our definition, it may offer other alternatives for defining measures applicable to subexponential classes.

A corollary of the proof of equivalence is as follows: a class $\mathcal{C}$ has p-measure zero in E if and only if all the sets in $\mathcal{C}$ are n^k-compressible for some fixed k. The n^k-compressible sets form a proper hierarchy, and E is not included in any fixed level of that hierarchy. Moreover, if we define $\mathrm{Comp}(n^k)$ as the class of sets in E that are n^k-compressible, then $\mathrm{E} = \cup_k \mathrm{Comp}(n^k)$. So the abundance meaning can be understood as follows: a class of sets X has p-measure 0 in E if and only if $X \cap \mathrm{E}$ is included in a fixed level of that hierarchy, while the hierarchy is itself infinite.

It should be noted that equivalence between a classical constructive measure and a definition based on Kolmogorov complexity has been studied in the context of random sequences. Martin-Löf's definition of random sequences [19] based on constructive measure is equivalent to a subsequent definition using incompressibility in the sense of a version of prefix Kolmogorov complexity due to Levin [12]. See also Schnorr [22] for a similar theorem. We refer the reader to the book by Li and Vitányi [13] for a more detailed account.

There is also a way to define compressibility in the nonuniform context and prove that Lebesgue measure zero is equivalent to that kind of compressibility [11].

Next we will use the new characterization to prove the following results:

(i) The class of $c \cdot n$ autoreducible sets has p-measure 0.

(ii) The class of sets that are polynomial-time truth-table reducible to a p-selective set has p-measure 0. It follows immediately that there is no p-selective set that is hard for E under polynomial-time truth-table reductions.

(iii) The class of sets that are Turing reducible to a set with subpolynomial density has p-measure 0. This strengthens the results in [6].

2. Preliminaries. Let $\Sigma = \{0, 1\}$. Strings are elements of $\Sigma^*$and are denoted by lower case letters $x, y, u, v, \ldots$. Infinite strings are elements of Σ^∞ and are denoted by lower case Greek letters. The empty string is λ. For any string x, the length of a string is denoted by $|x|$, $x[i..j]$ is the substring of x from index i to index j inclusively, $x[i]$ stands for $x[i..i]$, $x = x[0..|x| - 1]$, and if $j < i$, then $x[i..j]$ is λ. For two strings x and y, $x \sqsubseteq y$ if y is an extension of x. Subsets of Σ^* are denoted by capital letters $A, B, C, S, \ldots$. The set $\Sigma^* - A$ is denoted by $\overline{A}$. The complement of a class of sets X is $X^c = \{A \subseteq \Sigma^* | A \notin X\}$. For a set A we use $A^{=n}(A^{\leq n})$ to denote the subset of all strings in A having length equal to n ($\leq n$, resp.). We use χ_A to denote the infinite binary string where bit i is 1 if and only if string $x_i \in A$, for a standard enumeration $\{x_i\}$ of finite binary strings. With any infinite binary string, we can associate a real between 0 and 1 through the natural binary expansion. This mapping is one-to-one except on multiples of powers of 2 (dyadic rationals or dyadic numbers). With each finite binary string x, we can associate an interval I_x

that corresponds to the set of all reals associated with possible infinite extensions of x. In this paper, intervals over the reals are half-open intervals, so I_x excludes its right boundary. For an interval I over the reals, $\mu(I)$ is the length of the interval. For any set A, the cardinality of A is denoted by $\|A\|$. We define C_w, the cylinder generated by w, as the class of languages $\{\alpha \in \Sigma^\infty | w \sqsubseteq \alpha\}$. We fix a pairing function $\lambda xy.\langle x, y\rangle$ that is computable in polynomial time from $\Sigma^* \times \Sigma^*$ to Σ^*. Without loss of generality, we assume that the pairing function respects the length of its arguments (i.e., $|x| + |y| \leq |\langle x, y\rangle| \leq 2(|x| + |y|)$). We assume that the reader is familiar with the standard Turing machine model. The class p is the class of all polynomials.

3. Resource bounded measure. We use the definition of resource bounded measure based on martingales.

Let $D = \{m2^{-n} | m, n \in N\}$ be the set of nonnegative dyadic rationals.

DEFINITION 1. *A* martingale *is a function $d : \Sigma^* \to D$ with the property that, for all $w \in \Sigma^*$,*

$$d(w) = \frac{d(w0) + d(w1)}{2}.$$

DEFINITION 2. *A martingale* succeeds *on language $A \subseteq \Sigma^*$ if*

$$\limsup_{n\to\infty} d(\chi_A[0..n-1]) = \infty.$$

A martingale d is p-computable, and we call it a p-martingale, if $d(w)$ can be computed in time polynomial in $|w|$.

The intuition behind this definition is a game where a player is trying to predict the next bit by looking at all the bits that have been produced so far. The player starts with an initial capital $d(\lambda)$ and can decide to bet an amount of money which is at most the current capital. If the predicted bit is correct, the capital increases by the amount bet, and if it's incorrect, the capital decreases by that same amount. The function d models the current capital of the betting strategy after having seen a finite binary string.

Lutz [15] defined a martingale to give a real value and required computations of the martingale to approximate the real value by using dyadic rationals. But, as shown independently by Mayordomo [21] and by Juedes and Lutz [10], defining the martingale directly with the dyadic rationals provides an equivalent definition of resource bounded measure. Moreover, without loss of generality, we may assume that $d(w)$ is a dyadic number $m2^{-n}$ such that $n < |w|$.

DEFINITION 3. *A class X of languages has p-measure 0, and we write $\mu_p(X) = 0$, if there is a p-martingale d that succeeds on every element of X.*

DEFINITION 4. *A class X of languages has p-measure 1, and we write $\mu_p(X) = 1$, if $\mu_p(X^c) = 0$.*

4. Compressibility. Compressibility of finite strings is usually defined using Kolmogorov complexity. There are various problems in defining compressibility of infinite strings in terms of the Kolmogorov compressibility of its prefixes, mainly because no string has all of its prefixes completely incompressible. We define compressibility of an infinite string by using another infinite string that can generate it, with the compressibility calculated as the amount of the prefix of the compressed string that is needed to reproduce a prefix of the uncompressed string. For the time bounded version, we also need that the compressed string can be efficiently computed, at least in some weak sense.

Results in this paper are in terms of p-measure, which is appropriate for studying abundance with respect to E. The obvious extension to p_2, the class of functions of the form $2^{\log^c n}$ and EXP also holds.

DEFINITION 5. *An infinite string $\omega \in \{0,1\}^\infty$ is f-compressible if $\exists \kappa \in \{0,1\}^\infty$ such that the following conditions hold.*

1. *(Uncompression) There is a Turing machine M that, given $\kappa[0..j]$, outputs a prefix $\omega[0..i]$ of ω in time at most $f(i+j)$, such that the value $i-j$ is not bounded by any constant.*

2. *(Compression) There is a Turing machine M' that, given $\omega[0..i]$, uses at most $f(i)$ time to output a finite number of strings, one of which is a prefix $\kappa[0..j']$ such that M, on input $\kappa[0..j']$, outputs a prefix of ω that is a proper extension of $\omega[0..i]$ consistent with ω.*

(Note: We could also add the restriction have that $i-j'$ is not bounded by any constant. By the composition of uncompression and compression, it would turn out to be an equivalent definition.)

We show in section 5 (Theorem 7) that

$$\mu_p(\mathcal{C}) = 0 \;\Rightarrow\; (\exists f \in p)(\forall A \in \mathcal{C})\;\; \chi_A \text{ is } f\text{-compressible},$$

and in section 6 (Theorem 9) that

$$(\exists f \in p)(\forall A \in \mathcal{C})\;\; \chi_A \text{ is } f\text{-compressible} \Rightarrow \mu_p(\mathcal{C}) = 0.$$

These two theorems together show that the definition of p-measure zero in terms of our compressibility definition is equivalent to that of Lutz in [15].

5. Measure zero implies compressibility. We have defined measure zero using basic martingales. We will need a few more properties of the martingales to prove that measure zero implies compressibility. The following shows that such special properties can be assumed without loss of generality.

LEMMA 6. *If $\mu_p(\mathcal{C}) = 0$, then there exists a p-martingale d that succeeds on every element of $\mathcal{C}$ and satisfies the following properties:*

1. $d(\lambda) = 1$,
2. *$d(x)$ is a dyadic number $m2^{-n}$ such that $n \le |x| + 4$,*
3. *$(\forall x, y)\ d(x)/4 \le d(xy)$, where y is a finite binary string,*
4. *$(\forall x, b)\ d(xb) \le (7/4)d(x)$, where b is a bit.*

Proof. Let d be a martingale witnessing that $\mu_p(\mathcal{C}) = 0$. Define a martingale d' in the following way. Given a finite string x, let $i_0 = -1$, and for $k \ge 1$, let i_k be the smallest integer, if it exists, such that

$$\frac{d(x[0..i_k])}{d(x[0..i_{k-1}])} \ge 2.$$

Define

$$\begin{aligned} d'(\lambda) &= 1, \\ d'(x) &= d'(x[0..i_k]) - 1/4 + (1/4)d(x)/d(x[0..i_k]), \\ &\qquad \text{where } k \text{ is the largest integer such that} \\ &\qquad x[0..i_k] \text{ is a proper prefix of } x. \end{aligned}$$

Informally, d' starts with \$1.00 on λ, and its betting strategy on successively longer prefixes of the characteristic function of a set A is to keep part of its capital

frozen and use the rest to bet in proportion with the d strategy. The amount of frozen capital is revised each time d has doubled its capital. At that time, all the capital is frozen except for \$0.25, which is kept for betting. Each time d is doubled, d' earns \$0.25. If d is successful on A, the doubling occurs infinitely often, so d' is unbounded as well, and is thus successful.

Although d' does not meet property 2 of the lemma, we show next that it meets stronger versions of properties 3 and 4:

(i) $(\forall x, y)\ (3/5)d'(x) \leq d'(xy)$, where y is a finite binary string,

(ii) $(\forall x, b)\ d'(xb) \leq (7/5)d'(x)$, where b is a bit.

To see this, let k_1 be the largest integer such that $xy[0..i_{k_1}]$ is a (not necessarily proper) prefix of xy, where indices i_k are defined as above. Similarly, let k_2 be the largest integer such that $x[0..i_{k_2}]$ is a (not necessarily proper) prefix of x. If $xy = xy[0..i_{k_1}]$, then the first equality below holds trivially. Otherwise, it holds by the definition of d'. A similar argument goes for the second equality:

$$\begin{aligned} d'(xy) &= d'(xy[0..i_{k_1}]) - 1/4 + (1/4)d(xy)/d(xy[0..i_{k_1}]), \\ d'(x) &= d'(xy[0..i_{k_2}]) - 1/4 + (1/4)d(x)/d(xy[0..i_{k_2}]). \end{aligned}$$

Now, notice that for any x, the function $f(k) = d(x[0..i_k])$ is monotonically increasing, so $d(x[0..i_{k_1}]) \geq 1$. Notice also that $d'(xy[0..i_{k_2}]) \leq d'(xy[0..i_{k_1}])$ because $i_{k_2} \leq i_{k_1}$ and by monotonicity. Finally, notice that $d(x)/d(x[0..i_{k_2}]) < 2$, because if it were ≥ 2, then k_2 would not, by our definition, be the largest of indices i_k. With these observations, we can derive

$$\begin{aligned} d'(xy) - (3/5)d'(x) &= d'(xy[0..i_{k_1}]) - 1/4 + (1/4)d(xy)/d(xy[0..i_{k_1}]) \\ &\quad -(3/5)d'(xy[0..i_{k_2}]) + 3/20 - (3/20)d(x)/d(x[0..i_{k_2}]) \\ &\geq d'(xy[0..i_{k_2}]) - 1/4 + (1/4)d(xy)/d(xy[0..i_{k_1}]) \\ &\quad -(3/5)d'(xy[0..i_{k_2}]) + 3/20 - (3/20)d(x)/d(x[0..i_{k_2}]) \\ &= (2/5)d'(xy[0..i_{k_2}]) - 1/4 + (1/4)d(xy)/d(xy[0..i_{k_1}]) \\ &\quad +3/20 - (3/20)d(x)/d(x[0..i_{k_2}]) \\ &\geq 2/5 - 1/4 + 0 + 3/20 - (3/20) * 2 \\ &= 0. \end{aligned}$$

The other property can be proved in a similar fashion. Let k_1 be the largest integer such that $x[0..i_{k_1}]$ is a (not necessarily proper) prefix of x. Then

$$\begin{aligned} d'(xb) &= d'(xb[0..i_{k_1}]) - 1/4 + d(xb)/4d(x[0..i_{k_1}]), \\ d'(x) &= d'(xb[0..i_{k_1}]) - 1/4 + d(x)/4d(x[0..i_{k_1}]), \end{aligned}$$

and we can conclude that

$$\begin{aligned} d'(xb) - (7/5)d'(x) &= d'(xb[0..i_{k_1}]) - (7/5)d'(xb[0..i_{k_1}]) - 1/4 + 7/20 \\ &\quad +d(xb)/4d(x[0..i_{k_1}]) - (7/20)d(x)/d(x[0..i_{k_1}]) \\ &\leq -(2/5)d'(xb[0..i_{k_1}]) + 1/10 + 2d(x)/4d(x[0..i_{k_1}]) \\ &\quad -(7/20)d(x)/d(x[0..i_{k_1}]) \\ &\leq -(2/5) + 1/10 + (3/20)d(x)/d(x[0..i_{k_1}]) \\ &\leq -(2/5) + 1/10 + 3/10 \\ &= 0. \end{aligned}$$

Now, define martingale d'' as follows. Define $d''(\lambda) = 1$. If $d'(x) > d'(\tilde{x})$, where $\tilde{x}$ is x where the last bit is flipped from 0 to 1 or from 1 to 0, then $d''(x)$ is the value $d'(x)$ approximated downwards to the closest dyadic number of the form $m2^{-n}$, where $n \leq |x| + 4$. Otherwise, it is approximated upwards. The loss of capital for d'' compared to d' is at most $2^{-|x|-4}$ when betting on x, which totals to at most $\Sigma_{i=1}^{\infty} 1/2^{i+4} = 1/16$ cumulatively. Also, because $d'(\lambda) = 1$ for all y, $d'(y) \geq 3/5$, so for all y, $(15/16)d'(y) \leq d''(y) \leq (17/16)d'(y)$. Combining this and the property of d' above, we get all properties 1, 2, 3, and 4 of the lemma. □

THEOREM 7. $\mu_p(\mathcal{C}) = 0 \Rightarrow (\exists f \in p)(\forall A \in \mathcal{C})$ χ_A *is* f*-compressible.*

Proof. Assume that $\mu_p(\mathcal{C}) = 0$ and let d be a p-martingale as in Lemma 6. Let $A \in \mathcal{C}$ and let $\omega = \chi_A$. If A is finite or cofinite, then χ_A is obviously f-compressible, so without loss of generality, assume A is neither finite nor cofinite. We need an infinite string κ to encode ω. We can interpret ω as the encoding of a real in the half-open interval $[0, 1)$. In the following arguments, all intervals have a boundary at dyadic numbers. Since we assume A is neither finite nor cofinite, ω is not the binary expansion of a dyadic number, so its binary expansion will never fall on the extremity of any interval. In a standard encoding, all infinite strings starting with 0 encode reals in the first half of the interval, and strings starting with 1 encode reals in the right half of the interval. We will create an encoding scheme that possibly moves this halfway border. If $d(0) > d(1)$, then the binary expansion of reals in an interval larger than $1/2$ will be used to encode reals between 0 and 0.5. Similarly, if $d(x0) > d(x1)$, then the size of the interval reserved to encode reals that extend $x0$ will be larger than that of extensions of $x1$. The idea is to keep an interval of reals to encode extensions such that the size of the interval is proportionally related to the current capital with the current betting strategy of the martingale. Intuitively, large intervals can be described with few bits, so a winning strategy will result in compression.

More formally, let g be a function from finite binary strings to half-open intervals in $[0, 1)$ defined as follows:

$$\begin{aligned} g(\lambda) &= [0, 1), \\ g(x0) &= \text{the left part of } g(x) \text{ of size } d(x0)/2^{|x0|}, \\ g(x1) &= g(x) - g(x0). \end{aligned}$$

The following lemma has a straightforward proof by induction on the length of the strings. The proof is omitted.

LEMMA 8. $\mu(g(x)) = d(x)/2^{|x|}$.

Since $g(xb)$ is a subinterval of $g(x)$ and since property 4 of Lemma 6 ensures that $\lim_{n\to\infty} \mu(g(\omega[0..n])) = 0$, an infinite sequence can be associated with the real r defined by $r = \lim_{n\to\infty} g(\omega[0..n])$. Since ω is not a dyadic number, r is not a dyadic number, and hence r has a unique binary expansion. Let κ be the binary expansion of r. We now have to show that κ is a valid compression of ω.

To generate $\omega[0..i]$ from $\kappa[0..j]$, simulate the martingale starting at λ on successively longer strings. Suppose we have generated the string x so far. If $g(x0)$ contains $I_{\kappa[0..j]}$, then append 0 to x. If $g(x1)$ contains $I_{\kappa[0..j]}$, then append 1 to x. Continue until $I_{\kappa[0..j]}$ is not contained in either of the intervals. At the end of this process, we have a string x such that $g(x)$ contains $I_{\kappa[0..j]}$. By the definition of κ, x is a prefix of ω, say $x = \omega[0..i]$.

To generate $\kappa[0..j]$ from $\omega[0..i]$, we compute the list of all possible strings that encode extensions of $\omega[0..i]0$ and of $\omega[0..i]1$. Let $I = g(\omega[0..i]0)$. Since $d(\omega[0..i]0)$ is a dyadic number $m2^{-n}$, where $n \leq |\omega[0..i]0| + 4 = i + 6$, and since $\mu(I) =$

$d(\omega[0..i]0)/2^{i+2}$, the borders of I can be expressed by dyadic numbers $m2^{-n}$, where $n \leq 2i+8$. The interval I can be covered exactly by a set of at most $2(2i+8)$ intervals each expressible as I_x for some x. This is done by starting from a point inside the interval which is expressible by a dyadic number $m2^{-n}$, where n is the smallest. Covering the part of I on the left of that point can be done with a set of at most $2i+8$ intervals because each interval chosen decreases in size by at least half and covers at least half of the remaining gap. Covering the part of I on the right of that point can be done with another set of at most $2i+8$ intervals. Interval $g(\omega[0..i]1)$ can be covered similarly. Output the set of all strings x such that I_x is one of the intervals needed for the above covering. All of them generate proper extensions of $\omega[0..i]$ via our uncompression algorithm, and one of them is a prefix of κ.

It remains to show that $i-j$ is not bounded by any constant, where i is the index of the last bit produced by the uncompression algorithm M on $\kappa[0..j]$. Select j such that M on $\kappa[0..j]$ produces $\omega[0..i]$ and such that M on $\kappa[0..j-1]$ produces a proper prefix of $\omega[0..i]$. Since M produces $\omega[0..i]$ on $\kappa[0..j]$, $g(\omega[0..i])$ includes $I_{\kappa[0..j]}$. Since M produces a proper prefix of $\omega[0..i]$ on $\kappa[0..j-1]$, $g(\omega[0..i])$ does not contain $I_{\kappa[0..j-1]}$. Interval $g(\omega[0..i])$ can be partitioned into 3 intervals: L, $I_{\kappa[0..j]}$, and R, where L (resp., R) corresponds to the reals in $g(\omega[0..i])$ that are smaller (resp., greater) than those in $I_{\kappa[0..j]}$. Assume that $\kappa[j]=0$. The case where $\kappa[j]=1$ is similar. Then, R is an interval included in $I_{\kappa[0..j-1]1}$ because $g(\omega[0..i])$ does not include $I_{\kappa[0..j-1]}$. So, $\mu(R) \leq \mu(\kappa[0..j])$. Because $g(\omega[0..i]0)$ intersects $I_{\kappa[0..j]}$, we can derive the following upper bound for the size of L:

$$\begin{aligned}
\mu(L) &\leq \mu(g(\omega[0..i]0)) \\
&= d(\omega[0..i]0)/2^{i+2} \\
&\leq (7/4)d(\omega[0..i])/2^{i+2} \\
&\leq 7d(\omega[0..i]1)/2^{i+2} \\
&= 7\mu(g(w[0..i]1)) \\
&\leq 7(\mu(I_{\kappa[0..j]}) + \mu(R)), \\
\mu(g(\omega[0..i])) &= \mu(L) + \mu(I_{\kappa[0..j]}) + \mu(R) \\
&\leq 8(\mu(I_{\kappa[0..j]}) + \mu(R)) \\
&\leq 16\mu(I_{\kappa[0..j]}).
\end{aligned}$$

But $\mu(g(\omega[0..i])) = d(\omega[0..i])/2^{i+1}$ and $\mu(I_{\kappa[0..j]}) = 1/2^{j+1}$, so we get

$$\begin{aligned}
d(\omega[0..i])/2^{i+1} &\leq 16/2^{j+1}, \\
d(\omega[0..i]) &\leq 2^{i-j+4}, \\
\log(d(\omega[0..i])) - 4 &\leq i-j.
\end{aligned}$$

Since the martingale is successful, $d(\omega[0..i])$ is unbounded and hence so is $i-j$. □

6. Compressibility implies measure zero. We now need to show the opposite implication to show the equivalence between compressibility and measure zero.

THEOREM 9. $(\exists f \in p)(\forall A \in \mathcal{C})$ χ_A *is* f*-compressible* $\Rightarrow \mu_p(\mathcal{C}) = 0$.

Proof. Let f be bounded by $n^k + k$, and assume that for each $A \in \mathcal{C}$, χ_A is f-compressible. For each $A \in \mathcal{C}$, we build below a martingale that succeeds on A and is $n^k + k$ time computable. Since $n^k + k$ time bounded martingales are easily enumerable for a fixed k, $\mathcal{C}$ is a p-union of p-measure 0 sets, so $\mu_p(\mathcal{C}) = 0$. (To build a p-union of p-measure 0 sets, we need to build a sequence of martingales $d_1, d_2, \ldots$

witnessing that the sets have p-measure 0 and a uniform algorithm that receives k and x as input and computes $d_k(x)$ in time polynomial in both k and x. See [15] for more details and a proof of this.)

Let $A \in \mathcal{C}$ be a set such that χ_A is f-compressible. Assume we have algorithms to compress $\omega = \chi_A$. Let's say algorithm B_1 (compress) computes κ from ω and algorithm B_2 (uncompress) computes ω from κ. We build a p-martingale. Suppose we are given $\omega[0..i]$. Let S_0 be $\{\lambda\}$. To form S_j from S_{j-1}, simulate B_1 on $\omega[0..j]$. From the strings output by B_1, keep only those that are extensions of strings in S_{j-1}. For each remaining string x, if $B_2(x)$ is not a proper extension of $\omega[0..j]$, then discard it. Otherwise, replace it by its smallest prefix y such that B_2 on y is a proper extension of $\omega[0..j]$. Call the set of remaining strings S_j.

Compute strings in S_i. Separate the strings in S_i into two groups, those that predict (via B_2) a 0 and those that predict a 1. Call these groups G_0 and G_1. Let each string vote for the next bit. The relative weight of each vote depends on the length of the string: the shorter the string, the more weight. Let $s_i = \Sigma_{x_k \in S_i} 2^{-|x_k|}$. Let $b_0 = \Sigma_{x_k \in G_0} 2^{-|x_k|}$. Let $b_1 = \Sigma_{x_k \in G_1} 2^{-|x_k|}$. Then $d(\omega[0..i]0) = d(\omega[0..i])(1 + (b_0 - b_1)/s_i)$ and $d(\omega[0..i]1) = d(\omega[0..i])(1 + (b_1 - b_0)/s_i)$.

We claim that for any i, $d(\omega[0..i]) \geq \sum_{x_k \in S_i} 2^{i-|x_k|} = 2^i s_i$. This can be proven by induction on i. The statement is true for $i = 0$, assuming initial capital of 1. Assume it's true up to i. At step $i+1$, without loss of generality, suppose $\omega[i+1] = 0$. Let $r = b_0/s_i$. Then, $d(\omega[1..i]0) = 2rd(\omega[0..i])$. This is greater than or equal to $2r2^i s_i = 2^{i+1} r s_i$, by the induction hypothesis. Thus, the proof of the claim is complete if we show that rs_i is greater than or equal to s_{i+1}. But rs_i is just b_0, and because strings in S_{i+1} are extensions of strings in G_0, b_0 is no smaller than s_{i+1}.

Now, let k be an arbitrary value, and let j be such that algorithm B_2 computes $\omega[0..i]$ from $\kappa[0..j]$ such that $i - j > k$. By the assumption on compressibility, such a j must exist. Our construction guarantees that $\kappa[0..j]$ is in S_{i-1}. By the claim above, $d(\omega[0..i-1]) \geq \sum_{x_k \in S_{i-1}} 2^{i-1-|x_k|} \geq 2^{i-1-|\kappa[0..j]|} \geq 2^{i-j-2} \geq 2^{k-2}$. This shows that the capital is unbounded, so the martingale succeeds. □

7. Applications of the new characterization. In this section we give some examples of how the new characterization can be applied. In each case we use Theorem 9 to prove that some class has p-measure zero. In other words, to show that a class has p-measure zero, we show that every set in the class can be f-compressed, where f is a polynomial not depending on the particular set being compressed.

DEFINITION 10. *A set A is* P-immune *if no infinite subset of A is in* P.

DEFINITION 11. *A set A is* P-bi-immune *if both A and $\overline{A}$ are* P*-immune.*

THEOREM 12 (see [20]). *The class of non-*P*-bi-immune sets has p-measure* 0.

Proof. Let A be an arbitrary non-P-bi-immune set and suppose there is an infinite set $C \subseteq A$ where C is in DTIME(n^k). (The case where $C \subseteq \overline{A}$ is analogous.) If $A \in P$, then χ_A can obviously be compressed. Otherwise, the compressed characteristic sequence of A is the concatenation of the bits $\chi_A(x)$ with $x \notin C$. Since C is infinite, the compression is unbounded.

To uncompress $\kappa[0..j]$ into $\chi_A[0..i]$, for each string x in lexicographic order, if $x \in C$, append 1, else append the next bit of κ. Continue until the bits of $\kappa[0..j]$ are exhausted. This requires $i+1$ tests of membership in C, for strings of length $O(\lg i)$. The total time is in $O(i \lg^k i) \subseteq O(i^2)$.

To compress $\chi_A[0..i]$, just remove the bits corresponding to elements of C. Then create two strings by adding a 0 and a 1. One of the two strings will uncompress into the correct extension of $\chi_A[0..i]$. This can also be done in time $O(i^2)$.

(From the time analysis, we can see that the theorem holds even for larger run times than P.) □

The next theorem investigates the class of sets that are autoreducible [2]. A set A is *autoreducible* if there is a polynomial-time oracle Turing machine that accepts A with A as oracle provided that, on input x, it never queries x to the oracle.

THEOREM 13. *For any fixed constant c, the class of sets that are autoreducible via oracle machines that query no more than $c \cdot |x|$ queries on input x has p-measure 0.*

Proof. Let A be an arbitrary set in the class mentioned in the statement of the theorem. Let $n_0 = 4$ and $n_{k+1} = n_k^{\log n_k}$. For k large enough, an autoreduction on 0^{n_k} will never query a string of size $\geq n_{k+1}$.

To compress $\chi_A[0..i]$, substitute the bit for 0^{n_k} by a sequence of bits corresponding to the answers to all queries of strings y for $y > 0^{n_k}$ in the lexicographic ordering. This results in a local expansion of χ_A. Then, remove from χ_A all the bits corresponding to those large queries. Overall, we removed the bit for 0^{n_k} and moved some other bits around in χ_A. This results in one bit of compression for each section of χ_A corresponding to strings between 0^{n_k} and $0^{n_{k+1}}$. Since this is done for each k, the number of bits of compression is unbounded.

To uncompress $\kappa[0..j]$, simulate the autoreduction machine on strings of the form 0^{n_k}, for successive k, using the appropriate bits of $\kappa[0..j]$ to answer the queries. This generates the missing bits and allows reordering of the other bits into $\chi_A[0..i]$. The time for uncompression includes the time for the autoreduction on strings of the form 0^{n_k}, which is polynomial in $\lg i$, and the time for reordering the bits, which is in $O(i)$.

To compress $\chi_A[0..i]$, if i does not represent a string of size n_k for some k, the compression is straightforward. Otherwise, the string $\chi_A[0..i]$ may not contain all the bits that are needed for the substitution in the position corresponding to string 0^{n_k}. For this case, substitute the bit by all possible sequences of cn_k bits, giving 2^{cn_k} candidate compressions. For each candidate compression, generate the rest of the bits according to the autoreduction using the sequence included. The time for the autoreduction is polynomial in n_k, and since $n_k \leq \lg(i+2)$, each candidate compression can be generated in linear time, and the total time is $O(i2^{c \lg i}) = O(i^{c+1})$. □

The next theorem deals with p-selective sets, introduced by Selman [23]. A set A is p-selective if there is a polynomial-time selector function f such that (1) $f(x,y) \in \{x,y\}$ and (2) if $x \in A$ or $y \in A$, then $f(x,y) \in A$. Intuitively, $f(x,y)$ hands back the most likely of x or y to be in A. We will use the following lemma.

LEMMA 14 (see [7]). *Let A be a p-selective set. Any finite set X can be ordered in polynomial time, using the p-selector, as follows: $\{x_1, \ldots, x_k\} = X$ such that $x_i \in A$ implies $x_j \in A$ for all $j \geq i$.*

THEOREM 15. *The class of sets that are polynomial-time truth-table reducible to a p-selective set has p-measure 0.*

Proof. Let A be polynomial-time truth-table reducible to a p-selective set S via reduction r. Assume that the reduction r runs in time n^k and that the p-selector function f also runs in time n^k. Let $n_j = 2^j$ for some integer j. We compress $\chi_A[2^{n_j} - 1..2^{n_j+1} - 2]$ (corresponding to strings of length n_j) into a string s_j as follows. Let S_j be the set of all queries to S generated by applying reduction r on all strings of length n_j. This set has at most $2^{n_j} n_j^k \leq 2^{2n_j}$ strings. Let s_j be the binary string of length $2n_j$ encoding $\|S_j \cap S\|$. The compressed string κ is χ_A, where for each n_j, $\chi_A[2^{n_j} - 1..2^{n_j+1} - 2]$ has been replaced by s_j. This will result

in an unbounded compression.

To recover a piece of χ_A that has been replaced by s_j, simulate the reduction r on every string of length n_j and collect all queries to form S_j. Then sort S_j by using the p-selector as a string comparator. (This sorting is possible, using the lemma above.) By the property of the p-selector, if s_j encodes m, then the m largest strings of S_j according to our sort are in S. Use this information about S and the reduction to S to compute the membership of all strings of length n_j in A. This requires 2^{n_j} applications of the reduction r, each using n_j^k time, and then $O(n_j 2^{n_j})$ comparisons for sorting, each comparison being an application of f using n_j^k time. If this process produces $\chi_A[0..i]$, the largest j for which this is done is $j = \lfloor \lg\lg((i+2)/2) \rfloor$. This process can be done in time 2^{2n_j} for each j, and the total process will recover $\chi_A[0..i]$ in time $\Sigma_{j \le \lfloor \log\log((i+2)/2) \rfloor} 2^{2n_j} \in O(i^2)$.

To compute κ, we only need to compute candidates for κ, one of which is the correct one. Generate all possible candidates. Starting from $\chi_A[0..i]$, the largest j for which s_j needs to be computed is $j = \lfloor \lg\lg(i+2) \rfloor$. The number of undetermined bits is at most $\Sigma_{j \le \lfloor \lg\lg(i+2) \rfloor} 2n_j \le 4\log(i+2)$, so the number of possible candidates is in $O(i^4)$. (Although not necessary here, it is possible to reduce the time to $O(i^2)$ by putting the indices n_j at 2^{2^j} instead of 2^j.) □

COROLLARY 16. *There is no p-selective set that is hard for E under polynomial-time truth-table reductions.*

Proof. E does not have p-measure 0 [16]. □

The next result shows that the class of sets that reduce by Turing reductions to a set that has subpolynomial density has p-measure 0. A function is subpolynomial if $\forall \epsilon \; \exists n_0 \; \forall n > n_0 : f(n) < n^\epsilon$.

THEOREM 17. *Let f be a subpolynomial function. The class of sets that Turing reduce to a set S with density f (i.e., $\|S^{\le n}\| \le f(n)$) has p-measure 0.*

Proof. Let $A \le_T^p S$, where S has density $f(n)$, via machine M_T in time $p(n)$. Again we have to show that the characteristic sequence of A can be compressed and uncompressed. We consider $w = \chi_A[2^n - 1..2^{n+1} - 2]$, corresponding to strings of size n. We will compress the first n bits of w. Each string x of length n is mapped by M_T to at most $p(n)$ many different queries, provided that the answers to these queries are known. We replace the first n bits of w by pairs of the form $\langle i, j \rangle$, meaning that the jth query asked by M_T on the ith string of length n is in S. The number of bits needed to write these pairs is $O(f(p(n)) \log n)$, which is less than n because f is subpolynomial. To uncompress, simulate M_T on all those n strings of length n, using the compressed string to answer queries to S. To compress, generate all possible sequences of pairs. This results in at most $2^{O(f(p(n))) \log n}$ possible extensions, with the hidden constant in the exponent not depending on A. Only output the ones that are consistent with the known part of χ_A. One of the possibilities has to be a prefix of κ. □

COROLLARY 18 (see [6]). *There is no Turing hard set for E with subpolynomial density.*

Acknowledgments. We would like to thank Paul Vitányi for suggesting that we look at a Kolmogorov type of characterization of resource bounded measure. We would like to thank Martin Strauss for useful discussions that led us to improved results, and David Martin for his useful comments. We also would like to thank A. M. Bobu for hosting the meeting that enabled us to initiate this research.

REFERENCES

[1] E. Allender and M. Strauss, *Measure on small complexity classes, with applications for BPP*, in Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 807–818.

[2] K. Ambos-Spies, *p-mitotic sets*, in Logic and Machines, Lecture Notes in Comput. Sci. 177, Springer-Verlag, New York, 1994, pp. 1–23.

[3] K. Ambos-Spies, C. Neis, and S. A. Terwijn, *Genericity and measure for exponential time*, Theoret. Comput. Sci., 168 (1996), pp. 3–19.

[4] K. Ambos-Spies, S. Terwijn, and X. Zheng, *Resource bounded randomness and weakly complete problems*, Theoret. Comput. Sci., 172 (1997), pp. 195–207.

[5] C. Bennett and J. Gill, *Relative to a random oracle A, $P^A \neq NP^A \neq Co\text{-}NP^A$ with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[6] H. Buhrman and M. Hermo, *On the sparse set conjecture for sets with low density*, in STACS 95, Lecture Notes in Comput. Sci. 900, E. W. Mayr and C. Puech, eds., Springer-Verlag, New York, 1995, pp. 609–618.

[7] H. Buhrman, P. van Helden, and L. Torenvliet, *P-selective self-reducible sets: A new characterization of P*, in Proceedings of the Eighth Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 44–51.

[8] J.-Y. Cai, D. Sivakumar, and M. Strauss, *Constant-depth circuits and the Lutz hypothesis*, in Proceedings of the 38th Foundations of Computer Science Conference, Miami Beach, FL, 1997, pp. 595–604.

[9] D. W. Juedes and J. H. Lutz, *The complexity and distribution of hard problems*, SIAM J. Comput., 24 (1995), pp. 279–295.

[10] D. W. Juedes and J. H. Lutz, *Weak completeness in E and E_2*, Theoret. Comput. Sci., 143 (1995), pp. 149–158.

[11] V. Kreinovich and L. Longpré, *Randomness as Incompressibility: A Non-Algorithmic Analogue*, manuscript.

[12] L. Levin, *On the notion of a random sequence*, Soviet Math. Dokl., 14 (1973), pp. 1413–1416.

[13] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993.

[14] J. Lutz, *Category and measure in complexity classes*, SIAM J. Comput., 19 (1990), pp. 1100–1131.

[15] J. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[16] J. Lutz, *Weakly hard problems*, SIAM J. Comput., 24 (1995), pp. 1170–1189.

[17] J. Lutz and E. Mayordomo, *Cook versus Karp-Levin: Separating completeness notions if NP is not small*, Theoret. Comput. Sci., 164 (1996), pp. 141–163.

[18] J. Lutz and E. Mayordomo, *Measure, stochasticity, and the density of hard languages*, SIAM J. Comput., 23 (1994), pp. 762–779.

[19] P. Martin-Löf, *The definition of random sequences*, Information and Control, 9 (1966), pp. 602–619.

[20] E. Mayordomo, *Almost every set in exponential time is p-bi-immune*, Theoret. Comput. Sci., 136 (1994), pp. 487–506.

[21] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Ph.D. thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.

[22] C. Schnorr, *Process complexity and effective random tests*, J. Comput. System Sci., 7 (1973), pp. 376–388.

[23] A. Selman, *P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP*, Math. Systems Theory, 13 (1979), pp. 55–65.

[24] M. Strauss, *Measure on P: Strength of the notion*, Inform. and Comput., 136 (1997), pp. 1–23.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 887–905

RESOURCE-BOUNDED KOLMOGOROV COMPLEXITY REVISITED*

HARRY BUHRMAN†, LANCE FORTNOW‡, AND SOPHIE LAPLANTE§

Abstract. We take a fresh look at **CD** complexity, where $\mathbf{CD}^t(x)$ is the size of the smallest program that distinguishes x from all other strings in time $t(|x|)$. We also look at **CND** complexity, a new nondeterministic variant of **CD** complexity, and time-bounded Kolmogorov complexity, denoted by **C** complexity.

We show several results relating time-bounded **C**, **CD**, and **CND** complexity and their applications to a variety of questions in computational complexity theory, including the following:

- Showing how to approximate the size of a set using **CD** complexity without using the random string as needed in Sipser's earlier proof of a similar result. Also, we give a new simpler proof of this result of Sipser's.
- Improving these bounds for almost all strings, using extractors.
- A proof of the Valiant–Vazirani lemma directly from Sipser's earlier **CD** lemma.
- A relativized lower bound for **CND** complexity.
- Exact characterizations of equivalences between **C**, **CD**, and **CND** complexity.
- Showing that satisfying assignments of a satisfiable Boolean formula can be enumerated in time polynomial in the size of the output if and only if a unique assignment can be found quickly. This answers an open question of Papadimitriou.
- A new Kolmogorov complexity-based proof that $\mathbf{BPP} \subseteq \Sigma_2^p$.
- New Kolmogorov complexity based constructions of the following relativized worlds:
  - There exists an infinite set in **P** with no sparse infinite **NP** subsets.
  - **EXP** = **NEXP** but there exists a **NEXP** machine whose accepting paths cannot be found in exponential time.
  - Satisfying assignments cannot be found with nonadaptive queries to **SAT**.

Key words. computational complexity, Kolmogorov complexity, CD complexity

AMS subject classifications. 68Q15, 68Q30

PII. S009753979834388X

1. Introduction.

Originally designed to measure the randomness of strings, Kolmogorov complexity has become an important tool in computability and complexity theory. A simple lower bound showing that there exist random strings of every length has had several important applications (see [19, Chapter 6]).

Early in the history of computational complexity theory, many people naturally looked at resource-bounded versions of Kolmogorov complexity. This line of research was initially fruitful and led to some interesting results. In particular, Sipser [24] invented a new variation of resource-bounded complexity, called **CD** complexity, where one considers the size of the smallest program that accepts one specific string and no others. Sipser used **CD** complexity for the first proof that **BPP** is contained in the polynomial-time hierarchy.

*Received by the editors August 27, 1998; accepted for publication (in revised form) May 30, 2001; published electronically January 11, 2002.

http://www.siam.org/journals/sicomp/31-3/34388.html

†CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands (buhrman@cwi.nl). Part of this research was done while the author was visiting the University of Chicago. This author was partially supported by the Dutch foundation for scientific research (NWO) through NFI Project ALADDIN, under contract NF 62-376.

‡NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 (fortnow@research.nj.nec.com). Part of this work was conducted at the University of Chicago. This author was supported in part by NSF grant CCR 92-53582.

§LRI, Université Paris-Sud, Bâtiment 490, 91405 Orsay, France (laplante@lri.fr). Part of this work was done while the author was at the University of Chicago.

Complexity theory has marched on for the past two decades, but resource-bounded Kolmogorov complexity has seen little interest. Now that computational complexity theory has matured a bit, we ought to look back at resource-bounded Kolmogorov complexity and see what new results and applications we can draw from it.

First, we use algebraic techniques to give a new upper bound lemma for **CD** complexity, without the additional advice required of Sipser's lemma [24]. With this lemma, we can measure the approximate size of a set using **CD** complexity.

We obtain better bounds on **CD** complexity using extractor graphs. These graphs are usually used for derandomization. However, these improved bounds apply only to most of the strings.

We also give a new, simpler proof of Sipser's lemma and show how it implies the important Valiant–Vazirani lemma [27] that randomly isolates satisfying assignments. Surprisingly, Sipser's paper predates the result of Valiant and Vazirani.

We define **CND** complexity as a variation of **CD** complexity where we allow nondeterministic computation. We prove a lower bound for **CND** complexity where we show that there exists an infinite set A such that every string in A has high **CND** complexity even if we allow access to A as an oracle. We use this lemma to prove some negative results on nondeterministic search vs. deterministic decision.

Once we have these tools in place, we use them to unify several important theorems in complexity theory. We answer an open question of Papadimitriou [22] by characterizing exactly when the set of satisfying assignments of a formula can be enumerated in output polynomial time. We also give straightforward proofs that **BPP** is in Σ_2^p (first proven by Gács (see [24])) and create relativized worlds where assignments to **SAT** cannot be found with nonadaptive queries to **SAT** (first proven by Buhrman and Thierauf [4]), and where **EXP** = **NEXP** but there exists a **NEXP** machine whose accepting paths cannot be found in exponential time (first proven by Impagliazzo and Tardos [15]).

These results in their original form require a great deal of time to fully understand the proofs because either the ideas and/or technical details are quite complex. We show that by understanding resource-bounded Kolmogorov complexity, one can see full and complete proofs of these results without much additional effort. We also look at when polynomial-time **C**, **CD**, and **CND** complexity coincide. We give a precise characterization of when we have equality of these measures, and some interesting consequences thereof.

2. Preliminaries. We use basic concepts and notation from computational complexity theory texts like Balcázar, Díaz, and Gabarró [1] and Kolmogorov complexity from the excellent book by Li and Vitányi [19]. We use $|x|$ to represent the length of a string x and $\|A\|$ to represent the number of elements in the set A. $A^{=n}$ is the set of strings in A of length n. $[N]$ denotes the set of integers between 1 and N. All of the logarithms are base 2.

Formally, we define the Kolmogorov complexity function $\mathbf{C}_\phi(x|y)$ by

$$\mathbf{C}_\phi(x|y) = \min_p\{|p| : \phi(p, y) = x\}.$$

There exists a universal machine U such that for all ϕ there is a constant c such that, for all x and y, $\mathbf{C}_U(x|y) \leq \mathbf{C}_\phi(x|y) + c$. We fix such a U and let $\mathbf{C}(x|y) = \mathbf{C}_U(x|y)$. We define unconditional Kolmogorov complexity by $\mathbf{C}(x) = \mathbf{C}(x|\epsilon)$.

A few basic facts about Kolmogorov complexity follow:

- The choice of U affects the Kolmogorov complexity by at most an additive constant.

- For some constant c, $\mathbf{C}(x) \leq |x| + c$ for every x.
- For every n and every y, there is an x such that $|x| = n$ and $\mathbf{C}(x|y) \geq n$.

We will also use time-bounded Kolmogorov complexity. Fix a fully time-constructible function $t(n) \geq n$. We define the $\mathbf{C}^t(x|y)$ complexity function as

$$\mathbf{C}^t(x|y) = \min_p\{|p| : U(p, y) = x \text{ and } U(p) \text{ runs in at most } t(|x| + |y|) \text{ steps}\}.$$

As before, we let $\mathbf{C}^t(x) = \mathbf{C}^t(x|\epsilon)$. A different universal U may affect the complexity by at most a constant additive term and the time by a $\log(t)$ factor.

While the usual Kolmogorov complexity asks about the smallest program to *produce* a given string, we may also want the smallest program to *distinguish* a string. While this difference affects the unbounded Kolmogorov complexity by only a constant, it can make a difference for the time-bounded case. Sipser [24] defined the distinguishing complexity $\mathbf{CD}^t$ by

$$\mathbf{CD}^t(x|y) = \min_p \left\{ |p| : \begin{array}{l} (1)\ U(p, x, y) \text{ accepts} \\ (2)\ U(p, z, y) \text{ rejects for all } z \neq x \\ (3)\ U(p, z, y) \text{ runs in } \leq t(|z|+|y|) \text{ steps for all } z \in \Sigma^* \end{array} \right\}.$$

When the auxiliary input string y is the empty string, we write $\mathbf{CD}^t(x)$.

We fix a universal nondeterministic Turing machine, U_n. We define the nondeterministic distinguishing complexity $\mathbf{CND}^t$ by

$$\mathbf{CND}^t(x|y) = \min_p \left\{ |p| : \begin{array}{l} (1)\ U_n(p, x, y) \text{ accepts} \\ (2)\ U_n(p, z, y) \text{ rejects for all } z \neq x \\ (3)\ U_n(p, z, y) \text{ runs in } \leq t(|z|+|y|) \text{ steps for all } z \in \Sigma^* \end{array} \right\}.$$

In this definition, we mean that the nondeterministic Turing machine accepts or rejects in the usual sense of nondeterministic computation. Once again we let $\mathbf{CND}^t(x) = \mathbf{CND}^t(x|\epsilon)$.

We can also allow for relativized Kolmogorov complexity. For example, $\mathbf{CD}^{t,A}(x|y)$ is defined as above except that the universal machine U has access to A as an oracle.

One can distinguish a string by generating it and then comparing it with the input, as stated in the following lemma.

LEMMA 2.1. *For all t $\exists c$ for all x, y : $\mathbf{CD}^{ct\log t}(x \mid y) \leq \mathbf{C}^t(x \mid y) + c$, where c is a constant.*

Likewise, every deterministic computation is also a nondeterministic computation, hence the lemma that follows.

LEMMA 2.2. *For all t $\exists c$ for all x, y : $\mathbf{CND}^{ct\log t}(x \mid y) \leq \mathbf{CD}^t(x \mid y) + c$.*

In section 7 we examine the consequences of the converses of these lemmas.

3. Approximating sets with distinguishing complexity. In this section we derive a lemma that enables one to approximate deterministically the density of a set, using polynomial-time distinguishing complexity. The technique we use of considering values modulo a prime is reminiscent of the hashing via the division method (see [17, p. 515]).

LEMMA 3.1. *Let $S = \{x_1, \ldots, x_d\} \subseteq \{0, \ldots, 2^n - 1\}$. For all $x_i \in S$ and at least half of the primes $p \leq 4dn^2$, $x_i \not\equiv x_j \bmod p$ for all $j \neq i$.*

Proof. For each $x_i, x_j \in S$, $i \neq j$, it holds that for at most n different prime numbers p, $x_i \equiv x_j \bmod p$ by the Chinese remainder theorem. (Alternatively, $|x_i - x_j| < 2^n$, so it can not have more than n prime factors.) For x_i there are at most dn

primes p such that $x_i \equiv x_j \bmod p$ for some $x_j \in S$. The prime number theorem [14] (see also [13]) states that for any m there are approximately $m/\ln(m) > m/\log(m)$ primes less than m. There are at least $4dn^2/\log(4dn^2) > 2dn$ primes less than $4dn^2$. So at least half of these primes p must have $x_i \not\equiv x_j \bmod p$ for all $j \neq i$. □

LEMMA 3.2. *Let A be any set. For all strings $x \in A^{=n}$ it holds that $\mathbf{CD}^{p,A^{=n}}(x) \leq 2\log(\|A^{=n}\|) + O(\log(n))$ for some polynomial p.*

Proof. Fix n and let $S = A^{=n}$. Fix $x \in S$ and a prime p_x fulfilling the conditions of Lemma 3.1 for x.

The $\mathbf{CD}^{poly,A}$ program for x works as follows:

input y
If $y \notin A^{=n}$ then **reject**
else if $y \bmod p_x = x \bmod p_x$ then **accept**
else **reject**

The size of the above program is $|p_x| + |x \bmod p_x| + O(1)$. This is $2\log(\|A\|) + O(\log(n))$. It is clear that the program runs in polynomial time and accepts only x. □

We note that Lemma 3.2 also works for $\mathbf{CND}^p$ complexity for some polynomial p.

Buhrman, Laplante, and Miltersen [3] show that Lemma 3.2 is tight.

THEOREM 3.3 (Buhrman–Laplante–Miltersen). *For every polynomial p and sufficiently large n there exists a set of strings $A \subseteq \{0,1\}^n$ containing more than $2^{n/50}$ strings such that there is an x in A with*

$$\mathbf{CD}^{p,A}(x) \geq 2\log(\|A^{=n}\|) - O(1).$$

COROLLARY 3.4. *Let A be a set in $\mathbf{P}$. For each string $x \in A$ it holds that $\mathbf{CD}^p(x) \leq 2\log(\|A^{=n}\|) + O(\log(n))$ for some polynomial p.*

Proof. We will use the same scheme as in Lemma 3.2, now using that $A \in \mathbf{P}$ and specifying the length of x, yielding an extra $\log(n)$ term for $|x|$ plus an additional $2\log\log(n)$ penalty for concatenating the strings. □

Theorem 3.3 also gives a relativized tightness result for Corollary 3.4.

COROLLARY 3.5.

1. *A set S is sparse if and only if, for all $x \in S$, $\mathbf{CD}^{p,S}(x) \leq O(\log(|x|))$ for some polynomial p.*
2. *A set $S \in \mathbf{P}$ is sparse if and only if, for all $x \in S$, $\mathbf{CD}^p(x) \leq O(\log(|x|))$ for some polynomial p.*
3. *A set $S \in \mathbf{NP}$ is sparse if and only if, for all $x \in S$, $\mathbf{CND}^p(x) \leq O(\log(|x|))$ for some polynomial p.*

Proof. Lemma 3.2 yields that all strings in a sparse set have $O(\log(n))$ $\mathbf{CD}^p$ complexity. On the other hand, simple counting shows that for any set A there must be a string $x \in A$ such that $\mathbf{CND}^A(x) \geq \log(\|A^{=|x|}\|)$. □

3.1. Sipser's lemma. We can also use Lemma 3.1 to give a simple proof of the following important result due to Sipser [24].

LEMMA 3.6 (Sipser). *For every polynomial-time computable set A there exists a polynomial p and constant c such that for every n, for most r in $\Sigma^{p(n)}$ and every $x \in A^{=n}$,*

$$\mathbf{CD}^{p,A^{=n}}(x|r) \leq \log(\|A^{=n}\|) + c\log(n).$$

Proof. For each k, $1 \leq k \leq n$, let r_k be a list of $4k(n+1)$ randomly chosen numbers less than 2^k. Let r be the concatenation of all of the r_k.

Fix $x \in A^{=n}$. Let $d = \|A^{=n}\|$. Fix k such that $2^{k-1} < 4dn^2 \leq 2^k$. Consider one of the numbers y listed in r_k. By the prime number theorem [13], the probability that y is prime and less than $4dn^2$ is at least $1/2\log(4dn^2)$. The probability that y fulfills the conditions of Lemma 3.1 for x is at least $1/4\log(4dn^2) > 1/4k$. With probability about $1 - 1/e^{n+1} > 1 - 1/2^{n+1}$, we have that some y in r_k fulfills the conditions of Lemma 3.1.

With probability at least $1/2$, for every $x \in A$ there is some y listed in r_k fulfilling the conditions of Lemma 3.1 for x.

We can now describe x by x mod y and the pointer to y in r. □

Note. Sipser's original proof gives a tighter bound than $c\log(n)$, but for most applications the additional $O(\log(n))$ additive factor makes no substantial difference.

Comparing our Lemma 3.2 with Sipser's lemma (Lemma 3.6), we are able to eliminate the random string required by Sipser at the cost of an additional $\log(\|A^{=n}\|)$ bits.

4. Approximating sets with extractors. By using extractors, we can obtain nearly the bound of Sipser's lemma 3.6 without the random string it requires. However, our result only works for most strings in A.

THEOREM 4.1. *For any set A and any function $\varepsilon(n)$ there is a polynomial p such that for all n and for all but a $2\varepsilon(n)$ fraction of the $x \in A^{=n}$,* $\mathbf{CD}^{p,A^{=n}}(x) \leq \log\|A^{=n}\| + \log^{O(1)}(n/\varepsilon(n))$.

We give a nondeterministic version of this result and give a bound on **CND** complexity. We also give a randomized version of these theorems, stating that the shorter string can be chosen at random and the probability of getting a short string which encodes as much information as the original string is bounded away from $1/2$.

4.1. Extractors. An extractor can be thought of as a bipartite graph, whose first color class is larger than the second color class. By convention, we think of the first color class as being on the left, and the second on the right. The vertices on the left side are all the strings of length n, so the first color class can be equated with the set $[N]$, where $N = 2^n$. Likewise, the vertices on the right side of the graph are labeled by strings of length $m \leq n$, so we let $M = 2^m$, and $[M]$ is equated with the vertices in the second color class.

4.1.1. Distributions. We will be choosing a node on the left side of the graph at random according to a distribution X. The result of choosing a neighbor uniformly at random in the graph will produce a distribution Y on vertices on the right.

The *min-entropy* of a distribution X over $[N]$ is defined as $\min\{-\log_2(X(x))|x \in [N]\}$. The min-entropy of X can be thought of as a measure of the randomness present in a string x chosen according to X.

A distribution Y is said to be ε-close to Z if both distributions are over the same space $[M]$, and such that for any $S \subseteq [M]$, $|Y(S) - Z(S)| \leq \varepsilon$.

4.1.2. Definition of extractors. A bipartite graph G with (independent) vertex sets $[N]$ and $[M]$, $N = 2^n, M = 2^m$, and for which the degree of all the vertices in the first color class is bounded by $D = 2^d$, is an $(n, k, d, m, \varepsilon)$ *extractor* if, given any distribution X on the N vertices whose min-entropy is at least k, the result of choosing an x according to this distribution and a random neighbor y of x in the graph is ε-close to the uniform distribution over $[M]$. In our setting, the distribution X will be the uniform distribution over a subset $A \subseteq [N]$, so k will be $\log(\|A\|)$.

$\Gamma(x)$ denotes the set of neighbors of x in G when x is a vertex on the left side of the graph. The number of edges originating at some vertex x on the left side of

the graph is called the *outdegree* of x, whereas the number $w(y)$ of edges adjacent with a vertex y on the right side of the graph is called the *indegree* (or weight) of y. $G(x,r)$ represents the rth neighbor of x in the graph, where multiple edges are allowed. When y is a vertex on the right side of the graph, $\Gamma_A^{-1}(y)$ is the subset of preimages of y which lie in A. The notation extends to sets in the natural way.

4.1.3. Best-known explicit constructions. The results we state are subject to improvement if better explicit extractor constructions are found. We have stated our results in general terms so that new results on extractors will be immediately applicable.

The current best-known explicit constructions for extractors are due to Ta-Shma [25], Zuckerman [29], Trevisan [26], and Raz, Reingold, and Vadhan [23]. The extractors best suited for our purposes are the ones which can be constructed for any k, and with $m = k$, with the smallest amount of additional randomness. We illustrate our results with the parameters obtained from Ta-Shma's construction.

THEOREM 4.2 (Ta-Shma). *There is an explicit construction that, for every n and for any function $\varepsilon(n)$ and every $m = m(n) \le n$, yields an extractor with parameters $(n, m, \log^{O(1)}(n/\varepsilon(n)), m, \varepsilon(n))$.*

It is useful to compare this construction to the current lower bound on extractors, due to Nisan and Zuckerman [20].

THEOREM 4.3 (Nisan–Zuckerman). *There is a constant c such that for all $n, m, k \le n-1$, $\varepsilon < 1/2$, if there is an extractor whose parameters are $(n, k, d, m, \varepsilon)$, then it must be the case that $d \ge \min\{m, c\log(n/\varepsilon)\}$.*

This lower bound also gives a good indication as to the limits of the techniques described in this paper.

4.2. Extracting CD complexity. Theorem 4.1 follows immediately from the following result, using the explicit extractor construction of Ta-Shma. For this theorem we assume that there is an explicit extractor construction whose parameters are $(n, k, d, m, \varepsilon)$, and we write $M = 2^m$.

THEOREM 4.4. *Fix a set A, a polynomial $q(n)$, and $\varepsilon = \varepsilon(n)$. Then there is a polynomial $p(n)$ such that for all n and for all but a 2ε fraction of the $x \in A^{=n}$, there is a y such that*

1. $|y| = m$,
2. $\mathbf{C}^p(y|x) \le d + O(1)$,
3. $\mathbf{CD}^{p,A^{=n}}(x) \le \mathbf{C}^q(y) + 3d + 2\log(\|A^{=n}\|/M) + c\log(n + d + \log(\|A^{=n}\|/M)) + O(1)$,

where the underlying extractor's parameters are determined by n and $k = \log(\|A^{=n}\|)$, and c is a small absolute constant.

For the remainder of this section, we fix n and we let $S = \|A^{=n}\|$. In our setting, we will think of the set $A^{=n}$ as defining a distribution of min-entropy $k = \log(S)$. The string x represents an element of $A^{=n}$ and y is one of its neighbors in the graph G. Hence y has length m; computing y from x requires knowing only a short ("random") string of length d. As we will see, y together with some short additional distinguishing information will suffice to distinguish the string x (in the sense of **CD** complexity).

The following lemmas are at the heart of the argument. They allow us to upper-bound the number of "bad" elements in $A^{=n}$, where "bad" means the strings x for which Theorem 4.4 will not apply.

In order to get a short description for x, we need to find a string y in its range which has small indegree (counting only those edges originating in $A^{=n}$).

In Lemma 4.5, we use the properties of the extractor to obtain an upper bound on the number of y which have large indegree. In the statement of the lemma, we use the variable w_0 to represent the threshold on degree: any vertex with degree larger than w_0 has large degree. A typical value for w_0 is twice the average degree of the graph.

Lemma 4.6 gives an upper bound on the number of x on the left side of the extractor whose neighbors all lie within a small subset of the right side of the graph. When the small subset is the set of vertices with large indegree, these x are the "bad" x to which the theorem will not apply.

LEMMA 4.5. *Consider the restriction of the extractor to the set of edges originating in $A^{=n}$. Recall that the degree of the graph is bounded by $D = 2^d$. In this restricted graph, let w_0 be an indegree threshold, $\frac{DS}{M} < w_0 \leq DS$, and Y be a subset of vertices on the right-hand side of the extractor graph. If for all $y \in Y$, $w(y) > w_0$, then $\|Y\| \leq \varepsilon \left(\frac{w_0}{DS} - \frac{1}{M}\right)^{-1}$.*

Proof. Let Y be the set of vertices whose indegree (in the restricted graph) exceeds w_0. Because the graph is an extractor, it must be the case that $\varepsilon \geq \frac{w(Y)}{\Gamma(A^{=n})} - \frac{\|Y\|}{M} \geq \frac{w(Y)}{DS} - \frac{\|Y\|}{M}$. Since $w(Y) \geq w_0\|Y\|$, we get $\|Y\| \leq \varepsilon \left(\frac{w_0}{DS} - \frac{1}{M}\right)^{-1}$ as claimed. □

LEMMA 4.6. *In the restricted graph, if Y is a set on the right side of the graph, then*

$$\|\{x \in A^{=n} : \Gamma(x) \subseteq Y\}\| \leq \left(\varepsilon + \frac{\|Y\|}{M}\right) S.$$

Proof. Let $X = \{x \in A^{=n} : \Gamma(x) \subseteq Y\}$. The distribution which consists of picking a random element of $A^{=n}$ and then choosing a random neighbor gives measure at least $\|X\|/S$ to the set Y. Because of the extractor property, $\frac{\|X\|}{S} - \frac{\|Y\|}{M} \leq \varepsilon$. □

To conclude, we give the proof of Theorem 4.4.

Proof. Let A be a set and ε, n be given as in the statement of the theorem. By Lemma 4.5, applied with $w_0 = 2DS/M$ ($D = 2^d$), and Lemma 4.6 with Y as in the hypothesis of Lemma 4.5, the size of the subset $B \subseteq A^{=n}$ such that for all $x \in B$, for all $y \in \Gamma(x)$, y has indegree at least w_0 can have size at most $2\varepsilon S$. Therefore for all but $2\varepsilon S$ of the x in $A^{=n}$, there is a y in its range whose indegree is at most $2DS/M$. For each such x, let r_x be the label of one of the edges in G which connects x to such a y. We need to verify 3 properties for each of these pairs x, y.

1. $|y| = m$: This is by choice of the extractor G.
2. $\mathbf{C}^p(y|x) \leq d + O(1)$: $y = G(x, r_x)$ for some $r_x \in \Sigma^d$, so the algorithm to print y will contain an encoding of r_x and on input x computes $G(x, r_x)$ and output the result.
3. $\mathbf{CD}^{p,A^{=n}}(x) \leq \mathbf{C}^q(y) + 3d + 2\log(S/M) + c\log(n) + O(1)$: The program to recognize x will contain an encoding for an r_x and y for which $G(x, r_x) = y$ and the indegree of y is at most $2DS/M$. It must also contain a distinguishing program p_x, which recognizes x among the $2DS/M$ vertices on the left originating in A that are adjacent to y. (The encoding of r_x is required to test that x is adjacent to y, but may be omitted if the degree of the graph is polynomial. This is not the case in the current explicit, efficient extractors, whose degree is on the order of $2^{\log^{O(1)}(n)}$.) The length of p_x is bounded by $2\log(2DS/M) + O(\log(n + \log(2DS/M)))$, by Lemma 3.2. (An additional logarithmic term is needed to encode the lengths of the various components of the encoding, but this is bundled in the O notation.)

The algorithm follows:

input z
If $z \notin A^{=n}$ then **reject**
else if $G(z, r_x) \neq y$ then **reject**
else if $p_x(z) = 1$ then **accept**
else **reject**

So the program requires an encoding of y, r, and the distinguishing program p_x, for a total length of $\mathbf{C}^q(y) + d + 2\log(2DS/M) + c\log(n + \log(2DS/M)) + O(1)$. □

4.3. Extracting CND complexity. A statement analogous to Theorem 4.4 can be made for **CND** complexity. Using a slight variant of the proof of Theorem 4.4, we can get a bound which is smaller by a term of d. Also, in the upper bound, $\mathbf{CD}^q(y)$ is used instead of the possibly larger term $\mathbf{C}^q(y)$.

THEOREM 4.7. *Fix a set A in* **NP**, *a polynomial $q(n)$, and $\varepsilon = \varepsilon(n)$. Then there is a polynomial $p(n)$ such that for every n and for all but a 2ε fraction of the $x \in A^{=n}$, there is a y such that*

1. $|y| = \log(\|A^{=n}\|)$,
2. $\mathbf{C}^p(y|x) \leq d + O(1)$,
3. $\mathbf{CND}^p(x) \leq \mathbf{CD}^q(y) + 2d + c\log(n + d) + O(1)$.

The proof is essentially the same as that of Theorem 4.4. To simplify the notation, we make the assumption that the extractor used achieves $k = m$, as does Ta-Shma's construction. To obtain property 3, we need only guess y and verify our guess using a distinguishing program for y whose length is bounded by $\mathbf{CD}^q(y)$. Likewise, we can simply guess r, omit its encoding, and use the distinguishing program p to verify our guess for r.

4.4. Randomly extracting CD complexity. Another variant that saves a $d = \log(D)$ term is to choose a counterpart y to a string x in a set in **P** at random. We will require only that, for most x, at least half of the edges from x map to a "good" y. Although this comes at the cost of applying only to "most" strings x, this improves upon the result of Sipser [24] by reducing the length of the random string from $n^{O(1)}$ to $\log^{O(1)}(n/\varepsilon)$. The proof is similar to that of Theorem 4.4; it requires only a slight modification to the counting argument.

THEOREM 4.8. *Fix a set A in* **P**, *a polynomial $q(n)$, and a function $\varepsilon(n)$. Then there is a polynomial $p(n)$ such that for every n and for all but a $4\varepsilon(n)$ fraction of the $x \in A^{=n}$, and at least half of the strings r of length d, there is a y such that*

1. $|y| = \log(\|A^{=n}\|)$,
2. $\mathbf{C}^p(y|x, r) \leq O(1)$,
3. $\mathbf{CD}^p(x|r) \leq \mathbf{C}^q(y) + 2d + 2\log(n + d) + O(1)$.

5. Extracting random strings. In the previous section, we used the fact that the strings examined were in a small set of bounded complexity, and we showed the existence of strings for which the mutual information was roughly the **CND** complexity of the original string. Here we use extractor techniques to a achieve a slightly different goal. We obtain an *incompressible* string whose length is close to the **CD** complexity of x and which can be computed from x using only $\log(n/\varepsilon)$ bits.

In the case of unbounded Kolmogorov complexity, it is easy to see that the following proposition is true.

PROPOSITION 5.1 (see [19, Example 2.1.5, p. 102]). *For any string x of length n, there is a y such that*

1. $|y| = \mathbf{C}(x)$,
2. $\mathbf{C}(y|x) \leq \log(n)$,
3. $\mathbf{C}(y) > |y| - O(1)$.

Namely, y is a minimal-length program for x and can be obtained from x by dovetailing, given the value of $\mathbf{C}(x)$. In the time-bounded setting, however, this argument fails, since dovetailing would take too much time. Our use of extractors is far afield from the above approach, yet it yields results surprisingly close to Proposition 5.1. (Nonexplicit extractors actually allow us to give an alternate proof of Proposition 5.1, although this is more an artifact than a useful new proof.)

THEOREM 5.2. *For any polynomial $q(n)$ and function $\varepsilon(n)$, there exists a polynomial $p(n)$ such that for any string x of length n there is a string y such that*

1. $|y| = \mathbf{CND}^p(x)/2 - c_1 \log(n)$,
2. $\mathbf{C}^p(y|x) \leq \log^{c_2}(n/\varepsilon(n))$,
3. $\mathbf{C}^q(y) > |y| - c_\varepsilon$,

where c_1, c_2 are absolute constants and c_ε depends only on ε.

Instead of giving the proof of Theorem 5.2, we prove the result in the following more general form, which may be improved as explicit extractor constructions are improved.

THEOREM 5.3. *For any polynomial $q(n)$ and $\varepsilon = \varepsilon(n)$, there exists a polynomial $p(n)$ such that for any string x there is a string y such that*

1. $|y| = m$,
2. $\mathbf{C}^p(y|x) \leq d + c_1$,
3. $\mathbf{C}^q(y) > |y| - c_\varepsilon$.

In the statement above, c_1 is an absolute constant, c_ε is a constant depending only on ε, $k = \frac{1}{2}(\mathbf{CND}^{2^d \cdot p}(x) - 2\log(n) - c_1 - 1)$ and $(n, k, d, m, \varepsilon)$ are the parameters of an explicit extractor.

Theorem 5.2 follows by applying Theorem 5.3 with parameters obtained from Ta-Shma's extractor [25].

Proof (sketch). Consider a family of extractors with parameters $n, k, m(k)$. Fix any n, k and let $G = G_{n,k,m}$, $m = m(k)$, be the extractor with parameters n, m, k. (Later we will fix k to be a specific value.) Let $A_{n,m} = \{x | \Gamma(x) \subseteq \mathbf{C}[q(n), m - c_\varepsilon]\}$, where $\mathbf{C}[t, l] = \{z | \mathbf{C}^t(z) \leq l\}$, and let c_ε be chosen so that $c_\varepsilon > \log(\frac{1}{1-\varepsilon})$ for large enough n.

The fact that G is an extractor prohibits the set $A_{n,m}$ from being large, as we see now. If $\|A_{n,m}\| > 2^k$, then by the properties of the extractor,

$$1 - \varepsilon \leq \frac{\|\mathbf{C}[q(n), m - c_\varepsilon]\|}{2^m}.$$

But $\|\mathbf{C}[q(n), m - c_\varepsilon]\| \leq 2^{m - c_\varepsilon}$, and we have chosen $c_\varepsilon > \log(\frac{1}{1-\varepsilon})$ in order to get a contradiction. Hence we must conclude that $\|A_{n,m}\| \leq 2^k$.

Now we may apply Lemma 3.2 for **CND** to conclude that all $x \in A_{n,m}$ must have small **CND** complexity. First notice that verifying membership in $A_{n,m}$ is in NTIME$[2^d \cdot p]$ for some polynomial p, since it suffices to guess, for each neighbor y of x in $G_{n,m}$, a program of length $m - c_\varepsilon$ which prints out y. Hence, there exists a constant c_1 such that for every $x \in A_{n,m}$, $\mathbf{CND}^{2^d \cdot p}(x) \leq 2\log(\|A_{n,m}\|) + 2\log(n) + c_1$.

Now consider x with respect to the extractor $G_{n,\hat{k},m(\hat{k})}$, where

$$\hat{k} = \frac{1}{2}\left(\mathbf{CND}^{2^d \cdot p}(x) - 2\log(n) - c_1 - 1\right)$$

and m is maximal for this k. By the observation above, it must be the case that $x \notin A_{n,m}$. Therefore there must be a y not in $\mathbf{C}[q(n), m - c_\varepsilon]$ to which x is mapped under $G_{n,k,m}$. It is easy to verify that y satisfies the properties claimed in the statement of the theorem. □

6. Lower bounds. In this section we show that there exists an infinite set A such that every string in A has high **CND** complexity, even relative to A.

Fortnow and Kummer [8] prove the following result about relativized **CD** complexity.

THEOREM 6.1. *There is an infinite set A such that for every polynomial p, $\mathbf{CD}^{p,A}(x) \geq |x|/5$ for almost all $x \in A$.*

We extend and strengthen their result for **CND** complexity.

THEOREM 6.2. *There is an infinite set A such that $\mathbf{CND}^{2^{\sqrt{|x|}},A}(x) \geq |x|/4$ for all $x \in A$.*

The proof of Fortnow and Kummer of Theorem 6.1 uses the fact that one can start with a large set A of strings of the same length such that any polynomial-time algorithm on an input x in A cannot query any other y in A. However, a nondeterministic machine may query every string of a given length. Thus we need a more careful proof.

This proof is based on the proof of a result due to Goldsmith, Hemachandra, and Kunen [9], which we obtain as Corollary 6.3 below. In section 9, we will also describe a rough equivalence between this result and an "X-search" theorem of Impagliazzo and Tardos [15].

Proof of Theorem 6.2. We create our set A in stages. In stage k, we pick a large n and add to A a nonempty set of strings B of length n such that for all nondeterministic programs p running in time $2^{\sqrt{n}}$ such that $|p| < n/4$, $p^{B \cup A}$ accepts either zero or more than one string in A. We first create a B that makes as many programs as possible accept zero strings in B. After that, we carefully remove some strings from B to guarantee that the rest of the programs accept at least two strings.

Let P be the set of nondeterministic programs of size less than $n/4$. We have $\|P\| < 2^{n/4}$. We will clock all of these programs so that they will reject if they take time more than $2^{\sqrt{n}}$. We also assume that on every program p in P, input x and oracle O, $p^O(x)$ queries x.

Let $v = 2^{\sqrt{n}+1}\|P\|$ and $w = \|P\|v2^{\sqrt{n}}$. Pick sets $\Delta \subseteq P$ and $H \subseteq \Sigma^n$ that maximize $\|\Delta\| + \|H\|$ such that $\|H\| \leq w\|\Delta\|$, and for all $X \subseteq \Sigma^n - H$ and $p \in \Delta$, $X \cap p^{A \cup X} = \emptyset$.

Note that $H \neq \Sigma^n$ since $\|H\| \leq w\|\Delta\| \leq w\|P\| \leq 2^{2\sqrt{n}+1}2^{3n/4} < 2^n$. Since some small program p always accepts, we have that $\Delta \neq P$.

Our final B will be a subset of $\Sigma^n - H$, which guarantees that for all $p \in \Delta$, $p^{A \cup B}$ will not accept any strings in B. We will create B such that for all $p \in P - \Delta$, $p^{A \cup B}$ accepts at least two strings in B. Initially let $B = \Sigma^n - H$. For each $p \in P - \Delta$ and for each integer i, $1 \leq i \leq v$, do the following:

- Pick a minimal $X \subseteq B$ such that for some $y \in X$, $p^{A \cup X}(y)$ accepts.
- Fix an accepting path and let Q_{p_i} be all the queries made on that path. Let $y_{p,i} = y$, $X_{p,i} = X$, and $B = B - Q_{p,i}$.

Note that $\|Q_{p,i}\| \leq 2^{\sqrt{n}}$. We remove no more than $\|P\|v2^{\sqrt{n}} \leq w$ strings in total. So if we cannot find an appropriate X, we have violated the maximality of $\|\Delta\| + \|H\|$. Note that $y_{p,i} \in X_{p,i} \subseteq Q_{p,i}$ and all of the $X_{p,i}$ are disjoint.

Initially set all of the $X_{p,i}$ as unmarked. For each $p \in P - \Delta$ do the following twice:

• Pick an unmarked $X_{p,i}$. Mark all $X_{q,j}$ such that $X_{q,j} \cap Q_{p,i} \neq \emptyset$.
Let $B = B \cup X_{p,i}$.

We have that $y_{p,i} \in B$ and $p^{A \cup B}(y_{p,i})$ accepts for every $X_{p,i}$ processed.

At most $2 \cdot 2^{\sqrt{n}} \|P\| - 1 < v$ of the $X_{q,j}$'s get marked before we have finished; we always can find an unmarked $X_{p,i}$.

Finally, note that $B \subseteq \Sigma^n - H$ and for every $p \in P - \Delta$ we have at least two $y \in B$ such that $p^{A \cup B}(y)$ accepts. Since $P - \Delta \neq \emptyset$, this also guarantees that $B \neq \emptyset$. Thus we have fulfilled the requirements for stage k. □

Using Theorem 6.2 we get the following corollary first proved by Goldsmith, Hemachandra, and Kunen [9].

COROLLARY 6.3 (Goldsmith–Hemachandra–Kunen). *Relative to some oracle, there exists an infinite polynomial-time computable set with no infinite sparse* **NP** *subsets.*

Proof. Let A from Theorem 6.2 be both the oracle and the set in P^A. Suppose A has an infinite sparse subset S in NP^A. Pick a large x such that $x \in S$. Applying Corollary 3.5 (3), it follows that $\mathbf{CND}^{A,p}(x) \leq O(\log(n))$. This contradicts the fact that $x \in S \subseteq A$ and Theorem 6.2. □

Actually, the above argument shows something stronger.

COROLLARY 6.4. *Relative to some oracle, there exists an infinite polynomial-time computable set with no infinite subset in* **NP** *of density less than* $2^{n/9}$.

It remains open whether Corollary 6.4 holds for $2^{\delta n}$ for $\frac{1}{9} < \delta < 1$.

7. CD vs. C and CND. This section deals with the consequences of the assumption that one of the complexity measures **C**, **CD**, and **CND** coincides for polynomial time. We will see that these assumptions are equivalent to well-studied complexity-theoretic assumptions. This allows us to apply the machinery developed in the previous sections. We will use the following function classes.

DEFINITION 7.1.

1. *The class* $\mathbf{FP}^{\mathbf{NP}[\log(n)]}$ *is the class of functions computable in polynomial time that can adaptively access an oracle in* **NP** *at most* $c\log(n)$ *times, for some* c.
2. *The class* $\mathbf{FP}_{tt}^{\mathbf{NP}}$ *is the class of functions computable in polynomial time that can nonadaptively access an oracle in* **NP**.

THEOREM 7.2. *The following are equivalent:*

1. *For all* p_2 $\exists p_1$, c *for all* $x, y : \mathbf{C}^{p_1}(x \mid y) \leq \mathbf{CND}^{p_2}(x \mid y) + c\log(|x| + |y|)$.
2. *For all* p_2 $\exists p_1$, c *for all* $x, y : \mathbf{CD}^{p_1}(x \mid y) \leq \mathbf{CND}^{p_2}(x \mid y) + c\log(|x| + |y|)$.
3. $\mathbf{FP}^{\mathbf{NP}[\log(n)]} = \mathbf{FP}_{tt}^{\mathbf{NP}}$.

Proof. (1 ⇒ 2) is trivial.

(2 ⇒ 3) We will first need the following lemma due to Lozano (see [16, pp. 184–185]).

LEMMA 7.3. $\mathbf{FP}^{\mathbf{NP}[\log(n)]} = \mathbf{FP}_{tt}^{\mathbf{NP}}$ *if and only if for every* f *in* $\mathbf{FP}_{tt}^{\mathbf{NP}}$ *there exists a function* $g \in \mathbf{FP}$ *that generates a polynomial-size set such that* $f(x) \in g(x)$.

In the following, let $f \in \mathbf{FP}_{tt}^{\mathbf{NP}}$. Let $f(x) = y$. We will see that there exists a p and c such that $\mathbf{CND}^p(y \mid x) \leq c\log(|x|)$. We can assume that the machine computing $f(x)$ produces a list of queries $Q = \{q_1, \ldots, q_l\}$ to **SAT**. Let w be the exact number of queries in Q that are in **SAT**. Thus $w = \|Q \bigcap \mathbf{SAT}\|$. Given x, consider the following $\mathbf{CND}^p$ program:

Input z.
Use $f(x)$ to generate Q.
Guess $q^1, \ldots, q^w \in Q$ that are in **SAT**

Guess satisfying assignments for $q^1, \ldots, q^w$.
Reject if not all $q^1, \ldots, q^w$ are satisfiable.
Compute $f(x)$ with $q^1, \ldots, q^w$ answered **yes**
and $q_j \in Q \setminus \{q^1, \ldots, q^w\}$ answered **no**.
Accept if and only if $f(x) = z$.

The size of the above program is $c\log(|x|)$, accepts only y, and runs in time p, for some polynomial p and constant c depending only on f. Also, it follows that all the prefixes of y have $\mathbf{CND}^p$ complexity bounded by $c\log(|x|) + \log(|x|) + O(1)$. By assumption there exists a polynomial p' and constant d such that $\mathbf{CD}^{p'}(z \mid x) \leq d\log(|x|)$ for z a prefix of y. For each of these z there is some program r such that

1. $|r| \leq d\log(|x|)$,
2. $U(r, z, x)$ accepts,
3. $U(r, u, x)$ rejects for each $u \neq z$, and
4. $U(r, u, x)$ runs in time at most $p'(|x|)$ for each $|u| \leq |x|$.

We can use the following procedure to enumerate possibilities for y.

Let $S_0 = \{\epsilon\}$.
For $m = 1$ to $|y|$.
Let S'_m consist of all strings u of length m such that
u extends some string in S_{m-1}.
Let S_m consist of all strings u in S'_m such that
there is some r, $|r| \leq d\log(|x|)$,
such that $U(r, u, x)$ accepts in $p'(|x|)$ steps, and
for all $v \in S'_m - \{u\}$, $U(r, v, x)$ does not accept in $p'(|x|)$ steps.

Note for all m, S_m and S'_m will have size bounded by $2|x|^d$, so the above algorithm runs in polynomial time. By our discussion, y will belong to $S_{|y|}$, so it follows (using Lemma 7.3) that $\mathbf{FP}^{\mathbf{NP}[\log(n)]} = \mathbf{FP}^{\mathbf{NP}}_{tt}$.

($3 \Rightarrow 1$) Let y be a string such that $\mathbf{CND}^{p'}(y \mid x) = k$. Let e be the program of length k that witnesses this. Consider the following function:

$$f(\langle e, 0^l, 0^m, x\rangle) = w_1 w_2 \ldots w_m,$$

where

$$w_i = \begin{cases} 1 & \text{if there is a } z \text{ of length } m \text{ with the } i\text{th bit equal to one} \\ & \text{such that } U_n(e, z, x) \text{ nondeterministically accepts in } l \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that if e is a **CND** program that runs in l steps, then it accepts exactly one string, w of length m. Hence $f(\langle r, 0^{p'(|y|+|x|)}, 0^{|y|}, x\rangle) = y$. It is not hard to see that, in general, f is in $\mathbf{FP}^{\mathbf{NP}}_{tt}$ and by assumption in $\mathbf{FP}^{\mathbf{NP}[\log(n)]}$ via machine M. Next, given $e = r$, $m = |y|$, $l = p'(|y| + |x|)$, x, and the $c\log(|y| + |x|)$ answers to the **NP** oracle that M makes, we can generate y in polynomial time. We have that $\mathbf{C}^p(y \mid x) \leq \mathbf{CND}^{p'}(y \mid x) + c\log(|y| + |x|)$. □

For the next corollary we will use some results from [16]. We will use the following class of limited nondeterminism defined in [6].

DEFINITION 7.4. *Let $f(n)$ be a function from $\mathbb{N} \mapsto \mathbb{N}$. The class $\mathbf{NP}[f(n)]$ denotes that class of languages that are accepted by polynomial-time bounded nondeterministic machines that, on inputs of length n, make at most $f(n)$ nondeterministic moves.*

Jenner and Torán [16] show a series of consequences of the assumption $\mathbf{FP}^{\mathbf{NP}[\log(n)]} = \mathbf{FP}_{tt}^{\mathbf{NP}}$. By Theorem 7.2 we also get these consequences from a collapse of **CD** and **CND** complexity.

COROLLARY 7.5. *If for all* p_2 $\exists p_1, c$ *for all* x, y : $\mathbf{CD}^{p_1}(x \mid y) \leq \mathbf{CND}^{p_2}(x \mid y) + c\log(|x| + |y|)$, *then for any* k

1. $\mathbf{NP}[\log^k(n)]$ *is included in* $\mathbf{P}$,
2. $\mathbf{SAT} \in \mathbf{NP}[\frac{n}{\log^k(n)}]$,
3. $\mathbf{SAT} \in \mathbf{DTIME}(2^{n^{O(1/\log\log(n))}})$,
4. *there exists a polynomial* q *such that for every* m *formulae* $\phi_1, \ldots, \phi_m$ *of* n *variables, each such that at least one is satisfiable, there exists an* i *such that* ϕ_i *is satisfiable and*

$$\mathbf{CND}^q(\phi_i | \langle \phi_1, \ldots, \phi_m \rangle) \leq O(\log\log(n+m)).$$

The last consequence simply restates one of the Jenner–Torán results in the notation of this paper.

We can use Corollary 7.5 to get a complete collapse if there is only a constant difference between **CD** and **CND** complexity.

THEOREM 7.6. *The following are equivalent:*

1. *For all* p_2 $\exists p_1, c$ *for all* $x, y : \mathbf{C}^{p_1}(x \mid y) \leq \mathbf{CND}^{p_2}(x \mid y) + c$.
2. *For all* p_2 $\exists p_1, c$ *for all* $x, y : \mathbf{CD}^{p_1}(x \mid y) \leq \mathbf{CND}^{p_2}(x \mid y) + c$.
3. $\mathbf{P} = \mathbf{NP}$.

Proof (sketch). $(1 \Rightarrow 2)$ and $(3 \Rightarrow 1)$ are easy.

$(2 \Rightarrow 3)$ By combining Corollary 7.5(4) with the assumption, we have for any formula $\phi_1, \ldots, \phi_m$, where at least one is satisfiable, that

$$\mathbf{CD}^{p_1}(\phi_i | \langle \phi_1, \ldots, \phi_m \rangle) \leq c\log\log(n+m)$$

for some satisfiable ϕ_i. We can enumerate all the programs p of length at most $c\log\log(n+m)$ and find all the formula ϕ_i such that $p(\phi_i, \langle \phi_1, \ldots, \phi_m \rangle) = 1$ and $p(\phi_j, \langle \phi_1, \ldots, \phi_m \rangle) = 0$ for $j \neq i$.

Thus, given $\phi_1, \ldots, \phi_m$, we can, in polynomial time, create a subset of size $\log^c(n+m)$ that contains a satisfiable formula if the original list did. We then apply a standard tree-pruning algorithm to find the satisfying assignment of any satisfiable formula. □

A simple modification of the proof shows that Theorem 7.6 holds if we replace the constant c with $a\log(n)$ for any $a < 1$.

For the next corollary we will need the following definition (see [7]).

DEFINITION 7.7. *A promise problem is a pair of sets* (Q, R). *A set* L *is called a solution to the promise problem* (Q, R) *if the following property holds: for all* $x(x \in Q \Rightarrow (x \in L \Leftrightarrow x \in R))$. *For any function* f, $f\mathbf{SAT}$ *denotes the set of Boolean formulas with at most* $f(n)$ *satisfying assignments for formulae of length* n.

The next theorem states that nondeterministic computations that have few accepting computations can be "compressed" to nondeterministic computations that have few nondeterministic moves if and only if $\mathbf{C}^{poly} \leq \mathbf{CD}^{poly}$.

THEOREM 7.8. *The following are equivalent:*

1. *For all* p_2 $\exists p_1, c$ *for all* $x, y : \mathbf{C}^{p_1}(x \mid y) \leq \mathbf{CD}^{p_2}(x \mid y) + c$.
2. $(1\mathbf{SAT}, \mathbf{SAT})$ *has a solution in* $\mathbf{P}$.
3. *For all time constructible* f, $(f\mathbf{SAT}, \mathbf{SAT})$ *has a solution in* $\mathbf{NP}[2\log(f(n)) + O(\log(n))]$.

Proof. ($1 \iff 2$) This was proven in [8].

($3 \Rightarrow 2$) Take $f(n) = 1$ and the fact [6] that $\mathbf{NP}[O(\log(n))] = \mathbf{P}$.

($2 \Rightarrow 3$) Let ϕ be a formula with at most $f(|\phi|)$ satisfying assignments. Lemma 3.2 yields that for every satisfying assignment a to ϕ, there exists a polynomial p such that $\mathbf{CD}^p(a \mid \phi) \leq 2\log(f(|\phi|)) + O(\log(|\phi|))$. Hence (using that $1 \iff 2$) it follows that $\mathbf{C}^{p'}(a \mid \phi) \leq 2\log(f(|\phi|)) + c\log(|\phi|)$ for some constant c and polynomial p'. The limited nondeterministic machine now guesses a $\mathbf{C}^{p'}$ program e of size at most $2\log(f(|\phi|)) + c\log(|\phi|)$, runs it (relative to ϕ), and accepts if and only if the generated string is a satisfying assignment to ϕ. ∎

COROLLARY 7.9. $\mathbf{FP}^{\mathbf{NP}[\log(n)]} = \mathbf{FP}_{tt}^{\mathbf{NP}}$ *implies the following:*

1. *For any k, the promise problem $(2^{\log^k(n)}\mathbf{SAT}, \mathbf{SAT})$ has a solution in $\mathbf{P}$.*
2. *For any k, the class of languages that is accepted by nondeterministic machines that have at most $2^{\log^k(n)}$ accepting paths on inputs of length n is included in $\mathbf{P}$.*

Proof. This follows from Theorem 7.2, Theorem 7.8, and Corollary 7.5. ∎

8. Satisfying assignments. We show several connections between **CD** complexity and finding satisfying assignments of Boolean formulae. By Cook's theorem [5], finding satisfying assignments is equivalent to finding accepting computation paths of any **NP** computation.

8.1. Enumerating satisfying assignments. Papadimitriou [22] mentioned the following proposition.

PROPOSITION 8.1. *There exists a Turing machine that, given a formula ϕ, will output the set A of satisfying assignments of ϕ in time polynomial in $|\phi|$ and $\|A\|$.*

We can use **CD** complexity to show the following.

THEOREM 8.2. *Proposition* 8.1 *is equivalent to* $(1\mathbf{SAT}, \mathbf{SAT})$ *having a solution in* $\mathbf{P}$.

In Proposition 8.1, we do not require the machine to halt after printing out the assignments. If the machine is required to halt in time polynomial in ϕ and $\|A\|$, we have that Proposition 8.1 is equivalent to $\mathbf{P} = \mathbf{NP}$.

Proof of Theorem 8.2. The implication of $(1\mathbf{SAT}, \mathbf{SAT})$ having a solution in $\mathbf{P}$ is straightforward. We concentrate on the other direction.

Let $d = \|A\|$. By Lemma 3.2 and Theorem 7.8 we have that for every element x of A, $\mathbf{C}^q(x|\phi) \leq 2\log(d) + c\log(n)$ for some polynomial q and constant c. We simply now try every program p in length-increasing order and enumerate $p(\phi)$ if it is a satisfying assignment of ϕ. ∎

8.2. Computing satisfying assignments. In this section we turn our attention to the question of the complexity of generating a satisfying assignment for a satisfiable formula [28, 12, 21, 2]. It is well known [18] that one can generate (the leftmost) satisfying assignment in $\mathbf{FP}^{\mathbf{NP}}$. A tantalizing open question is whether one can compute some (not necessarily the leftmost) satisfying assignment in $\mathbf{FP}_{tt}^{\mathbf{NP}}$. Formalizing this question, we define the function class $\mathbf{F}_{sat}$ by $f \in \mathbf{F}_{sat}$ if, when $\varphi \in \mathbf{SAT}$, $f(\varphi)$ is a satisfying assignment of φ.

The question now becomes $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} = \emptyset$? Translating this to a **CND** setting, we have the following lemma.

LEMMA 8.3. $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} \neq \emptyset$ *if and only if for all $\phi \in \mathbf{SAT}$ there exists a satisfying assignment a of ϕ such that $\mathbf{CND}^p(a \mid \phi) \leq c\log(|\phi|)$ for some polynomial p and constant c.*

Lemma 8.3 relativizes when we consider a relativized version of $\mathbf{SAT}^A$ [10] by adding a series of extra predicates $A_0, A_1, A_2, \ldots$ such that $A_n(x_1, \ldots, x_n)$ is true if $x_1, \ldots, x_n$ is in A.

Toda and Watanabe [28] showed that, relative to a random oracle, $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} \neq \emptyset$. On the other hand, Buhrman and Thierauf [4] showed that there exists an oracle where $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} = \emptyset$. Their result also holds relative to the set constructed in Theorem 6.2.

THEOREM 8.4. *Relative to the set A constructed in Theorem* 6.2, $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} = \emptyset$.

Proof. For some n, let ϕ be the formula on n variables such that $\phi(x)$ is true if and only if $x \in A$. Suppose $\mathbf{F}_{sat} \bigcap \mathbf{FP}_{tt}^{\mathbf{NP}} \neq \emptyset$. It now follows by Lemma 8.3 that there exists an $x \in A$ such that $\mathbf{CND}^{p,A}(x) \leq O(\log(|x|))$ for some polynomial p, contradicting the fact that for all $x \in A$, $\mathbf{CND}^{2^{\sqrt{|x|}},A}(x) \geq |x|/4$. □

8.3. Isolating satisfying assignments. In this section we take a Kolmogorov complexity view of the statement and proof of the famous Valiant–Vazirani lemma [27]. The Valiant–Vazirani lemma gives a randomized reduction from a satisfiable formula to another formula that, with a nonnegligible probability, has exactly one satisfying assignment.

We state the lemma in terms of Kolmogorov complexity.

LEMMA 8.5. *There is some polynomial p such that for all ϕ in* **SAT** *and all r such that $|r| = p(|\phi|)$ and $\mathbf{C}(r) \geq |r|$ there is some satisfying assignment a of ϕ such that $\mathbf{CD}^p(a|\langle \phi, r \rangle) \leq O(\log(|\phi|))$.*

The usual Valiant–Vazirani lemma follows from the statement of Lemma 8.5 by choosing r and the $O(\log(|\phi|))$-size program randomly.

We show how to derive the Valiant–Vazirani lemma from Sipser's lemma (see Lemma 3.6). Note that Sipser's result predates Valiant–Vazirani by a couple of years.

Proof of Lemma 8.5. Let $n = |\phi|$. Consider A, the set of satisfying assignments of ϕ. We can apply Lemma 3.6 conditioned on ϕ using part of r as the random strings. Let $d = \lfloor \log(\|A\|) \rfloor$. We get that every element of A has a **CD** program of length bounded by $d + c\log(n)$ for some constant c. Since two different elements from A must have different programs, we have at least $1/n^c$ of the strings of length $d + c\log(n)$ that must distinguish some assignment in A.

We use the rest of r to list n^{2c} different strings of length $d + c\log(n)$. Since r is random, one of these strings w must be a program that distinguishes some assignment a in A. We can give a **CD** program for a in $O(\log(n))$ bits by giving d and a pointer to w in r. □

9. Search vs. decision in exponential time. If $\mathbf{P} = \mathbf{NP}$, then given a satisfiable formula, one can use binary search to find the assignment.

One might expect a similar result for exponential-time computation, i.e., if $\mathbf{EXP} = \mathbf{NEXP}$, then one should find a witness of a **NEXP** computation in exponential time. However, the proof for polynomial time breaks down because as one does the binary search the input questions get too long. Impagliazzo and Tardos [15] give relativized evidence that this problem is indeed hard.

THEOREM 9.1 (see [15]). *There exists a relativized world where $\mathbf{EXP} = \mathbf{NEXP}$, but there exists a* **NEXP** *machine whose accepting paths cannot be found in exponential time.*

We can give a short proof of this theorem using Theorem 6.2.

Proof of Theorem 9.1. Let A be from Theorem 6.2.

We will encode a tally set T such that $\mathbf{EXP}^{A\oplus T} = \mathbf{NEXP}^{A\oplus T}$. Let M be a nondeterministic oracle machine such that M runs in time 2^n and, for all B, M^B is $\mathbf{NEXP}^B$-complete.

Initially let $T = \emptyset$. For every string w in lexicographic order, put 1^{2w} into T if $M^{A\oplus T}(x)$ accepts.

Let $B = A\oplus T$ at the end of the construction. Since $M(w)$ could only query strings with length at most $2^{|w|} \leq w$, this construction will give us $\mathbf{EXP}^B = \mathbf{NEXP}^B$.

We will show that there exists a $\mathbf{NEXP}^B$ machine whose accepting paths cannot be found in time that is exponential relative to B.

Consider the $\mathbf{NEXP}^B$ machine M that on input n guesses a string y of length n and accepts if y is in A. Note that M runs in time $2^{|n|} \leq n$.

Suppose accepting computations of M^B can be found in time $2^{|n|^k} = 2^{\log^k(n)}$ relative to B. By Theorem 6.2, we can fix some large n such that $A^{=n} \neq \emptyset$ and, for all $x \in A^{=n}$,

$$\mathbf{CND}^{2^{\log^k(n)},A}(x) \geq n/4. \tag{9.1}$$

Let $w_i = \|\{1^m \mid 1^m \in T \text{ and } 2^i < m \leq 2^{i+1}\}\|$. We will show the following lemma.

LEMMA 9.2.

$$\mathbf{CND}^{2^{\log^k(n)},A}\left(x|w_1,\ldots,w_{\log^k(n)}\right) \leq \log(n) + O(1).$$

Assuming Lemma 9.2, Theorem 9.1 follows since, for each i, $|w_i| \leq i+1$. We thus have our contradiction with (9.1). □

Proof of Lemma 9.2. We will construct a program p^A to nondeterministically distinguish x. We use $\log(n)$ bits to encode n. First p will reconstruct T using the w_i's.

Suppose we have reconstructed T up to length 2^i. By our construction of T, strings of T of length at most 2^{i+1} can only depend on oracle strings of length at most $2^{i+1}/2 = 2^i$. We guess w_i strings of the form 1^m for $2^i < m \leq 2^{i+1}$ and nondeterministically verify that these are the strings in T. Once we have T, we also have $B = A \oplus T$, so in time $2^{\log^k(n)}$, we can find x. □

Impagliazzo and Tardos [15] prove Theorem 9.1 using an "X-search" problem. We can also relate this problem to **CND** complexity and Theorem 6.2.

DEFINITION 9.3. *The X-search problem has a player who, given N input variables not all zero, wants to find a one. The player can ask r rounds of l parallel queries of a certain type each and wins if the player discovers a one.*

Impagliazzo and Tardos [15] use the following result about the X-search problem to prove Theorem 9.1.

THEOREM 9.4 (see [15]). *If the queries are restricted to k-DNFs and $N > 2(klr)^2(l+1)^r$, then the player will lose on some nonzero setting of the variables.*

One can use a proof similar to that of Theorem 9.1 to prove a similar bound for Theorem 9.4: one just needs to apply Theorem 6.2 relative to the strategy of the player.

One can also use Theorem 9.4 to prove a weaker version of Theorem 6.2. Pick a large n and a time bound t. Let $N = 2^n$ and suppose for all $B \subseteq \Sigma^n$ there is an x in B, $\mathbf{CND}^{t,B}(x) \leq w$. Let $N = 2^n$.

For a fixed B and x, let p be the **CND** program that distinguishes x. Nondeterministically we can find the ith bit of x using t queries to B by guessing x

and the accepting computation of $U_n(p,x)$. We can express this computation as the complement of a t-**DNF** question.

We now build a strategy for X-search: Try all p and i, $|p| \leq 2^{|p|}$ and $1 \leq i \leq n$, in the first round using the t-**DNF** described above. This gives us a list of $2^{|p|}$ possible x's. We just try them all.

This solves the X-search problem using $k = t$, $l = n2^{|w|}$, and $r = 2$. By Theorem 9.4 we have $N \leq 2(tn2^w2)^2(n2^w)^2$. Taking logarithms we get $n \leq 2(\log t + \log n + w + \log 2) + 2(\log n + w)$. Thus we have a contradiction whenever $w = an$ and $t = 2^{bn}$ for $4a + 2b < 1$. In particular this gives us an infinite set A such that $\mathbf{CND}^{2^{n^{1/6}},A}(x) \geq |x|/7$ for all x in A.

10. BPP in the second level of the polynomial hierarchy. One of the applications of Sipser's [24] randomized version of Lemma 3.2 is the proof that **BPP** is in Σ_2^p. We will show that the approach taken in Lemma 3.2 yields a new proof of this result. We will first prove the following variation of Lemma 3.1.

LEMMA 10.1. *Let $S = \{x_1, \ldots, x_d\} \subseteq \{0, \ldots, n-1\}$. There exists a prime number p such that for all $x_i, x_j \in S$ $(i \neq j), x_i \not\equiv x_j \bmod p$, and such that $p \leq 2d^2 \log(n)$.*

Proof. We consider only prime numbers between c and $2c$. For $x_i, x_j \in S$ it holds that, for at most $\log_c(n) = \frac{\log(n)}{\log(c)}$ different prime numbers p, $x_i \equiv x_j \bmod p$. Moreover, there are at most $d(d-1)$ different pairs of strings in S, so there exists a prime number p among the first $d(d-1)\frac{\log(n)}{\log(c)} + 1$ prime numbers such that, for all $x_i, x_j \in S (i \neq j)$, it holds that $x_i \not\equiv x_j \bmod p$. Applying again the prime number theorem [13], it follows that if we take $c > d(d-1)\log(n)$, then $p \leq 2d^2 \log(n)$. ∎

The idea is to use Lemma 10.1 as a way to approximate the number of accepting paths of a **BPP** machine M. Note that the set of accepting paths $\mathbf{ACCEPT}_{M(x)}$ of M on x is in **P**. If this set is "small," then there exists a prime number satisfying Lemma 10.1. On the other hand, if the set is "big," no such prime number exists. This can be verified in Σ_2^p: There exists a number p such that for all pairs of accepting paths x_i, x_j of M, $x_i \not\equiv x_j \bmod p$. In order to apply this idea, we need the gap between the number of accepting paths when x is in the set and when it is not to be a square: if x is not in the set, then $\|\mathbf{ACCEPT}_{M(x)}\| \leq k(|x|)$, and if x is in the set, $\|\mathbf{ACCEPT}_{M(x)}\| > k^2(|x|)$. We will apply Zuckerman's oblivious sampler construction [29] to obtain this gap.

THEOREM 10.2. *Let M be a probabilistic machine that witnesses that some set A is in* **BPP**. *Assume that $M(x)$ uses m random bits. There exists a machine M' that uses $3m + 9m^{1-\frac{1}{2\log^*(m)}}$ random bits such that if $x \in A$, then $\Pr[M'(x)$ accepts$] > 1 - \frac{1}{2^{2m}}$, and if $x \notin A$, then $\Pr[M'(x)$accepts$] < \frac{1}{2^{2m}}$.*

Proof. Use the sampler in [29] with $\epsilon < 1/6$, $\gamma = \frac{1}{2^{2m}}$, and $\alpha = 3m^{-\frac{1}{2\log^*(m)}}$.

Let $A \in \mathbf{BPP}$ be witnessed by probabilistic machine M. Apply Theorem 10.2 to obtain M'. The Σ_2^p algorithm for $\overline{A}$ works as follows:

input x

Guess $p \leq 2^{2m+18m^{1-\frac{1}{2\log^*(m)}}}(3m + 9m^{1-\frac{1}{2\log^*(m)}})$

If for all $u, v \in \mathbf{ACCEPT}_{M'(x)}$ $u \not\equiv v \bmod p$ **accept** else **reject**

If $x \in \overline{A}$, then $\|\mathbf{ACCEPT}_{M'(x)}\| \leq 2^{m+9m^{1-\frac{1}{2\log^*(m)}}}$, and Lemma 10.1 guarantees that the above program accepts. On the other hand, if $x \in A$, then $\|\mathbf{ACCEPT}_{M'(x)}\| >$

$2^{3m+9m^{1-\frac{1}{2\log^*(m)}}-1}$ and for every prime number

$$p \leq 2^{2m+18m^{1-\frac{1}{2\log^*(m)}}}(3m+9m^{1-\frac{1}{2\log^*(m)}})$$

there will be a pair of strings in $\mathbf{ACCEPT}_{M'(x)}$ that are not congruent modulo p. This follows because for every number $p \leq 2^{2m+18m^{1-\frac{1}{2\log^*(m)}}}(3m+9m^{1-\frac{1}{2\log^*(m)}})$ at most $2^{2m+18m^{1-\frac{1}{2\log^*(m)}}}(3m+9m^{1-\frac{1}{2\log^*(m)}})$ different u and v it holds that $u \not\equiv v \bmod p$. □

Goldreich and Zuckerman [11] independently used Zuckerman's sampler [29] to give another proof that **BPP** is in Σ_2^p.

Acknowledgments. We would like to thank José Balcázar, Leen Torenvliet, and David Zuckerman for their comments on this subject. We thank John Tromp for the current presentation of the proof of Lemma 3.2. We thank Richard Beigel, Bill Gasarch, Stuart Kurtz, Amber Settle, Leen Torenvliet, and the two anonymous referees for comments on earlier drafts.

REFERENCES

[1] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, Springer-Verlag, Berlin, 1988.

[2] H. Buhrman, J. Kadin, and T. Thierauf, *Functions computable with nonadaptive queries to NP*, Theory Comput. Syst., 31 (1998), pp. 77–92.

[3] H. Buhrman, S. Laplante, and P. Miltersen, *New bounds for the language compression problem*, in Proceedings of the 15th IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 126–130.

[4] H. Buhrman and T. Thierauf, *The complexity of generating and checking proofs of membership*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, C. Pueach and R. Reischuk, eds., Springer-Verlag, New York, 1996, pp. 75–86.

[5] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd ACM Symposium on the Theory of Computing, Shaker Heights, OH, 1971, pp. 151–158.

[6] J. Díaz and J. Torán, *Classes of bounded nondeterminism*, Math. Systems Theory, 23 (1990), pp. 21–32.

[7] S. Even, A. L. Selman, and Y. Yacobi, *The complexity of promise problems with applications to public-key cryptography*, Inform. and Control, 61 (1984), pp. 159–173.

[8] L. Fortnow and M. Kummer, *On resource-bounded instance complexity*, Theoret. Comput. Sci., 161 (1996), pp. 123–140.

[9] J. Goldsmith, L. Hemachandra, and K. Kunen, *Polynomial-time compression*, Comput. Complexity, 2 (1992), pp. 18–39.

[10] J. Goldsmith and D. Joseph, *Relativized isomorphisms of NP-complete sets*, Comput. Complexity, 3 (1993), pp. 186–205.

[11] O. Goldreich and D. Zuckerman, *Another proof that BPP subseteq PH (and more)*, report TR97-045, Electronic Colloquium on Computational Complexity, University of Trier, Trier, Germany, 1997; available via ftp from ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1997/TR97-045/Paper.ps

[12] L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman, *Computing solutions uniquely collapses the polynomial hierarchy*, SIAM J. Comput., 25 (1996), pp. 697–708.

[13] G. Hardy and E. Wright, *An Introduction to the Theory of Numbers*, 5th ed., Oxford University Press, London, 1979.

[14] A. Ingham, *The Distribution of Prime Numbers*, Cambridge Tracts in Mathematics and Mathematical Physics, Cambridge University Press, Cambridge, UK, 1932.

[15] R. Impagliazzo and G. Tardos, *Decision versus search problems in super-polynomial time*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 222–227.

[16] B. Jenner and J. Torán, *Computing functions with parallel queries to NP*, Theoret. Comput. Sci., 141 (1995), pp. 175–193.

[17] D. Knuth, *Sorting and Searching*, *The Art of Computer Programming, Sorting and Searching,* Vol. 3, 2nd ed., Addison-Wesley, Reading, MA, 1998.
[18] M. Krentel, *The complexity of optimization problem*, J. Comput. System Sci., 36 (1988), pp. 490–509.
[19] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, New York, 1997.
[20] N. Nisan and D. Zuckerman, *More deterministic simulation in logspace*, in Proceedings of the 25th ACM Symposium on the Theory of Computing, 1993, pp. 330–335.
[21] M. Ogihara, *Functions computable with limited access to NP*, Inform. Process. Lett, 58 (1996), pp. 35–38.
[22] C. Papadimitriou, *The complexity of knowledge representation*, invited presentation at the 11th Annual IEEE Conference on Computational Complexity, Philadelphia, PA, 1996.
[23] R. Raz, O. Reingold, and S. Vadhan, *Extracting all the randomness and reducing the error in Trevisan's extractors*, in Proceedings of the 29th ACM Symposium on the Theory of Computing, 1999, pp. 149–158.
[24] M. Sipser, *A complexity theoretic approach to randomness*, in Proceedings of the 15th ACM Symposium on the Theory of Computing, 1983, pp. 330–335.
[25] A. Ta-Shma, *On extracting randomness from weak random sources (extended abstract)*, in Proceedings of the 28th ACM Symposium on the Theory of Computing, 1996, pp. 276–285.
[26] L. Trevisan, *Construction of extractors using pseudo-random generators*, in Proceedings of the 29th ACM Symposium on the Theory of Computing, 1999, pp. 141–148.
[27] L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.
[28] O. Watanabe and S. Toda, *Structural analysis on the complexity of inverse functions*, Math. Systems Theory, 26 (1993), pp. 203–214.
[29] D. Zuckerman, *Randomness-optimal sampling, extractors, and constructive leader election*, in Proceedings of the 28th ACM Symposium on the Theory of Computing, 1996, pp. 286–295.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 906–918

SEPARATION OF NP-COMPLETENESS NOTIONS*

A. PAVAN† AND ALAN L. SELMAN‡

Abstract. We use hypotheses of structural complexity theory to separate various NP-completeness notions. In particular, we introduce an hypothesis from which we describe a set in NP that is $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete but not $\leq^{\mathrm{P}}_{tt}$-complete. We provide fairly thorough analyses of the hypotheses that we introduce.

Key words. Turing completeness, truth-table completeness, many-one completeness, p-selectivity, p-genericity

AMS subject classification. 68Q15

PII. S0097539701387039

1. Introduction.

Ladner, Lynch, and Selman [LLS75] were the first to compare the strength of polynomial-time reducibilities. They showed, for the common polynomial-time reducibilities, Turing ($\leq^{\mathrm{P}}_{\mathrm{T}}$), truth-table ($\leq^{\mathrm{P}}_{tt}$), bounded truth-table ($\leq^{\mathrm{P}}_{btt}$), and many-one ($\leq^{\mathrm{P}}_{m}$), that

$$\leq^{\mathrm{P}}_{m} \prec \leq^{\mathrm{P}}_{btt} \prec \leq^{\mathrm{P}}_{tt} \prec \leq^{\mathrm{P}}_{\mathrm{T}},$$

where $\leq^{\mathrm{P}}_{r} \prec \leq^{\mathrm{P}}_{s}$ means that $\leq^{\mathrm{P}}_{r}$ is *properly stronger* than $\leq^{\mathrm{P}}_{s}$; that is, $A \leq^{\mathrm{P}}_{r} B$ implies $A \leq^{\mathrm{P}}_{s} B$, but the converse does not hold. In each case, the verifying sets belong to $\mathrm{E} = \mathrm{DTIME}(2^n)$. Ladner, Lynch, and Selman raised the obvious question of whether reducibilities differ on NP. If there exist sets A and B in NP (other than the empty set or Σ^*) such that $A \leq^{\mathrm{P}}_{\mathrm{T}} B$ but $A \not\leq^{\mathrm{P}}_{m} B$, then, of course, $\mathrm{P} \neq \mathrm{NP}$ follows immediately. With this in mind, they conjectured that $\mathrm{P} \neq \mathrm{NP}$ implies that $\leq^{\mathrm{P}}_{\mathrm{T}}$ and $\leq^{\mathrm{P}}_{m}$ differ on NP.

In the intervening years, many results have explained the behavior of polynomial-time reducibilities within other complexity classes and have led to a complete understanding of the completeness notions that these reducibilities induce. For example, Ko and Moore [KM81] demonstrated the existence of $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete sets for EXP that are not $\leq^{\mathrm{P}}_{m}$-complete. Watanabe [Wat87] extended this result significantly, showing that $\leq^{\mathrm{P}}_{1-tt}$-, $\leq^{\mathrm{P}}_{btt}$-, $\leq^{\mathrm{P}}_{tt}$-, and $\leq^{\mathrm{P}}_{\mathrm{T}}$-completeness for EXP are mutually different, while Homer, Kurtz, and Royer [HKR93] proved that $\leq^{\mathrm{P}}_{m}$- and $\leq^{\mathrm{P}}_{1-tt}$-completeness are identical.

However, there have been few results comparing reducibilities within NP, and we have known very little concerning various notions of NP-completeness. It is surprising that no NP-complete problem has been discovered that requires anything other than many-one reducibility for proving its completeness. The first result to distinguish reducibilities within NP is an observation of Wilson in one of Selman's papers on p-selective sets [Sel82]. It is a corollary to results there that if NE$\cap$*co*-NE $\neq$ E, then there exist sets A and B belonging to NP such that $A \leq^{\mathrm{P}}_{ptt} B$, $B \leq^{\mathrm{P}}_{tt} A$, and $B \not\leq^{\mathrm{P}}_{ptt} A$, where

*Received by the editors April 3, 2001; accepted for publication (in revised form) July 18, 2001; published electronically February 8, 2002.
http://www.siam.org/journals/sicomp/31-3/38703.html
†NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 (apavan@research.nj.nec.com).
‡Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY 14260 (selman@cse.buffalo.edu).

$\leq^{\mathrm{P}}_{ptt}$ denotes positive truth-table reducibility. Regarding completeness, Longpré and Young [LY90] proved that there are $\leq^{\mathrm{P}}_{m}$-complete sets for NP for which $\leq^{\mathrm{P}}_{\mathrm{T}}$-reductions to these sets are *faster*, but they did not prove that the completeness notions differ. The first to give technical evidence that $\leq^{\mathrm{P}}_{\mathrm{T}}$- and $\leq^{\mathrm{P}}_{m}$-completeness for NP differ are Lutz and Mayordomo [LM96], who proved that if the p-measure of NP is not zero, then there exists a $\leq^{\mathrm{P}}_{3\text{-}tt}$-complete set that is not $\leq^{\mathrm{P}}_{m}$-complete. Ambos-Spies and Bentzien [ASB00] extended this result significantly. They used an hypothesis of resource-bounded category theory that is weaker than that of Lutz and Mayordomo to separate nearly all NP-completeness notions for the bounded truth-table reducibilities.

It has remained an open question as to whether we can separate NP-completeness notions without using hypotheses that involve essentially stochastic concepts. Furthermore, the only comparisons of reducibilities within NP known to date have been those just listed.

Here we report some exciting new progress on these questions. Our main new result introduces a strong, but reasonable, hypothesis to prove existence of a $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete set in NP that is not $\leq^{\mathrm{P}}_{tt}$-complete. Our result is the first to provide evidence that $\leq^{\mathrm{P}}_{tt}$-completeness is weaker than $\leq^{\mathrm{P}}_{\mathrm{T}}$-completeness. Let Hypothesis H be the following assertion: There is a UP-machine M that accepts 0^* such that (i) no polynomial time-bounded Turing machine correctly computes infinitely many accepting computations of M, and (ii) for some $\epsilon > 0$, no 2^{n^ϵ} time-bounded Turing machine correctly computes all accepting computations of M. Hypothesis H is similar to, but seemingly stronger than, hypotheses considered by researchers previously, notably Fenner et al. [FFNR96], Hemaspaandra, Rothe, and Wechsung [HRW97], and Fortnow, Pavan, and Selman [FPS99].

This result is especially interesting because the measure theory and category theory techniques seem to be successful primarily for the nonadaptive reducibilities. We will prove an elegant characterization of the genericity hypothesis of Ambos-Spies and Bentzien and compare it with Hypothesis H. Here, somewhat informally, let us say this: The genericity hypothesis asserts existence of a set L in NP such that no 2^{2n} time-bounded Turing machine can correctly *predict* membership of infinitely many x in L from the initial characteristic sequence $L|x = \{y \in L \mid y < x\}$. That is, L is almost-everywhere unpredictable within time 2^{2n}. Clearly, such a set L is 2^{2n}-bi-immune. In contrast, we show that Hypothesis H holds if there is a set L in UP$\cap$co-UP such that L is P-bi-immune and $L \cap 0^*$ is not in DTIME(2^{n^ϵ}) for some $\epsilon > 0$. Thus, we replace "almost-everywhere unpredictable" with P-bi-immunity and we lower the time bound from 2^{2n} to 2^{n^ϵ}, but we require L to belong to UP $\cap$ co-UP rather than NP.

We prove several other separations as well and some with significantly weaker hypotheses. For example, we prove that NP contains $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete sets that are not $\leq^{\mathrm{P}}_{m}$-complete, if NP $\cap$ co-NP contains a set that is 2^{n^ϵ}-bi-immune, for some $\epsilon > 0$.

2. Preliminaries. We use standard notation for polynomial-time reductions [LLS75], and we assume that readers are familiar with Turing, $\leq^{\mathrm{P}}_{\mathrm{T}}$, and many-one, $\leq^{\mathrm{P}}_{m}$, reducibilities. A set A is *truth-table* reducible to a set B (in symbols $A \leq^{P}_{tt} B$) if there exist polynomial-time computable functions g and h such that on input x, $g(x)$ is a set of queries $Q = \{q_1, q_2, \cdots, q_k\}$, and $x \in A$ if and only if $h(x, B(q_1), B(q_2), \cdots, B(q_k)) = 1$. The function g is the *truth-table generator* and h is the *truth-table evaluator*. For a constant $k > 0$, A is *k-truth-table* reducible to B ($A \leq^{P}_{k\text{-}tt} B$) if for all x, $\|Q\| = k$, and A is *bounded-truth-table* reducible to B ($A \leq^{\mathrm{P}}_{btt} B$) if there is a constant $k > 0$ such that $A \leq^{P}_{k\text{-}tt} B$. Given a polynomial-time reducibility $\leq^{\mathrm{P}}_{r}$, recall that a set S is

$\leq_r^{\mathrm{P}}$-complete for NP if $S \in \mathrm{NP}$ and every set in NP is $\leq_r^{\mathrm{P}}$-reducible to S.

Recall that a set L is *p-selective* if there exists a polynomial-time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all x and y, $f(x,y) \in \{x,y\}$ and $f(x,y)$ belongs to L if either $x \in L$ or $y \in L$ [Sel79]. The function f is called a *selector* for L.

Given a finite alphabet Σ, let Σ^ω denote the set of all strings of infinite length of order type ω. For $r \in \Sigma^* \cup \Sigma^\omega$, the standard left cut of r [Sel79, Sel82] is the set

$$L(r) = \{x \mid x < r\},$$

where $<$ is the ordinary dictionary ordering of strings with 0 less than 1. It is obvious that every standard left cut is p-selective with selector $f(x,y) = \min(x,y)$.

Given a p-selective set L such that the function f defined by $f(x,y) = \min(x,y)$ is a selector for L, we call f a *min-selector* for L. We will use the following simplified version of a lemma of Toda [Tod91]. Let $\perp$ be a special symbol such that $\perp \leq x$ for all $x \in \Sigma^*$.

LEMMA 2.1. *Let L be a p-selective set with a min-selector f. For any finite set Q there exists a string $z \in Q \cup \{\perp\}$ such that $Q \cap L = \{y \in Q \mid y \leq z\}$ and $Q \cap \overline{L} = \{y \in Q \mid y > z\}$. The string z is called a "pivot" string.*

Now we review various notions related to almost-everywhere hardness. A language L is *immune* to a complexity class $\mathcal{C}$, or $\mathcal{C}$-*immune*, if L is infinite and no infinite subset of L belongs to $\mathcal{C}$. A language L is *bi-immune* to a complexity class $\mathcal{C}$, or $\mathcal{C}$-*bi-immune*, if L and $\overline{L}$ are both immune to $\mathcal{C}$. Then, L is *bi-immune* to $\mathcal{C}$ if and only if L is infinite, no infinite subset of L belongs to $\mathcal{C}$, and no infinite subset of $\overline{L}$ belongs to $\mathcal{C}$. A language is $\mathrm{DTIME}(T(n))$-*complex* if L does not belong to $\mathrm{DTIME}(T(n))$ almost everywhere; that is, every Turing machine M that accepts L runs in time greater than $T(|x|)$ for all but finitely many words x. Balcázar and Schöning [BS85] proved that for every time-constructible function T, L is $\mathrm{DTIME}(T(n))$-complex if and only if L is bi-immune to $\mathrm{DTIME}(T(n))$.

Given a time bound $T(n)$, a language L is $T(n)$-*printable* if there exists a $T(n)$ time-bounded Turing machine that, on input 0^n, prints all elements of $L \cap \Sigma^{=n}$ [HY84]. A set S is $T(n)$-*printable-immune* if S is infinite and no infinite subset of S is $T(n)$-printable.

In order to compare our hypotheses with the genericity hypothesis we describe time-bounded genericity [ASFH87]. For this purpose, we follow the exposition of Ambos-Spies, Neis, and Terwijn [ASNT96]. Given a set A and string x, $A|x = \{y \mid y < x \text{ and } y \in A\}$. Let $\Sigma^* = \{z_n\}_n$, where z_n is the nth string in lexicographic order. We identify the initial segment $A|z_n$ with its characteristic sequence; i.e., $A|z_n = A(z_0) \cdots A(z_{n-1})$. A *condition* is a set $C \subseteq \Sigma^*$. *A meets C* if for some x, the characteristic sequence $A|x \in C$. C is *dense along A* if for infinitely many strings x there exists $i \in \{0,1\}$ such that the concatenation $(A|x)i \in C$. Then, the set A is $\mathrm{DTIME}(t(n))$-*generic* if A meets every condition $C \in \mathrm{DTIME}(t(n))$ that is dense along A. To simplify the notation, we say that A is $t(n)$-*generic* if it is $\mathrm{DTIME}(t(n))$-generic.

Finally, we briefly describe the Kolmogorov complexity of a finite string. Later we will use this in an oracle construction. The interested reader should refer to Li and Vitányi [LV97] for an in-depth study. Fix a universal Turing machine U. Given a string x and a finite set $S \subseteq \Sigma^*$, the *Kolmogorov complexity of x with respect to S* is defined by

$$K(x|S) = \min\{|p| \mid U(p,S) = x\}.$$

If $S = \emptyset$, then $K(x|S)$ is called the *Kolmogorov complexity* of x, denoted $K(x)$. We will also use time-bounded Kolmogorov complexity $K^t(x)$. For this definition, we require that $U(p)$ runs in at most $t(|x|)$ steps.

3. Separation results. Let Hypothesis H be the following assertion.

Hypothesis H. There is a UP-machine M that accepts 0^* such that

1. no polynomial time-bounded Turing machine correctly computes infinitely many accepting computations of M, and
2. for some $\epsilon > 0$, no 2^{n^ϵ} time-bounded Turing machine correctly computes all accepting computations of M.

THEOREM 3.1. *If Hypothesis* H *is true, then there exists a* $\leq^{\mathrm{P}}_{\mathrm{T}}$*-complete language for* NP *that is not* $\leq^{\mathrm{P}}_{tt}$*-complete for* NP.

Proof. Let $\epsilon > 0$, and let M be a UP-machine that satisfy the conditions of Hypothesis H. For each $n \geq 0$, let a_n be the unique accepting computation of M on 0^n, and let $l_n = |a_n|$. Define the language

$$L_1 = \{\langle x, a_n\rangle \mid |x| = n, \text{ and } x \in \mathrm{SAT}\}.$$

Define the infinite string $a = a_1 a_2 \ldots$, and define

$$L_2 = L(a) = \{x \mid x < a\}$$

to be the standard left-cut of a.

We define $L = L_1 \oplus L_2$ to be the disjoint union of L_1 and L_2. We will prove that L is $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete for NP but not $\leq^{\mathrm{P}}_{tt}$-complete.

LEMMA 3.2. *L is $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete for* NP.

Proof. It is clear that L belongs to NP. The following reduction witnesses that $\mathrm{SAT} \leq^{\mathrm{P}}_{\mathrm{T}} L$: Given an input string x, where $|x| = n$, use a binary search algorithm that queries L_2 to find a_n. Then, note that $x \in \mathrm{SAT}$ if and only if $\langle x, a_n\rangle$ belongs to L_1. □

LEMMA 3.3. *L is not $\leq^{\mathrm{P}}_{tt}$-complete for* NP.

Proof. Assume that L is $\leq^{\mathrm{P}}_{tt}$-complete for NP. Define the set

$$S = \{\langle 0^n, i\rangle \mid \text{ the } i\text{th bit of } a_n = 1\}.$$

Clearly, S belongs to NP. Thus, by our assumption, there is a $\leq^{\mathrm{P}}_{tt}$-reduction $\langle g, h\rangle$ from S to L. Given this reduction, we will derive a contradiction to Hypothesis H.

Consider the following procedure $\mathcal{A}$:

1. Input 0^n.
2. Compute the sets $Q^j = g(\langle 0^n, j\rangle)$ for $1 \leq j \leq l_n$. Let $Q = \bigcup\{Q^j \mid 1 \leq j \leq l_n\}$.
3. Let Q_1 be the set of all queries in Q to L_1, and let Q_2 be the set of all queries in Q to L_2 ($Q = Q_1 \cup Q_2$).
4. If Q_1 contains a query $\langle x, a_t\rangle$, where $t \geq n^\epsilon$, then output "Unsuccessful" and Print a_t, else output "Successful."

Observe that this procedure runs in polynomial time. We treat two cases, namely, $\mathcal{A}(0^n)$ is either unsuccessful, for infinitely many n, or it is successful for all but finitely many n.

CLAIM 1. *If the procedure $\mathcal{A}(0^n)$ is unsuccessful for infinitely many n, then there is a polynomial time-bounded Turing machine that correctly computes infinitely many accepting computations of M, thereby contradicting Clause* 1 *of Hypothesis* H.

Proof. If $\mathcal{A}(0^n)$ is unsuccessful, then it outputs a string a_t such that $t \geq n^\epsilon$. Hence, if $\mathcal{A}(0^n)$ is unsuccessful for infinitely many n, then for infinitely many t there

exists an n, where $n \leq t^{1/\epsilon}$, and $\mathcal{A}(0^n)$ outputs a_t. The following procedure uses this observation to compute infinitely many accepting computations of M in polynomial time.

input 0^t;
for $j = 1$ to $j = t^{1/\epsilon}$ **do**
    if $\mathcal{A}(0^j)$ outputs a_t
        then output a_t and halt.

The procedure runs in polynomial time because the procedure $\mathcal{A}(0^j)$ runs in polynomial time. □

CLAIM 2. *If $\mathcal{A}(0^n)$ is successful for all but finitely many n, then there is a 2^{n^ϵ} time-bounded Turing machine that correctly computes all accepting computations of M, thereby contradicting Clause* 2 *of Hypothesis* H.

Proof. We will demonstrate a procedure $\mathcal{B}$ such that for each n, if $\mathcal{A}(0^n)$ is successful, then $\mathcal{B}$ on input 0^n outputs the accepting computation of M on 0^n in 2^{n^ϵ} time.

If $\mathcal{A}(0^n)$ is successful, then no member of the set Q_1 is of the form $\langle x, a_t\rangle$ where $t \geq n^\epsilon$. We begin our task with the following procedure $\mathcal{C}$ that for each query $q = \langle y, z\rangle$ in Q_1 decides whether $q \in L_1$.

1. Input $q = \langle y, z\rangle$.
2. If $z \neq a_t$ for any t, then $\langle y, z\rangle$ does not belong to L_1. (This can be determined in polynomial time.)
3. If $z = a_t$, where $t \leq n^\epsilon$, then $\langle y, z\rangle$ belongs to L_1 only if $|y| = t$ and y belongs to SAT. (Since $t \leq n^\epsilon$ this step can be done in time 2^{n^ϵ}.)

Thus, $\mathcal{C}$ decides membership in L_1 for all queries q in Q_1. Therefore, if for each query q in Q_2, we can decide whether q belongs to L_2, then the evaluator h can determine whether each input $\langle 0^n, j\rangle$, $1 \leq j \leq l_n$, belongs to S. That is, if for each query q in Q_2, we can decide whether q belongs to L_2, then we can compute a_n. We can accomplish this using a standard proof technique for p-selective sets [HNOS96, Tod91]. Namely, since L_2 is a standard left-cut, by Lemma 2.1, there exists a pivot string z in $Q_2 \cup \{\perp\}$ such that $Q_2 \cap L_2$ is the set of all strings in Q_2 that are less than or equal to z. We do not know which string is the pivot string, but there are only $\|Q_2\|$ choices, which is bounded by a polynomial in n. Thus, procedure $\mathcal{B}$ on input 0^n proceeds as follows to compute a_n: For each possible choice of pivot and the output from procedure $\mathcal{C}$, the evaluator h computes a possible value for each jth bit of a_n. There are only a polynomial number of possible choices of a_n because there are only a polynomial number of pivots. $\mathcal{B}$ verifies which choice is the correct accepting computation of M on 0^n and outputs that value. Finally, we have only to note that the entire process can be carried out in 2^{n^ϵ} steps. This completes the proof of our claim. □

Lemma 3.3 follows from Claims 1 and 2, and the theorem follows from Lemmas 3.2 and 3.3. □

Let Hypothesis H′ be the following assertion.

Hypothesis H′. There is an NP-machine M that accepts 0^* such that for some $0 < \epsilon < 1$, no 2^{n^ϵ} time-bounded Turing machine correctly computes infinitely many accepting computations of M.

THEOREM 3.4. *If Hypothesis* H′ *is true, then there exists a Turing complete language for* NP *that is not* $\leq_m^{\mathrm{P}}$*-complete for* NP.

Proof. Let $0 < \epsilon < 1$, and let M be an NP-machine that satisfy the conditions of Hypothesis H′. For each $n \geq 0$, let a_n be the lexicographically maximum accepting

input 0^n;
$y := \lambda$;
Repeat l_n times
  begin
  $x_0 := f(\langle 0^n, y0\rangle)$;
  $x_1 := f(\langle 0^n, y1\rangle)$;
  if both x_0 and x_1 are queries to L_2
    then if $x_0 \le x_1$
      then $y := y0$
      else $y := y1$
    else {At least one of x_0 and x_1 is a query to L_1; let $b \in \{0,1\}$ be the least index such that x_b queries L_1, and let $x_b = \langle z, u\rangle$.}
      if u is not an accepting computation of M {thus, $x_b \notin L_1$}
        then $y = y\bar{b}$
        else {u is an accepting computation of M on some string 0^t}
          if $t \ge n^\epsilon$
            then output "Unsuccessful," print u, and terminate
            else $\{t < n^\epsilon\}$
              if $|z| = t^\epsilon/2$ and $z \in$ SAT {thus, $x_b \in L_1$}
                then $y := yb$
                else $\{x_b \notin L_1\}$ $y := y\bar{b}$
  end;
output "Successful" and print y.

FIG. 3.1. *Procedure* $\mathcal{D}$.

computation of M on 0^n, and let $|a_n| = l_n$. Define the language

$$L_1 = \{\langle x, u\rangle \mid |x| = n,\ u \text{ is an accepting computation of } M \text{ on } 0^m,\ n = m^\epsilon/2,\ \text{and } x \in \text{SAT}\}.$$

Let $a = a_1 a_2 a_3 \cdots$, and define

$$L_2 = L(a) = \{x \mid x < a\}.$$

Define $L = L_1 \oplus L_2$.

It is easy to see, as in the previous argument, that L is $\le_{\mathrm{T}}^{\mathrm{P}}$-complete for NP. In order to prove that L is not $\le_m^{\mathrm{P}}$-complete, we define the set

$$S = \{\langle 0^n, y\rangle \mid y \text{ is a prefix of an accepting computation of } M \text{ on } 0^n\},$$

which belongs to NP, and assume there is a $\le_m^{\mathrm{P}}$-reduction f from S to L. Consider the procedure $\mathcal{D}$ in Figure 3.1: First we will analyze the running time and then we treat two cases, namely, $\mathcal{D}(0^n)$ is either successful for infinitely many n, or it is unsuccessful for all but finitely many n.

CLAIM 3. *The above procedure halts in* $O(l_n 2^{n^{\epsilon^2}/2})$ *steps.*

Proof. Consider an iteration of the repeat loop. The most expensive step is the test of whether "$z \in$ SAT." This test occurs only when $|z| = t^\epsilon/2$ and $t < n^\epsilon$. Hence we can decide whether z belongs to SAT in $2^{n^{\epsilon^2}/2}$ steps. All other steps take polynomial time. Hence the time taken by the procedure is $O(l_n 2^{n^{\epsilon^2}/2})$. ∎

Since $0 < \epsilon < 1$, the running time of procedure $\mathcal{D}$ is bounded by 2^{n^ϵ}.

CLAIM 4. *If $\mathcal{D}(0^n)$ is successful for infinitely many n, then there is a 2^{n^ϵ}-time-bounded Turing machine that correctly computes infinitely many accepting computations of M.*

Proof. We demonstrate that if $\mathcal{D}$ is successful on an input 0^n, then the string that is printed is an accepting computation of M on 0^n. In order to accomplish this, we prove by induction that y is a prefix of an accepting computation of M on 0^n during every iteration of the repeat loop (i.e., a loop invariant). Initially, when $y = \lambda$, this is true. Assume that y is a prefix of an accepting computation of M at the beginning of an iteration. Then, at least one of $f(\langle 0^n, y0\rangle) = x_0$, $f(\langle 0^n, y1\rangle) = x_1$ must belong to L. If both x_0 and x_1 are queries to L_2, then the smaller of x_0 and x_1 belongs to L_2 because L_2 is p-selective. Thus, in this case, the procedure extends y correctly. If at least one of x_0 and x_1 is a query to L_1, then the procedure determines whether $x_b \in L_1$, where x_b is the query to L_1 with least index. If x_b belongs to L, then $\langle 0^n, yb\rangle \in S$. Hence, yb is a prefix of an accepting computation. If $x_b \notin L$, then $x_{\bar{b}}$ belongs to L because at least one of x_b or $x_{\bar{b}}$ belongs to L. Thus, in this case, $y\bar{b}$ is a prefix of an accepting computation. This completes the induction argument.

The loop repeats l_n times. Therefore, the final value of y, which is the string that $\mathcal{D}$ prints, is an accepting computation. □

CLAIM 5. *If $\mathcal{D}(0^n)$ is unsuccessful for all but finitely many n, then there is a 2^{n^ϵ}-time-bounded Turing machine that correctly computes infinitely many accepting computations of M.*

Proof. The proof is similar to the proof of Claim 1. The following procedure computes infinitely many accepting computations of M.

input 0^n;
for $j = 1$ to $j = n^{1/\epsilon}$ **do**
    if $\mathcal{D}(0^j)$ outputs u and u is an accepting computation of M on 0^n
        then print u and terminate.

The running time of this algorithm can be bounded as follows: The procedure $\mathcal{D}(0^j)$ runs in time $l_j 2^{j^{\epsilon^2}/2}$ steps. So the total running time is $\sum_1^{n^{1/\epsilon}} l_j 2^{j^{\epsilon^2}/2} = O(2^{n^\epsilon})$. □

Since the cases treated both by Claims 4 and 5 demonstrate Turing machines that correctly compute infinitely many accepting computations of M in 2^{n^ϵ} time, we have a contradiction to Hypothesis H′. Thus L is not $\leq_m^{\mathrm{P}}$-complete for NP. □

The following results give fine separations of polynomial-time reducibilities in NP from significantly weaker hypotheses. Moreover, they follow readily from results in the literature.

THEOREM 3.5. *If there is a tally language in* UP − P, *then there exist two languages L_1 and L_2 in* NP *such that $L_1 \leq_{tt}^{\mathrm{P}} L_2$, $L_2 \leq_{\mathrm{T}}^{\mathrm{P}} L_1$, but $L_1 \not\leq_{btt}^{\mathrm{P}} L_2$.*[1]

Proof. Let L be a tally language in UP − P. Let R be the polynomial-time computable relation associated with the language L. Define

$$L_1 = \{\langle 0^n, y\rangle \mid \exists w, R(0^n, w) \text{ and } y \leq w\}$$

and

$$L_2 = \{\langle 0^n, i\rangle \mid \exists w, R(0^n, w) \text{ and } i\text{th bit of } w \text{ is one}\}.$$

[1] The class of all languages that are $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to L_1 is a noncollapsing degree.

It is clear that L_1 is $\leq^{\mathrm{P}}_{tt}$-reducible to L_2. To see that L_2 is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to L_1, implement a binary search algorithm that accesses L_1 to determine the unique witness w such that $R(0^n, w)$ and then find the ith bit.

Observe that L_2 is a sparse set. Ogiwara and Watanabe [OW91] call L_1 the *left set* of L, and they and Homer and Longpré [HL94] proved for every L in NP that if the left set of L is $\leq^{\mathrm{P}}_{btt}$-reducible to a sparse set, then L is in P. Hence $L_1 \not\leq_{btt} L_2$. □

We now prove that Turing and truth-table reducibilities also differ in NP under the same hypothesis.

THEOREM 3.6. *If there is a tally language in* $\mathrm{UP} - \mathrm{P}$*, then there exist two languages* L_1 *and* L_2 *in* NP *such that* $L_1 \leq^{\mathrm{P}}_{\mathrm{T}} L_2$*, but* $L_1 \not\leq^{\mathrm{P}}_{tt} L_2$*.*

Proof. Hemaspaandra et al. [HNOS96] proved that the hypothesis implies existence of a tally language L in $\mathrm{UP}-\mathrm{P}$ such that L is not $\leq^{\mathrm{P}}_{tt}$-reducible to any p-selective set. In the same paper they also showed, given a tally language L in $\mathrm{NP} - \mathrm{P}$, how to obtain a p-selective set S such that L is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to S. Combining the two results we obtain the theorem. □

4. Analysis of the hypotheses. This section contains a number of results that help us to understand the strength of Hypotheses H and H′.

4.1. Comparisons with other complexity-theoretic assertions. We begin with some equivalent formulations of these hypotheses and then relate them to other complexity-theoretic assertions. The question of whether P contains a P-printable-immune set was studied by Allender and Rubinstein [AR88], and the equivalence of items 1 and 3 in the following theorem is similar to results of Hemaspaandra, Rothe, and Wechsung [HRW97] and Fortnow, Pavan, and Selman [FPS99]. The second item is similar to the characterization of Grollmann and Selman [GS88] of *one-one, one-way* functions with the addition of the attribute *almost-always one-way* of Fortnow, Pavan, and Selman.

THEOREM 4.1. *The following statements are equivalent:*

1. *There is a language* L *in* P *that contains exactly one string of every length such that* L *is* P*-printable-immune and, for some* $\epsilon > 0$*,* L *is not* 2^{n^ϵ}*-printable.*
2. *There exists a polynomial-bounded, one-one, function* $f : 0^* \to \Sigma^*$ *such that* f *is almost-everywhere not computable in polynomial time, for some* $\epsilon > 0$*,* f *is not computable in time* 2^{n^ϵ}*, and the graph of* f *belongs to* P.[2]
3. *Hypothesis* H *is true for some* $\epsilon > 0$*.*

Proof. Let L satisfy item 1. In order to prove item 2, define

$$f(0^n) = \text{ the unique string of length } n \text{ that belongs to } L.$$

Clearly, f is polynomial-bounded and one-one. The graph of f belongs to P because L belongs to P. Suppose that M is a Turing machine that computes f and that runs in polynomial time on infinitely many inputs. Then, on these inputs, M prints $L \cap \Sigma^n$. Similarly, f is not computable in time 2^{n^ϵ}.

Let f satisfy item 2. To prove that item 3 holds, define a UP-machine M to accept 0^* as follows: On input 0^n, M guesses a string y of length within the polynomial bound of f and accepts if and only if $\langle 0^n, y\rangle \in \mathit{graph}(f)$. The rest of the proof is clear.

Now we prove that item 3 implies item 1. Let M be a UP-machine that satisfies item 3, i.e., that satisfies the conditions of Hypothesis H. Let a_n be the unique accepting computation of M on 0^n, and let $|a_n| = n^l$. Let r_n be the rank of a_n among

[2] f is *polynomial-bounded* if for some polynomial q, for all x, $|f(x)| \leq q(|x|)$.

all strings of length n^l. Now we define L as follows: Given a string x, if $|x| = n^l$ for some n, then x belongs to L if and only if $x = a_n$. If $(n-1)^l < |x| < n^l$, then x belongs to L if and only if the rank of x (among all the string of length $|x|$) is r_{n-1}. It is clear that $L \in \mathrm{P}$ and has exactly one string per each length. We claim that L is P-printable-immune and is not 2^{n^ρ}-printable, where $\epsilon = l\rho$. Any machine that prints infinitely many strings of L in polynomial time can be used to print infinitely many accepting computations of M in polynomial time. Thus L is P-printable-immune. Any machine that prints all the strings of L in 2^{n^ρ} time can be used to print all the accepting computations of M in 2^{n^ϵ} time. Thus L is not 2^{n^ρ}-printable. □

We prove the following theorem similarly.

THEOREM 4.2. *The following statements are equivalent:*

1. *There is a language L in* P *that contains* at least *one string of every length such that, for some $\epsilon > 0$, L is 2^{n^ϵ}-printable-immune.*
2. *There is a polynomial-bounded, multivalued function $f : 0^* \to \Sigma^*$ such that every refinement of f is almost-everywhere not computable in 2^{n^ϵ}-time, and the graph of f belongs to* P.
3. *Hypothesis* H′ *holds for some $\epsilon > 0$.*

Next we compare our hypotheses with the following complexity-theoretic assertions:

1. For some $\epsilon > 0$, there is a P-bi-immune language L in UP ∩ co-UP such that $L \cap 0^*$ is not in DTIME(2^{n^ϵ}).
2. For some $\epsilon > 0$, there is a language L in UP ∩ co-UP such that L is not in DTIME(2^{n^ϵ}).
3. For some $\epsilon > 0$, there is a 2^{n^ϵ}-bi-immune language in NP ∩ co-NP.

THEOREM 4.3. *Assertion* 1 *implies Hypothesis* H *and Hypothesis* H *implies assertion* 2.

Proof. Let L be a language in UP ∩ co-UP that satisfies assertion 1. Define M to be the UP-machine that accepts 0^* as follows: On input 0^n, nondeterministically guess a string w. If w either witnesses that 0^n is in L or witnesses that 0^n is in $\overline{L}$, then accept 0^n. It is immediate that M satisfies the conditions of Hypothesis H.

To prove the second implication, let M be a UP-machine that satisfies the conditions of Hypothesis H. Let a_n denote the unique accepting computation of M on 0^n and define

$$L = \{\langle 0^n, x\rangle \mid x \leq a_n\}.$$

It is clear that $L \in \mathrm{UP} \cap \text{co-UP}$. If $L \in \mathrm{DTIME}(2^{n^\epsilon})$, then a binary search algorithm can correctly compute a_n, for every n, in time 2^{n^ϵ}. This would contradict Hypothesis H. Hence, $L \notin \mathrm{DTIME}(2^{n^\epsilon})$. □

The discrete logarithm problem is an interesting possible witness for assertion 2. The best known deterministic algorithm requires time greater than $2^{n^{\frac{1}{3}}}$ [Gor93]. Thus, the discrete logarithm problem is a candidate witness for the noninclusion UP ∩ co-UP $\not\subseteq$ DTIME(2^{n^ϵ}) for any $0 < \epsilon \leq \frac{1}{3}$.

COROLLARY 1. *If, for some $\epsilon > 0$,* UP ∩ co-UP *has a 2^{n^ϵ}-bi-immune language, then $\leq^{\mathrm{P}}_{\mathrm{T}}$-completeness is different from $\leq^{\mathrm{P}}_{tt}$-completeness for* NP.

THEOREM 4.4. *Assertion* 3 *implies Hypothesis* H′.

COROLLARY 2. *If, for some $\epsilon > 0$,* NP ∩ co-NP *has a 2^{n^ϵ}-bi-immune language, then $\leq^{\mathrm{P}}_{\mathrm{T}}$-completeness is different from $\leq^{\mathrm{P}}_{m}$-completeness for* NP.

4.2. Comparisons with genericity. The genericity hypothesis of Ambos-Spies and Bentzien [ASB00], which they used successfully to separate NP-completeness notions for the bounded-truth-table reducibilities, states that "NP contains an n^2-generic language." Our next result enables us to compare this with our hypotheses.

We say that a deterministic oracle Turing machine M is a *predictor* for a language L if for every input word x, M decides whether $x \in L$ with oracle $L|x$. L is *predictable in time* $t(n)$ if there is a $t(n)$ time-bounded predictor for L. We define a set L to be *almost-everywhere unpredictable in time* $t(n)$ if every predictor for L requires more than $t(n)$ time for all but finitely many x. This concept obviously implies DTIME($t(n)$)-complex almost everywhere, but the converse does not hold.

PROPOSITION 4.5. EXP *contains languages that are* DTIME(2^n)*-complex but* not *almost-everywhere unpredictable in time* 2^n.

Let $L \in$ EXP be DTIME(2^n)-complex [GHS91]. Then $L \oplus L$ satisfies the conditions of the proposition. Now we state our characterization of $t(n)$-genericity.

THEOREM 4.6. *Let $t(n)$ be a polynomial. A decidable language L is $t(n)$-generic if and only if it is almost-everywhere unpredictable in time $t(2^n - 1)$.*

Proof. Assume that L is not almost-everywhere unpredictable in time $t(2^n - 1)$, and let M be a predictor for L that for infinitely many strings x runs in time $t(2^n - 1)$. Define a condition C so that the characteristic sequence

$$(L|x)\overline{x} \in C \Leftrightarrow M \text{ with oracle } L|x \text{ runs in time } t(2^{|x|} - 1) \text{ on input } x,$$

where $\overline{x} = \neg(M \text{ accepts } x)$. Then, C is dense along L because M correctly predicts whether $x \in L$ for infinitely many x. It is easy to see that $C \in$ DTIME($t(n)$). However, L is not $t(n)$-generic because we defined C so that L does not meet C.

Assume that L is not $t(n)$-generic, and let $C \in$ DTIME($t(n)$) be a condition that is dense along L such that L does not meet C. Let T be a deterministic Turing machine that halts on all inputs and accepts L. Define a predictor M for L to behave as follows on input x with oracle $A|x$: If $(A|x)1 \in C$, then M rejects x, and if $(A|x)0 \in C$, then M accepts x. If neither holds, then M determines membership in L by simulating T on x. Since L does not meet C, M is a predictor for L. Since C is dense along L and L does not meet C, for infinitely many x, either $(A|x)1 \in C$ or $(A|x)0 \in C$, and in each of these cases, M runs for at most $t(2 \cdot 2^{|x|})$ steps. Since $t(n)$ is a polynomial function, by the linear speedup theorem [HS65], there is a Turing machine that is equivalent to M that runs in time $t(2^{|x|} - 1)$. □

COROLLARY 4.7. NP *contains an n^2-generic language if and only if* NP *contains a set that is almost-everywhere unpredictable in time* 2^{2n}.

By Theorem 4.4, Hypothesis H′ holds if NP ∩ co-NP contains a set that, for some $\epsilon > 0$, is 2^{n^ϵ}-bi-immune. So, Hypothesis H′ requires bi-immunity, which is weaker than almost-everywhere unpredictability, and the time bound is reduced from 2^{2n} to 2^{n^ϵ}. On the other hand, we require the language to belong to NP ∩ co-NP instead of NP. Similarly, when we consider Hypothesis H, we require the language to be P-bi-immune and not in DTIME(2^{n^ϵ}), whereas now we require the language to be in UP ∩ co-UP. Moreover, the conclusion of Theorem 3.1 is not known to follow from the genericity hypothesis. At the same time, we note that the genericity hypothesis separates several bounded-truth-table completeness notions in NP that do not seem obtainable from our hypotheses.

4.3. Relativization.

THEOREM 4.8. *There exists an oracle relative to which the polynomial hierarchy is infinite and Hypotheses* H *and* H′ *both hold.*

Proof. Define Kolmogorov random strings $r_0, r_1, \ldots$ as follows: r_n is the first string of length n such that

$$K^{2^n}(r_n \mid r_0, r_1, \ldots, r_{n-1}) > n/2.$$

Then, define the oracle $A = \{r_n \mid n \geq 0\}$.

Define M to be an oracle Turing machine that accepts 0^* with oracle A as follows: On input 0^n, guess a string y of length n. If $y \in A$, then accept. M is a UP^A-machine that accepts 0^* because A contains exactly one string of every length.

Now we show that no 2^{n^ϵ} oracle Turing machine with oracle A, for any $0 < \epsilon < 1$, correctly computes infinitely many accepting computations of M. Observe that relative to A, this implies both Hypotheses H and H$'$. Suppose otherwise, and let T be such an oracle Turing machine. The gist of the remainder of the proof is that we will show how to simulate T without using the oracle, and that will contradict the randomness of r_n.

Suppose that $T^A(0^n) = r_n$. Let $l = 3n^\epsilon$. Then we simulate this computation without using an oracle as follows:

1. Compute $r_0, r_1, \ldots, r_{l-1}$. Do this iteratively: Compute r_i by running every program (with input strings $r_0, r_1, \ldots, r_{i-1}$) of length $\leq i/2$ for 2^i steps. Then r_i is the first string of length i that is not output by any of these programs. Note that the total time for executing this step is

$$l 2^{l/2} 2^l \leq l 2^{3l/2} \leq 2^{5n^\epsilon}.$$

2. Simulate T on input 0^n, except replace all oracle queries q by the following rules: If $|q| < l$, answer using the previous computations. Otherwise, just answer "no."

If the simulation is correct (i.e., the rules in step 2 correctly answer each query q to A), then this procedure outputs r_n without using the oracle. The running time of this procedure on input 0^n is $2^{5n^\epsilon} + 2^{n^\epsilon}$, which is less than 2^n. So we can describe r_n by a string of length $O(\log n)$, namely a description of T and 0^n. This contradicts the definition of r_n.

We need to show that the simulation is correct. The simulation can only be incorrect if $|q| \geq l$ and $q = r_m$ for some $m > l$. Let r_m be the first such query. This yields a short description of r_m, given $r_0, r_1, \ldots, r_{l-1}$. Namely, the description consists of the description of T (a constant), the description of 0^n ($\log n$ bits), and the description of the number j such that $q = r_m$ is the jth query (at most n^ϵ). Thus, the length of the description is $O(n^\epsilon)$. Since $l = 3n^\epsilon$, it follows that the length of the description of r_m is less than $m/2$. The running time of T, given $r_0, r_1, \ldots, r_{l-1}$, is 2^{n^ϵ}, which is less than 2^m. (The reason is that the first step in the simulation of T is not needed.) Therefore, the simulation is correct.

Finally, because A is a sparse set, using results of Balcázar, Book, and Schöning [BBS86], there is an oracle relative to which the hypotheses holds and the polynomial hierarchy is infinite. □

Hypothesis H fails relative to any oracle for which P = NP ∩ co-NP [BGS75]. Fortnow and Rogers [FR94] obtained an oracle relative to which NP ≠ co-NP and Hypothesis H$'$ fails. We know of no oracle relative to which P ≠ NP, and every $\leq^{\mathrm{P}}_{\mathrm{T}}$-complete set is $\leq^{\mathrm{P}}_m$-complete.

4.4. Extensions. The extensions in this section are independently observed by Regan and Watanabe [RW01]. In Hypothesis H we can replace the UP-machine by

an NP-machine under a stronger intractability assumption. Consider the following hypothesis: There is an NP-machine M that accepts 0^* such that

1. no probabilistic polynomial time-bounded Turing machine correctly outputs infinitely many accepting computations with nontrivial (inverse polynomial) probability, and
2. for some $\epsilon > 0$, no 2^{n^ϵ} time-bounded Turing machine correctly computes all accepting computations with nontrivial probability.

We can prove that Turing completeness is different from truth-table completeness in NP under the above hypothesis. The proof uses the randomized reduction of Valiant and Vazirani [VV86] that isolates the accepting computations. We define L as in the proof of Theorem 3.4. Let

$$S = \{\langle 0^n, k, r_1, r_2, \cdots, r_k, i\rangle \mid \exists v \text{ such that } v \text{ is an accepting computation of } M,$$
$$v.r_1 = v.r_2 = \cdots = v.r_k = 0, \text{ and the } i\text{th bit of } v = 1\},$$

where $v.r_i$ denotes the inner product over GF[2].

Valiant and Vazirani showed that if we randomly pick $r_1, r_2, \ldots, r_k$, then with a nontrivial probability there exists exactly one accepting computation v of M whose inner product with each r_i is 0. Thus, for a random choice of $r_1, \ldots, r_k$, there is exactly one witness v for $\langle 0^n, k, r_1, \ldots, r_k, i\rangle$. The rest of the proof is similar to that of Theorem 3.1.

We also note that we can replace the UP-machine in Hypothesis H with a FewP-machine.

REFERENCES

[AR88] E.W. Allender and R.S. Rubinstein, P-*printable sets*, SIAM J. Comput., 17 (1988), pp. 1193–1202.

[ASB00] K. Ambos-Spies and L. Bentzien, *Separating* NP-*completeness under strong hypotheses*, J. Comput. System Sci., 61 (2000), pp. 335–361.

[ASFH87] K. Ambos-Spies, H. Fleischhack, and H. Huwig, *Diagonalizations over polynomial time computable sets*, Theoret. Comput. Sci., 51 (1987), pp. 177–204.

[ASNT96] K. Ambos-Spies, H. Neis, and A. Terwijn, *Genericity and measure for exponential time*, Theoret. Comput. Sci., 168 (1996), pp. 3–19.

[BGS75] T. Baker, J. Gill, and R. Solovay, *Relativizations of the* $\mathcal{P} =? \mathcal{NP}$ *question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[BBS86] J. Balcázar, R. Book, and U. Schöning, *The polynomial-time hierarchy and sparse oracles*, J. ACM, 33 (1986), pp. 603–617.

[BS85] J. Balcázar and U. Schöning, *Bi-immune sets for complexity classes*, Math. Systems Theory, 18 (1985), pp. 1–10.

[FFNR96] S. Fenner, L. Fortnow, A. Naik, and J. Rogers, *On inverting onto functions*, in Proceedings of the 11th annual IEEE Conference on Computational Complexity, 1996, pp. 213–223.

[FPS99] L. Fortnow, A. Pavan, and A. Selman, *Distributionally hard languages*, Theory Comput. Syst., 34 (2001), pp. 245–262.

[FR94] L Fortnow and J. Rogers, *Separability and one-way functions*, in Proceedings of the Fifth International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 834, D.Z. Du and X.S. Zhang, eds., Springer-Verlag, Berlin, 1994, pp. 396–404.

[GHS91] J. Geske, D. Huynh, and J. Seiferas, *A note on almost-everywhere-complex sets and separating deterministic-time-complexity classes*, Inform. Comput., 92 (1991), pp. 97–104.

[Gor93] D. Gordon, *Discrete logarithms in GF(p) using the number field sieve*, SIAM Journal on Discrete Mathematics, 6 (1993), pp. 124–138.

[GS88] J. Grollmann and A.L. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.

[HKR93] S. Homer, S. Kurtz, and J. Royer, *On 1-truth-table-hard languages*, Theoret. Comput. Sci., 115 (1993), pp. 383–389.

[HL94] S. Homer and L. Longpré, *On reductions of* NP *sets to sparse sets*, J. Comput. System Sci., 48 (1994), pp. 324–336.

[HNOS96] E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman, P*-selective sets and reducing search to decision vs. self-reducibility*, J. Comput. System Sci., 53 (1996), pp. 194–209.

[HRW97] L. Hemaspaandra, J. Rothe, and G. Wechsung, *Easy sets and hard certificate schemes*, Acta Inform., 34 (1997), pp. 859–879.

[HS65] J. Hartmanis and R. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

[HY84] J. Hartmanis and Y. Yesha, *Computation times of* NP *sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[KM81] K. Ko and D. Moore, *Completeness, approximation and density*, SIAM J. Comput., 10 (1981), pp. 787–796.

[LLS75] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[LM96] J. Lutz and E. Mayordomo, *Cook versus Karp-Levin: Separating completeness notions if* NP *is not small*, Theoret. Comput. Sci., 164 (1996), pp. 141–163.

[LV97] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Grad. Texts Comput. Sci., Springer-Verlag, New York, 1997.

[LY90] L. Longpré and P. Young, *Cook reducibility is faster than Karp reducibility*, J. Comput. System Sci., 41 (1990), pp. 389–401.

[OW91] M. Ogiwara and O. Watanabe, *On polynomial-time bounded truth-table reducibility of* NP *sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.

[RW01] K. Regan and O. Watanabe, *private communication*.

[Sel79] A. Selman, P*-selective sets, tally languages, and the behavior of polynomial time reducibilities on* NP, Mathematical Systems Theory, 13 (1979), pp. 55–65.

[Sel82] A. Selman, *Reductions on* NP *and* P*-selective sets*, Theoret. Comput. Sci., 19 (1982), pp. 287–304.

[Tod91] S. Toda, *On polynomial-time truth-table reducibilities of intractable sets to* P*-selective sets*, Math. Systems Theory, 24 (1991), pp. 69–82.

[VV86] L. Valiant and V. Vazirani, NP *is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.

[Wat87] O. Watanabe, *A comparison of polynomial time completeness notions*, Theoret. Comput. Sci., 54 (1987), pp. 249–265.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 919–946

APPROXIMATION ALGORITHMS FOR SINGLE-SOURCE UNSPLITTABLE FLOW*

STAVROS G. KOLLIOPOULOS† AND CLIFFORD STEIN‡

Abstract. In the *single-source unsplittable flow* problem, we are given a network G, a source vertex s, and k commodities with sinks t_i and real-valued demands ρ_i, $1 \leq i \leq k$. We seek to route the demand ρ_i of each commodity i along a single s-t_i flow path so that the total flow routed across any edge e is bounded by the edge capacity u_e. The conceptual difficulty of this NP-hard problem arises from combining packing constraints due to the existence of capacities with path selection in a graph of arbitrary topology. In this paper we give a generic framework, which yields approximation algorithms that are simpler than those previously known and achieve significant improvements upon the approximation ratios. Our framework, with appropriate subroutines, applies to all optimization versions previously considered and, unlike previous work, treats in a unified manner directed and undirected graphs. We provide extensions of our algorithms which yield the best possible approximation guarantees for restricted sets of demand values and an associated scheduling problem.

Key words. approximation algorithms, network algorithms, routing, network flow, unsplittable flow, disjoint paths, parallel machine scheduling

AMS subject classifications. 68Q25, 90B10, 90B12, 90B35

PII. S0097539799355314

1. Introduction.

In the *single-source unsplittable flow* problem (UFP), we are given a network $G = (V, E, u)$, a source vertex s, and a set of k commodities with sinks $t_1, \ldots, t_k$ and associated real-valued demands $\rho_1, \ldots, \rho_k$. We seek to route the demand ρ_i of each commodity i along a single s-t_i flow path so that the total flow routed across any edge e is bounded by the edge capacity u_e. See Figure 1 for an example instance. The minimum edge capacity is assumed to have value at least $\max_i \rho_i$. The requirement of routing each commodity on a single path bears resemblance to integral multicommodity flow and in particular to the multiple-source edge-disjoint path problem. For the latter problem, a large amount of work exists either for solving exactly interesting special cases (e.g., [9, 37, 36]) or for approximating, with limited success, various objective functions (e.g., [35, 11, 24, 25, 22]). Further, shedding light on single-source unsplittable flow may lead to a better understanding of multiple-source disjoint-path problems. From a different perspective, UFP can be approached as a variant of standard, single-commodity, maximum flow since in both problems there is one source for the flow. From this point of view, UFP generalizes single-source edge-disjoint paths. UFP is also important as a unifying framework for a variety of scheduling and load balancing problems [23]. It can be used to model bin packing

*Received by the editors April 28, 1999; accepted for publication (in revised form) June 5, 2001; published electronically February 8, 2002. A preliminary version of this paper appeared in the *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science,* 1997, pp. 426–435.
http://www.siam.org/journals/sicomp/31-3/35531.html

†Department of Computing and Software, Faculty of Engineering, McMaster University, Hamilton, ON, L8S 4K1, Canada (stavros@mcmaster.ca). This author's research was partially supported by NSERC grant 227809-00 and a CFI New Opportunities Award. Part of this work was performed while at the Department of Computer Science, Dartmouth College, and partially supported by NSF award CCR-9308701 and NSF Career Award CCR-9624828.

‡Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027 (cliff@ieor.columbia.edu). This author's research was performed while at the Department of Computer Science, Dartmouth College, and partially supported by NSF award CCR-9308701, NSF grant EIA-98-02068, and NSF Career Award CCR-9624828.

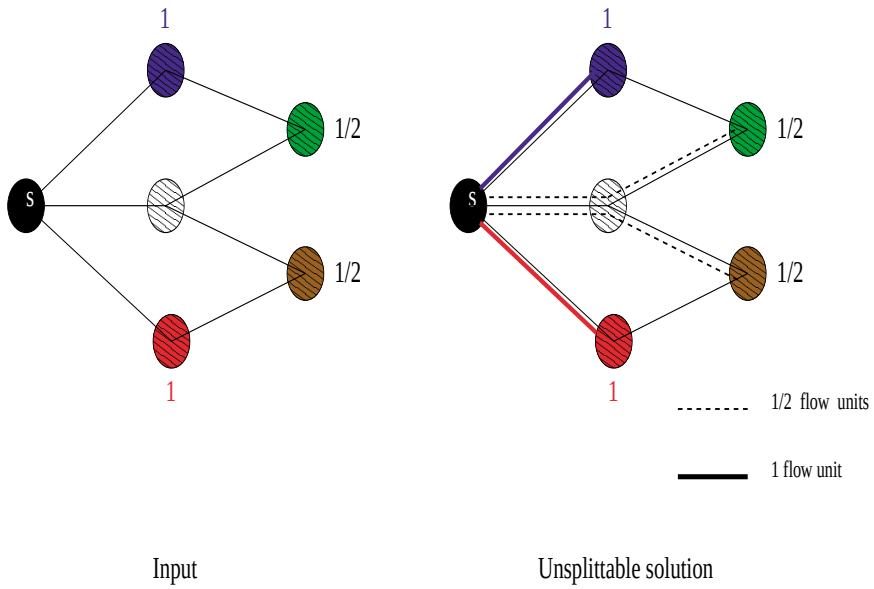


FIG. 1. *Example* UFP *instance. Numbers on the vertices denote demands; all edge capacities are equal to* 1.

[23] and certain scheduling problems on parallel machines [32]. Another possible application for UFP is in virtual-circuit routing, where the source would represent a node wishing to multicast data to selected sinks. The conceptual difficulty in UFP arises from combining packing constraints due to the existence of capacities with path selection in a graph of arbitrary topology.

The feasibility question for UFP, i.e., can all commodities be routed unsplittably, is strongly NP-complete [22]. We focus thus on efficient algorithms to obtain approximate solutions. For a minimization (resp., maximization) problem, a *ρ-approximation algorithm,* $\rho > 1$ (resp., $\rho < 1$), outputs in polynomial time a solution with objective value at most (resp., at least) ρ times the optimum. Three main optimization versions of unsplittable flow can be defined.

Minimum congestion. Find an unsplittable flow f satisfying all demands, which minimizes *congestion,* i.e., the quantity $\max\{\max_e\{f_e/u_e\}, 1\}$. That is, the minimum congestion is the smallest number $\alpha \geq 1$ such that if we multiplied all capacities by α, f would satisfy the capacity constraints. The congestion metric was a primary test bed for the randomized rounding technique of Raghavan and Thompson [35] and has been studied extensively for its connections to multicommodity flow and cuts (e.g., [31, 21, 30, 20]).[1]

Maximum routable demand. Route unsplittably a subset of the commodities of maximum total demand, while respecting capacity constraints.

Minimum number of rounds. Partition the commodities into a minimum number of sets, called *rounds,* so that commodities assigned to the same round can be routed

[1] In [30] congestion is defined as $\max_e\{f_e/u_e\}$. In this setting a congestion of $\lambda \in (0, 1)$ implies that it is possible to multiply all demands by $1/\lambda$ and still satisfy them by a flow which respects the capacity constraints. Algorithmically the two definitions are equivalent: with a polynomial number of invocations of an algorithm which minimizes congestion according to our definition, one can minimize congestion according to the definition in [30].

unsplittably without violating the capacity constraints.

The randomized rounding technique applies to the more general multiple-source unsplittable flow problem and provides an $O(\log|E|/\log\log|E|)$ approximation for congestion; for the maximum demand and minimum number of rounds versions, it can give meaningful approximations only when the minimum capacity value is $\Omega(\log|E|)$ times the value of the maximum demand [35, 34]. Kleinberg [22, 23] was the first to consider the UFP as a distinct problem and gave constant-factor approximation algorithms for all three optimization versions presented above, on both directed and undirected graphs.

Our results. We give a new approach for single-source unsplittable flow that has three main advantages.

- The algorithms are simple and do not need the machinery developed in [23].
- Our approach treats directed and undirected graphs in a unified manner.
- We obtain significant improvements upon the approximation ratios for all three optimization versions.

Our algorithms follow the same *grouping-and-scaling* technique. We first find a maximum flow. This relaxation is a "splittable" solution, i.e., it allows the demand for a commodity to be split along more than one path. As in the previous work [23], we use the maximum flow solution as a guide for the discovery of an unsplittable routing. However, we use this solution in a much stronger way: we employ information from maximum flow to *allocate capacity* to a set of subproblems, where each subproblem G^i contains commodities with demands in a fixed range. For each subproblem G^i, a near-optimal routing g^i is separately computed by exploiting the fact that demands are close in range. These solutions are then combined by superimposing the g^i's on the original graph. Since we are interested in constant-factor approximations, when combining the solutions it is important to avoid an additive deterioration of the approximation guarantees obtained for each individual G^i. One of our conceptual contributions lies in showing that this is possible, given an appropriate initial capacity allocation. Combining solutions presents different challenges for the maximum demand and minimum number of rounds versions of our algorithms as opposed to minimizing congestion. In the congestion version, increasing the original edge capacities as the routings g^i are superimposed on G only affects the approximation ratio. For the other two versions, it would be infeasible to do so. We note that a grouping scheme on the commodities is also used in [23], though in a more complicated way; ours is based solely on demand values and does not require any topological information from the graph. We now elaborate on the approximation ratios we obtain.

Minimum congestion. We give a 3-approximation algorithm for both directed and undirected graphs. The bounds known before our work were 16 for the directed and 8.25 for the undirected case [23]. In the first publication of this work [27] we gave PARTITION, a 4-approximation algorithm and obtained also a $10/3 = 3.33\ldots$ bound[2] for a more elaborate version of PARTITION. Dinitz, Garg, and Goemans recently obtained a new 2-approximation result (appearing in [5]). Following this development we further refined our 10/3-approximation algorithm and obtained H_PARTITION, a 3-approximation algorithm appearing also in [26]. In this paper we describe both PARTITION and H_PARTITION. Algorithm H_PARTITION also exploits the grouping and scaling technique by scaling demands in a fixed range to the same value. However, instead of producing "in parallel" an unsplittable solution for all the ranges, H_PARTITION outputs the partial routings g^i by iteratively improving the quality

[2]Erroneously reported in [27] as 3.23.

of the solution to a maximum flow relaxation. The quality of the latter solution is characterized by the minimum amount of flow routed along a single path to any commodity which has not been routed yet unsplittably.

Minimizing congestion and cost. A natural optimization question arises when routing a unit of flow through edge e incurs cost c_e. The objective is to find an unsplittable flow that minimizes the total cost. More precisely, if z is the optimal congestion, define the optimal cost to be the minimum cost of an unsplittable flow that is feasible with respect to the capacity function zu. Since UFP without costs is already NP-hard, we aim for a bicriteria approximation of cost and congestion. Our approach gives the first constant-factor approximation for minimum-cost, minimum-congestion unsplittable flow on directed graphs; it also improves considerably upon the constants known for the minimum cost version on undirected graphs. In particular, existing results for undirected graphs [23] give a simultaneous $(7.473, 10.473)$ approximation for cost and congestion. We provide a $(2, 3)$ simultaneous approximation on both directed and undirected graphs. Moreover, we show how to obtain a $1 + \delta$ approximation for cost, for any $\delta > 0$, at the expense of a larger constant factor for congestion.

Maximum routable demand. We show how to route at least .075, i.e., 7.5% of the optimum. In [23] the constant is not given explicitly, but it can be as low as .031 for undirected graphs and on the order of 10^{-9} for directed graphs.

Minimum number of rounds. We show how to route all the demands in at most 13 times the optimum number of rounds, an improvement upon the 32 upper bound given in [23]. A guarantee of routing in x rounds implies that at least $1/x$ of the total demand is routed in some round. Kleinberg's 32-approximation algorithm for rounds can be seen to imply a $1/32 = 0.03125$ approximation for routable demand when the cut condition is met, i.e., when the maximum flow relaxation satisfies all demands. Accordingly we obtain a $1/13 = .0769$ approximation for routable demand under the cut condition. We elaborate on this observation in section 8.

We emphasize that our algorithms for all three different versions are simple and make use of the same generic framework. Although they are presented separately for ease of exposition, they could all be stated in terms of one algorithm, with different subroutines invoked for subproblems. The dominant computational steps are maximum flow and flow decomposition; these are tools that work well on both directed and undirected graphs. In [23] it is noted that "the disjoint paths problem is much less well understood for directed than it is for undirected graphs ... in keeping with the general principle we will find that the algorithms we obtain for directed graphs will be more complicated."

Generalizations. The algorithmic techniques we introduce are quite general and can be applied to related problems. We show how to use them to obtain a constant-factor approximation for minimum congestion on a network, where the minimum edge capacity can be arbitrarily low with respect to the maximum demand. This is the first result of this type that we are aware of. In subsequent work [28], we show how to apply the algorithmic techniques developed here for UFP to obtain improved approximations for classes of packing integer programs and multiple-source unsplittable flow. On the negative side, we show in this paper that for the unsplittable flow problem with two sources on a directed network, no ρ-approximation with $\rho < 2$ can be achieved for congestion, unless P = NP, even when all the demands and capacities have value 1. This gives an idea of how hard it might be to formulate an approximate version of the fundamental Theorem 2.1 that we use in our single-source algorithms.

Applications to scheduling. We also examine applications of unsplittable flow to scheduling problems. A lower bound of 3/2 exists for the approximability of mini-

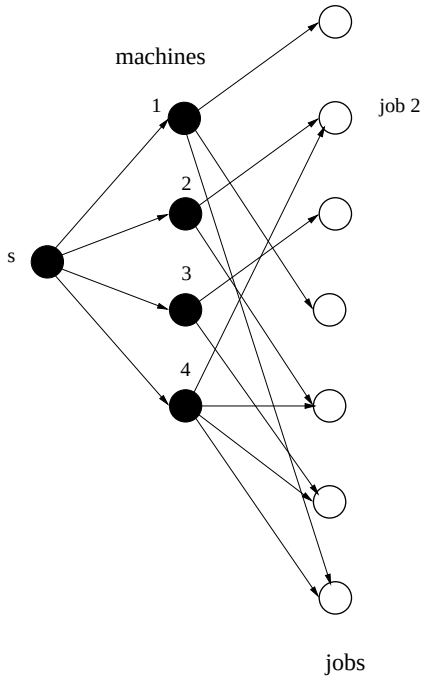


FIG. 2. *Reduction of the scheduling problem to* UFP. *Job* 2 *can be processed only on machines* 2 *and* 4. *The capacity of the edges connecting* s *to the machine vertices is equal to a makespan estimate.*

mum makespan on parallel machines when a job j can be processed only on a subset $M(j)$ of the available machines [32]. Let $\mathcal{S}$ denote this scheduling problem. The 2-approximation algorithm of Lenstra, Shmoys, and Tardos [32] or Shmoys and Tardos [38] for the more general problem of scheduling on unrelated machines is the best known for $\mathcal{S}$. On the other hand an approximation-preserving reduction is known to exist from $\mathcal{S}$ to single-source unsplittable flow (see Figure 2) [23], so the 3/2 lower bound for $\mathcal{S}$ [32] applies to single-source unsplittable flow as well. The lower bound holds when processing times (or demands) take the values p or $2p$ for some $p > 0$. We provide an additive p approximation for this restricted unsplittable flow problem, which corresponds to a ratio of at most 3/2. This is the best possible approximation ratio for the problem unless P = NP. Improved approximation factors are also obtained for the more general case, in which all demand values are powers of 1/2.

The outline of this paper is as follows. In section 2 we present definitions and a basic theorem from flow theory. In section 3 we provide a $(4 + o(1))$-approximation algorithm for congestion, which introduces our basic techniques. In section 4 we provide the improved 3-approximation. In section 5 we give the approximation algorithm for minimum congestion unsplittable flow with arbitrarily small capacities. In section 6 we present the simultaneous cost-congestion approximation. In sections 7 and 8 the approximation algorithms for maximum total demand and minimum number of rounds are presented. In section 9 we present the hardness result for 2-source unsplittable flow. Finally, in section 10 we examine the connection between UFP and scheduling problems. A preliminary version of the material in this paper appeared in [27]. Experimental results on implementations of our algorithms appeared in [29]. Af-

ter the first publication of this work, Dinitz, Garg, and Goemans obtained improved approximation ratios by using a different approach [5], and Skutella gave improved results for the simultaneous optimization of cost and each of the three metrics [39].

2. Preliminaries. All logarithms in this paper are base 2 unless otherwise noted. A *single-source unsplittable flow problem* (UFP) $(G, s, \mathcal{T})$ is defined on a capacitated network $G = (V, E, u)$ where a subset $\mathcal{T} = \{t_1, \ldots, t_k\}$ of V is the set of *sinks*; each edge e has a real-valued capacity u_e; each sink t_i has a real-valued demand ρ_i. The pair (t_i, ρ_i) defines the *commodity* i. To avoid cumbersome notation, we occasionally use the set $\mathcal{T}$ to denote the set of commodities. We assume in our definition that each vertex in V hosts at most one sink; the input network can always be transformed to fulfill this requirement. A *source* vertex $s \in V$ is specified. For any $I \subseteq \mathcal{T}$, a *partial routing* is a set of $|I|$ s-t_i paths, in which path P_i is used to route ρ_i amount of flow from s to t_i for each $t_i \in I$. A partial routing that contains a path to each sink in $\mathcal{T}$ is simply called a *routing.* Given a routing g, the flow g_e through edge e is equal to $\sum_{P_i \in g | P_i \ni e} \rho_i$. A (partial) routing g for which $g_e \leq u_e$ for each edge e is a *(partial) unsplittable flow.* A *fractional routing* is a set of flow paths, satisfying capacity constraints, in which the flow to each sink is split along potentially many paths. A fractional routing can be computed by standard maximum flow. Thus we shall refer to a fractional routing satisfying all demands of the underlying UFP as a *feasible fractional flow solution.* For clarity we will also refer to a standard maximum flow as *fractional maximum flow.* We assume that a UFP input meets the following two requirements:

Cut condition. There is a feasible fractional flow solution.

Balance condition. All demands have value at most 1 and the minimum edge capacity is 1.

The first requirement is introduced only for simplicity. For each of the optimization metrics studied, an optimal fractional solution, potentially violating the unsplittability requirement, can be efficiently found. Since our analysis uses the fractional optima as lower bounds, our approximation results hold without the cut condition. For the balance condition, we note that the actual numerical value of 1 is not crucial; demands and capacities can be scaled without affecting solutions. The important requirement is that the minimum capacity is greater than or equal to the maximum demand. We will explicitly state it when we deal with a UFP in which the minimum capacity can be arbitrarily small. Finally, the *k-source unsplittable flow problem* (k-UFP) is similarly defined, but now $k > 1$ sources exist and demands are defined for pairs (s_i, t_i) of sources and sinks. Unless otherwise stated, we use n, m to denote $|V|, |E|$, respectively.

The following theorem is an easy consequence of the well-known augmenting path algorithm [7, 6] and will be of use. It was also used as part of the approximation techniques in [23] and is a consequence of the integrality property for maximum flow with all demands and capacities scaled by σ.

THEOREM 2.1. *Let $(G = (V, E, u), s, \mathcal{T})$ be a* UFP *on an arbitrary network G with all demands ρ_i, $1 \leq i \leq k$, equal to the same value σ, and edge capacities u_e equal to integral multiples of σ for all $e \in E$. If there is a fractional flow of value f, then there is an unsplittable flow of value at least f. This unsplittable flow can be found in $O(km)$ time, where k is the number of sinks in $\mathcal{T}$.*

We defined a routing as a collection of flow paths; it can equivalently be represented as an *edge flow,* i.e., a collection of functions assigning flow values to each edge. Conversely, an edge flow, whether fractional or unsplittable, can be represented

as a path flow. In particular our algorithms will make use of the well-known *flow decomposition theorem* [2]. Given problem $(G, s, \mathcal{T})$ let a fractional flow solution f be represented as an assignment of flow values to edges. Then one can find, in $O(nm)$ time, a representation of f as a path and cycle flow, where the paths join s to sinks in $\mathcal{T}$. In the remainder of the paper, we will assume that the cycles are deleted from the output of a flow decomposition. The flow decomposition theorem trivially applies also to transforming an unsplittable flow from edge representation to a path representation. We will point out cases in which outputting the unsplittable flow as an edge flow results in faster algorithms.

3. The approximation algorithm for minimum congestion. In this section we give a $(4 + o(1))$-approximation algorithm for minimizing congestion. In section 4 we show how to improve the approximation guarantee to 3. All our results can be seen to hold for minimizing *absolute congestion* as well, defined as $\max\{\max_e\{f_e\}, 1\}$. The same algorithm works on both directed and undirected graphs with the same performance guarantee. The algorithm works by partitioning the original UFP into subproblems, each containing only demands in a fixed range. We use an initial maximum flow solution to allocate capacity to different subproblems. Capacities in the subproblems are arbitrary and may violate the balance condition. The subproblems are separately solved by introducing bounded congestion; the partial solutions are then combined by superimposing on the original network G the flow paths obtained in the respective routings. The combination step will have a subadditive effect on the total congestion and thus near-optimality of the global solution is achieved.

We present first two simple lemmas for cases in which demands are bounded. They will be used to provide subroutines to the final algorithm.

LEMMA 3.1. *For a* UFP *with demands in the interval* $(0, \alpha]$, *there is an algorithm,* α-ROUTING, *which finds, in time* $O(n + m)$, *an unsplittable routing in which the flow through each edge is at most* $n\alpha$.

Proof. Algorithm α-ROUTING works as follows. A depth first search from the source s gives a path from s to each of the at most n sinks. For each sink t_i, route, along the corresponding path, flow equal to the demand $\rho_i \leq \alpha$. □

Given a real number interval with endpoints a and b, $0 < a < b$, the *ratio* $r(a, b)$ of the interval is b/a. The following fact is also used in [23]; we include a proof for the sake of completeness.

LEMMA 3.2 (see [23]). *Given a* UFP $\Pi = (G, s, \mathcal{T})$, *with demands in the interval* $(a, b]$ *and arbitrary capacities, there is an algorithm,* INTERVAL_ROUTING, *which finds in polynomial time an unsplittable routing* g, *such that for all edges* e, $g_e \leq r(a, b)u_e + b$.

Remark. The statement of the lemma holds also when demands lie in the interval $[a, b]$.

Proof. Let f be a feasible fractional solution to Π. Round all demands up to b and call Π' the resulting UFP instance. To obtain a feasible fractional solution for Π', it suffices to multiply the flow f_e and the capacity u_e on each edge e by a factor of at most $r(a, b)$. Further round all capacities up to the closest multiple of b by adding at most b. The new edge capacities are now at most $r(a, b)u_e + b$. By Theorem 2.1 an unsplittable routing for Π' can now be found in polynomial time. Algorithm INTERVAL_ROUTING finds first this unsplittable routing g' for Π'. To obtain an unsplittable routing g for Π, from each path in g', the flow in excess of the demands in $\mathcal{T}$ is removed. □

We are now ready to present Algorithm PARTITION in Figure 3. Algorithm PARTITION takes as input a UFP $(G, s, \mathcal{T})$, together with a set of parameters $\xi, \alpha_1, \ldots, \alpha_\xi$,

ALGORITHM PARTITION$(G = (V, E, u), s, \mathcal{T}, \xi, \alpha_1, \ldots, \alpha_\xi)$

***Step* 1.** Find a feasible fractional solution f.

***Step* 2.** Define a partition of the $(0, 1]$ interval into ξ consecutive subintervals $(0, \alpha_1], (\alpha_1, \alpha_2], \ldots, (\alpha_{\xi-1}, \alpha_\xi]$, $\alpha_\xi = 1$. Construct ξ copies of G where the set of sinks in G_i, $1 \le i \le \xi$, is the subset $\mathcal{T}_i$ of $\mathcal{T}$ with demands in the interval $(\alpha_{i-1}, \alpha_i]$. Using flow decomposition determine for each edge e the amount u_e^i of u_e used by f to route flow to sinks in $\mathcal{T}_i$. Set the capacity of edge e in G_i to u_e^i.

***Step* 3.** Invoke INTERVAL_ROUTING on each G_i, $2 \le i \le \xi$, to obtain an unsplittable routing g^i. Invoke α-ROUTING on G_1 to obtain an unsplittable routing g^1.

***Step* 4.** Output routing g, the union of the path sets g^i, $1 \le i \le \xi$.

FIG. 3. *Algorithm* PARTITION.

that will determine the exact partitioning scheme. Subsequently we show how to choose these parameters so as to optimize the approximation ratio. The algorithm outputs an unsplittable routing g. The two previous lemmas will be used to provide subroutines. Recall that in a UFP demands lie in the interval $(0, 1]$ and capacities are at least 1.

We need to examine the effect of the combination on the total congestion.

LEMMA 3.3. *Algorithm* PARTITION *outputs an unsplittable routing g with congestion at most*

$$n\alpha_1 + \max_{2 \le i \le \xi} \{r(\alpha_{i-1}, \alpha_i)\} + \sum_{i=2}^{\xi} \alpha_i.$$

The running time of PARTITION *is $O(T_1(n, m) + nm + m\xi)$ where $T_1(n, m)$ is the time to solve a fractional maximum flow problem.*

Proof. In order to determine in Step 2 the capacity u_e^i of edge e in the ith copy G_i, $1 \le i \le \xi$, of G we use the flow decomposition theorem [8]. Any flow along cycles given by the decomposition theorem is discarded since it does not contribute to the routing of demands. We focus on the congestion of a particular edge e. By Lemma 3.2 algorithm INTERVAL_ROUTING finds an unsplittable routing in G_i, $2 \le i \le \xi$, at Step 3 by pushing through edge e at most $r(\alpha_{i-1}, \alpha_i)u_e^i + \alpha_i$ units of flow. By Lemma 3.1 algorithm α-ROUTING pushes at most $n\alpha_1$ units of flow through edge e to find a routing on G_1. At Step 4 the superposition of the routings g^i gives an approximate unsplittable solution g, in which, for each edge e,

$$\begin{aligned} g_e &\le n\alpha_1 + \sum_{i=2}^{\xi} u_e^i r(\alpha_{i-1}, \alpha_i) + \sum_{i=2}^{\xi} \alpha_i \\ &\le n\alpha_1 + \max_{2 \le i \le \xi} \{r(\alpha_{i-1}, \alpha_i)\} \sum_{i=2}^{\xi} u_e^i + \sum_{i=2}^{\xi} \alpha_i \\ &\le n\alpha_1 u_e + \max_{2 \le i \le \xi} \{r(\alpha_{i-1}, \alpha_i)\} u_e + \sum_{i=2}^{\xi} \alpha_i u_e. \end{aligned}$$

The last step follows because $\sum_{i=2}^{\xi} u_e^i \leq u_e$ from the feasibility of f, and $u_e \geq 1$, and thus the congestion bound in the lemma follows.

As for the running time, finding the fractional flow in Step 1 requires $T_1(n,m)$ time. Flow decomposition can be done in $O(nm)$ time [2], thus the running time of Step 2 is $O(nm + m\xi)$. Implementing INTERVAL_ROUTING with the augmenting path algorithm gives total running time $O(km)$ for all the $\xi - 1$ invocations at Step 3. This time includes the time required by flow decomposition to output each partial routing in a path representation. The reason is that by Theorem 2.1 the algorithm takes $O(m)$ time per sink. The total number of sinks in the ξ subproblems is $k \leq n$. Hence the total running time of PARTITION is $O(T_1(n,m) + nm + m\xi)$. □

In order to substantiate the approximation factor we need to come up with a partitioning scheme to be used at Step 2 of algorithm PARTITION. The following theorem is the main result of this section.

THEOREM 3.1. *Given a* UFP Π*, algorithm* PARTITION *finds a* $(4+\varepsilon)$*-approximation for relative congestion for any* $\varepsilon > 0$*. The running time of the algorithm is* $O(T_1(n,m) + nm + m\log(n/\varepsilon))$*, where* $T_1(n,m)$ *is the time to solve a fractional maximum flow problem.*

Proof. At Step 2 of PARTITION, partition the interval $(0,1]$ of demand values into ξ geometrically increasing subintervals

$$(0, 1/2^{\xi-1}], \ldots, (1/2^{i+1}, 1/2^i], \ldots, (1/2^2, 1/2], (1/2, 1]$$

such that $1/2^{\xi-1} \leq \varepsilon/n$. Thus it suffices for ξ to be $\Theta(\log(n/\varepsilon))$. By Lemma 3.3, the congestion of the routing g returned at Step 4 of the algorithm is at most $n\frac{\varepsilon}{n} + 2 + \sum_{i=0}^{\xi-2} \frac{1}{2^i} < 4 + \varepsilon$. □

In order to achieve $\varepsilon = o(1)$, it suffices to set $1/2^{\xi-1} \leq 1/n^c$, $c > 1$. Obtaining a better $o(1)$ factor is straightforward by increasing the number of intervals. A fractional maximum flow can be found by the push-relabel method of Goldberg and Tarjan [13], whose currently fastest implementation has running time $T_1(n,m) = O(nm \log_{\frac{m}{n\log n}} n)$ [19]. In that case even when our algorithm is used to obtain a $(4 + \frac{1}{2^n})$-approximation, the running time is dominated by a single maximum flow computation. Alternatively, the new maximum flow algorithm of Goldberg and Rao [12] with $T_1(n,m) = O(\min(n^{2/3}, m^{1/2}) m \log(\frac{n^2}{m}) \log U)$ may be used, if edge capacities can be expressed as integers in a range $[1, \ldots, U]$. We point out that if an edge representation of the output routing suffices, the Goldberg–Rao algorithm can be used to surpass the $O(nm)$ bottleneck.

THEOREM 3.2. *If an edge-representation of the output routing is sought, algorithm* PARTITION *can be implemented to run in* $O(\log(n/\varepsilon) \min(n^{2/3}, m^{1/2}) m \log(\frac{n^2}{m}) \log(UD))$ *time, assuming in the original network we have integral capacities in the range* $[1, \ldots, U]$ *and the minimum demand is* $1/D$ *for* $D \in Z_{>0}$*.*

Proof. The flow decomposition computation in Step 2 of PARTITION can be replaced by $\xi = O(\log(n/\varepsilon))$ maximum flow computations; in the ith such computation only the set of sinks $\mathcal{T}_i$, as defined in Step 3, with demand values in the corresponding subinterval is included.

The first maximum flow computation has available on each edge the capacity used by the original fractional solution f. The flow found at the ith computation is subtracted from the capacity allotted to the $(i+1)$st computation. Intuitively, this iterative procedure is correct for the same reason that standard flow decomposition is correct. In the latter case, flow is subtracted along a single path (or cycle) at each

iteration. In each iteration of our scheme, we use Goldberg–Rao to find in one "shot" all the flow paths leading to sinks with demand values in a given subinterval and remove their capacity. It is enough to show that implementing flow decomposition by successive invocations of Goldberg–Rao maintains the following invariant.

INV: After the ith Goldberg–Rao flow computation, the available capacity across any cut (S, T) with $s \in S$ is equal to the sum of the demand values of the sinks in $T \cap (\mathcal{T} - \cup_{j \le i} \mathcal{T}_j)$.

The induction hypothesis is that INV is true after the end of the $(i-1)$st iteration. Across a given (S, T) cut the available capacity is equal to the sum of the demand values of the sinks in $T \cap (\mathcal{T} - \cup_{j < i} \mathcal{T}_j)$. Goldberg–Rao will make unusable during the ith invocation an amount of cut capacity equal to the sum of the demand values of the sinks in $T \cap \mathcal{T}_i$. Hence INV will be true after the ith iteration as well.

Also the INTERVAL_ROUTING subroutine at Step 3 can be implemented by the Goldberg–Rao algorithm yielding an edge-only representation of the partial routing. Finally, the demand value of a sink can be represented as the capacity of the unique edge leading to it. Multiplying demands and capacities by D results in a network with integral capacities in the range $[1, \ldots, UD]$. □

Although in the next section we improve upon the 4-approximation, algorithm PARTITION introduces some of the basic ideas behind all of our algorithms. For this reason a demonstration of the algorithm execution on an example instance follows in Figures 4–7. In these figures, numbers on vertices denote demands while numbers on edges denote capacities. The capacities on the original input are equal to 1. Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry 1 and 1/2 units of flow, respectively.

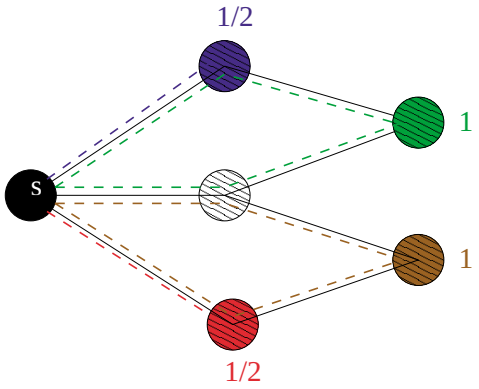
4. A 3-approximation algorithm for congestion. An improved approximation algorithm can be obtained by combining the partitioning idea above with a more careful treatment of the subproblems. At the first publication of our results [27] we obtained a $(10/3 + o(1))$-approximation. Subsequently we improved our scheme to obtain a 3 ratio; prior to this improvement, a 2-approximation for UFP was independently obtained by Dinitz, Garg, and Goemans [5]; see also [26].

We give first a specialized version of Lemma 3.2 to be used in the new algorithm.

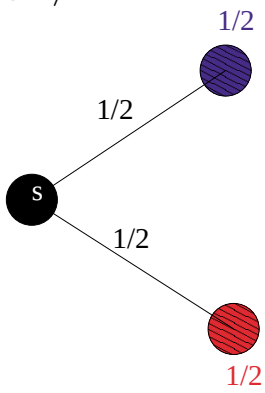
LEMMA 4.1. *Let $\Pi = (G, s, \mathcal{T})$ be a* UFP *in which all demands have value $1/2^x$ for $x \in N$, and all capacities are multiples of $1/2^{x+1}$, and let f be a fractional flow solution such that the flow f_e through any edge is a multiple of $1/2^{x+1}$. We can find in polynomial time an unsplittable routing g where the flow g_e through an edge e is at most $f_e + 1/2^{x+1}$.*

Algorithm PARTITION finds a routing for each subproblem by scaling up subproblem capacities to ensure they conform to the requirements of Lemma 3.2. The new algorithm operates in phases, during which the number of distinct paths used to route flow for a particular commodity is progressively decreased. Call *granularity* the minimum amount of flow routed along a single path to any commodity which has not been routed yet unsplittably. Algorithm PARTITION starts with a fractional solution of arbitrary granularity and, in essentially one step per commodity, rounds this fractional solution to an unsplittable one. In the new algorithm, we delay the rounding process in the sense that we keep computing a fractional solution of successively increasing granularity. The granularity will always be a power of 1/2, and this unit will be doubled after each iteration. Once the granularity surpasses $1/2^x$, for some x, we guarantee that all commodities with demands $1/2^x$ or less have already been routed unsplittably. Each iteration improves the granularity at the expense of increasing the

Fractional Solution:



Subproblem G^1 for demands of 1/2:



Subproblem G^2 for demands of 1:

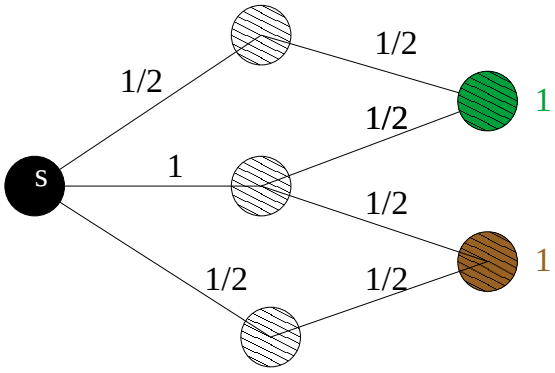


FIG. 4. *Algorithm* PARTITION *Demonstration* I. *Numbers on the edges and the vertices denote capacities and demands, respectively. Original input capacities are* 1. *Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry* 1 *and* 1/2 *units of flow, respectively.*

congestion. The method has the advantage that, by Lemma 4.1, a fractional solution of granularity $1/2^x$ requires only a $1/2^x$ additive offset to current capacities to find a new flow of granularity $1/2^{x-1}$. Therefore, if the algorithm starts with a fractional

Scale up capacities for G^1 to $2u_e^1 + (1/2)$. Unsplittable solution:

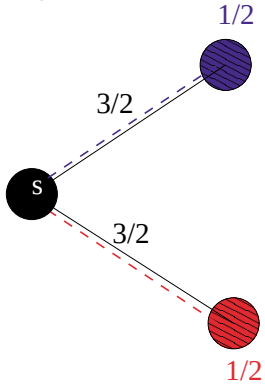


FIG. 5. *Algorithm* PARTITION *Demonstration* II. *Numbers on the edges and the vertices denote capacities and demands, respectively. Original input capacities are* 1. *Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry* 1 *and* 1/2 *units of flow, respectively.*

Scale up capacities for G^2 to $2u_e^2 + 1$. Unsplittable solution:

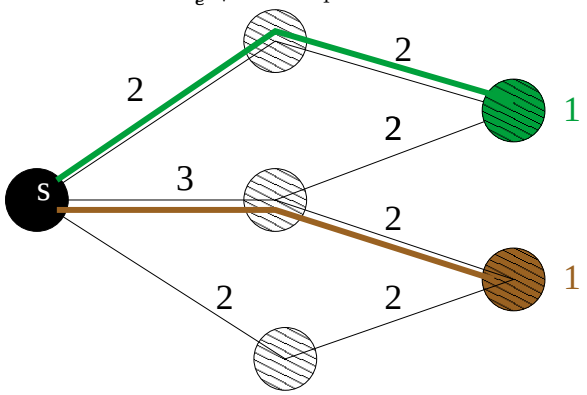


FIG. 6. *Algorithm* PARTITION *Demonstration* III. *Numbers on the edges and the vertices denote capacities and demands, respectively. Original input capacities are* 1. *Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry* 1 *and* 1/2 *units of flow, respectively.*

solution of granularity $1/2^\lambda$, for some potentially large λ, the total increase to an edge capacity from then on would be at most $\sum_{j=1}^{j=\lambda} 1/2^j < 1$. Expressing the granularity in powers of $1/2$ requires an initial rounding of the demand values; this rounding will force us to scale capacities by at most a factor of 2. We first provide an algorithm for the case in which demand values are powers of $1/2$. The algorithm for the general case will then follow easily.

THEOREM 4.1. *Let* $\Pi = (G = (V, E, u), s, \mathcal{T})$ *be a* UFP *in which all demand values are of the form* $2^{-\nu}$ *with* $\nu \in N$, *the maximum demand value is denoted by* $\rho_{\max}$, *and the minimum demand value is denoted by* $\rho_{\min}$. *There is an algorithm,* 2H_PARTITION, *which obtains in polynomial time an unsplittable routing such that the flow through any edge* e *is at most* $zu_e + \rho_{\max} - \rho_{\min}$ *where* z *is the optimum congestion.*

Superimpose the two solutions to obtain final solution:

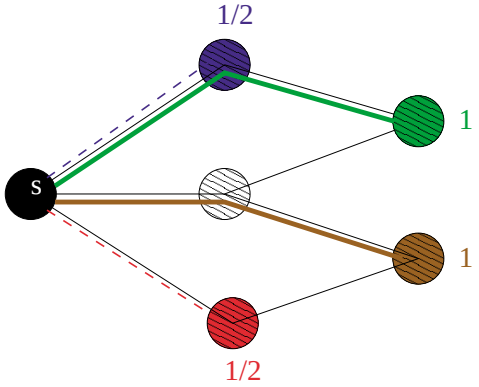


FIG. 7. *Algorithm* PARTITION *Demonstration* IV. *Numbers on the edges and the vertices denote capacities and demands, respectively. Original input capacities are* 1. *Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry* 1 *and* 1/2 *units of flow, respectively. Final congestion is* 3/2.

Remark. We first obtained a relative $(1+\rho_{\max}-\rho_{\min})$-approximation for the case with demand values in $\{1/2, 1\}$ in [27]. Dinitz, Garg, and Goemans [5] first extended that theorem to arbitrary powers of 1/2 to obtain a $u_e+\rho_{\max}-\rho_{\min}$ absolute guarantee on the flow through edge e. Their derivation uses a different approach.

Proof. We describe the algorithm 2H_PARTITION. The crux of the algorithm is a relaxed decision procedure that addresses the question, Is there an unsplittable flow after scaling all the input capacities by x? The procedure will either return no or will output a solution where the flow through edge e is at most $xu_e + \rho_{\max} - \rho_{\min}$. Embedding the relaxed decision procedure in a binary search completes the algorithm. See [15] for background on relaxed decision procedures. The following claim will be used to show the correctness of the procedure.

CLAIM 4.1. *Given a* UFP Π *with capacity function* r, *all demands of the form* $2^{-\nu}$, $\nu \in N$, *and minimum demand* $2^{-\lambda}$, *let* Π' *be the modified version of* Π *in which each edge capacity* r_e *has been rounded down to the closest multiple* r'_e *of* $1/2^{\lambda}$. *If the optimum congestion is* 1 *for* Π, *the optimum congestion is* 1 *for* Π' *as well.*

Proof of claim. Let G be the network with capacity function r and G' be the network with rounded capacities r'. The amount $r_e - r'_e$ of capacity will be unused by any optimal unsplittable flow in G. The reason is that the sum $\sum_{i\in S} 1/2^i$ for any finite multiset $S \subset N$ is equal to a multiple of $1/2^{i_0}$, $i_0 = \min_{i\in S}\{i\}$. □

We describe now the relaxed decision procedure. Let u' denote the scaled capacity function xu. Let $\rho_{\min} = 1/2^{\lambda}$. The relaxed decision procedure is broken into at most $\lambda - \log(\rho_{\max}^{-1}) + 1$ iterations; each iteration aims to double the granularity.

During the first iteration, split each commodity in $\mathcal{T}$, called an *original commodity*, with demand $1/2^y$, $0 \le y \le \lambda$, into $2^{\lambda-y}$ *virtual commodities* each with demand $1/2^{\lambda}$. Call the resulting set of commodities $\mathcal{T}_1$. Round down all capacities to the closest multiple of $1/2^{\lambda}$. By Theorem 2.1 we can find an unsplittable flow solution $f^0 \equiv g^1$, for $\mathcal{T}_1$, in which the flow through any edge is a multiple of $1/2^{\lambda}$. If this solution does not satisfy all demands, the decision procedures returns no; by Claim 4.1 no unsplittable solution is possible without increasing the capacities u'. Otherwise the decision procedure continues to iterate as described below. We denote by u_e^i, $i \ge 1$, the flow through edge e at the end of the ith iteration. Observe that f^0 is a fractional solution

for the set of original commodities as well, a solution with granularity $1/2^{\lambda}$. The flow u_e^1 through an edge is trivially $f_e^0 \leq u'_e \leq u_e$.

At iteration $i > 1$, the following steps are taken.

Step i1. Extract from g^{i-1}, using flow decomposition, the set of flow paths f^{i-1} used to route the set S_{i-1} of virtual commodities in $\mathcal{T}_{i-1}$ which correspond to original commodities in $\mathcal{T}$ with demand $1/2^{\lambda-i+1}$ or higher.

Step i2. Pair up the virtual commodities in S_{i-1} corresponding to the same original commodity. Call the resulting set of virtual commodities $\mathcal{T}_i$.

Step i3. Find an unsplittable routing g^i for $\mathcal{T}_i$.

At Step i2, pairing is possible since the demand of an original commodity is inductively an even multiple of the demands of the virtual commodities. $\mathcal{T}_i$ will contain virtual commodities with demand $1/2^{\lambda-i+1}$. For Step i3 observe that the set of flow paths f^{i-1} is a fractional solution for the commodity set $\mathcal{T}_i$. Inductively each flow path in f^{i-1} carries flow which is a multiple of $1/2^{\lambda-i+2}$. By Lemma 4.1, we can find an unsplittable routing g^i for $\mathcal{T}_i$ such that the flow g_e^i through an edge e is at most

$$f_e^{i-1} + 1/2^{\lambda-i+2}.$$

After the ith iteration the granularity has been doubled to $1/2^{\lambda-i+1}$. A crucial observation is that from this quantity, g_e^i, only an amount of at most $1/2^{\lambda-i+2}$ corresponds to new flow, added to e during iteration i. It is easy to see that during the execution of the procedure the following two invariants are maintained.

*INV*1: After iteration i, all original commodities with demands $1/2^{\lambda-i+1}$ or less have been routed unsplittably.

*INV*2: After iteration $i > 1$, the total flow u_e^i through edge e, which is due to all i first iterations, satisfies the relation

$$u_e^i \leq u_e^{i-1} + 1/2^{\lambda-i+2}.$$

Given that $u_e^1 = u'_e$, the total flow through e used to route unsplittably all commodities in $\mathcal{T}$ is at most

$$u'_e + \sum_{i=2}^{\lambda-\log(\rho_{\max}^{-1})+1} 1/2^{\lambda-i+2} = xu_e + \rho_{\max} - 1/2^{\lambda} = xu_e + \rho_{\max} - \rho_{\min}.$$

Therefore the relaxed decision procedure returns the claimed guarantee. For the running time, we observe that in each of the $l = \lambda - \log(\rho_{\max}^{-1}) + 1$ iterations, the number of virtual commodities appears to be $O(k2^l)$, where k is the number of the original commodities. We opted to present a "pseudopolynomial" version of the decision procedure for clarity. There are two approaches to ensure polynomiality of our method. One is to consider only demands of value $1/2^{\lambda} > 1/n^d$ for some $d > 1$. It is known [27] that we can route the very small demands while affecting the approximation ratio by an $o(1)$ factor. The other approach relies on changing the units at which demands and capacities are expressed. During iteration i scale all demands up by multiplying by a factor of $2^{\lambda-i+1}$. This means that all demands of value $1/2^{\lambda-i+1}$ are now demands of 1. The different virtual commodities hosted by the same sink t are not represented explicitly. Their number will be equal to the total demand of t under the new units. Lemma 4.1 requires the capacity of each edge e to be a multiple $d_e(1/2^{\lambda-i+1})$ of $1/2^{\lambda-i+1}$; change this capacity to d_e. An integral

solution to the problem after remapping demands and capacities in the manner described corresponds to an unsplittable one for the virtual commodities of demand $1/2^{\lambda-i+1}$. □

Recall that for a given UFP we assume that the cut condition holds. In the proof of Theorem 4.1 one can avoid the binary search if all the capacities are multiples of $2^{-\lambda}$. In particular, integral capacities fulfill this requirement. Together with the cut condition, the extra condition implies that under the original capacity function u, a routing g^1 which satisfies all the demands in $\mathcal{T}_1$ can always be found. This gives the following corollary.

COROLLARY 4.1. *Let $\Pi = (G = (V, E, u), s, \mathcal{T})$ be a* UFP *in which all demand values are of the form $2^{-\nu}$ with $\nu \in N$, the maximum demand value is denoted by $\rho_{\max}$, and the minimum demand value is denoted by $\rho_{\min}$. If all capacity values are multiples of $\rho_{\min}$, algorithm* 2H_PARTITION *will obtain in polynomial time an unsplittable routing such that the flow through any edge e is at most $u_e + \rho_{\max} - \rho_{\min}$.*

THEOREM 4.2. *Let $\Pi = (G = (V, E, u), s, \mathcal{T})$ be a* UFP *with maximum and minimum demand values $\rho_{\max}$ and $\rho_{\min}$, respectively. There is an algorithm,* H_PARTITION, *which obtains in polynomial time a*

$$\min\{3 - \rho_{\min}, 2 + 2\rho_{\max} - \rho_{\min}\}$$

approximation for congestion.

Proof. We describe the algorithm H_PARTITION. Round up each demand to its closest power of 1/2. Call the resulting UFP Π' with sink set $\mathcal{T}'$. Multiply all capacities in Π' by at most 2 and let u' denote the resulting capacity function in Π'. Then there is a fractional feasible solution f' to Π'. The purpose of the rounding is to always be able to express the granularity as a power of 1/2. The remainder of the proof consists of finding an unsplittable solution to Π'; this solution can be efficiently converted to an unsplittable solution to Π without further capacity increase. Such a solution to Π' can be found by the algorithm 2H_PARTITION from Theorem 4.1. Observe that the optimum congestion for Π' is at most z, the optimum congestion for Π. 2H_PARTITION will route through edge e flow that is at most

$$zu'_e + \rho'_{\max} - \rho'_{\min} \leq 2zu_e + \rho'_{\max} - \rho'_{\min},$$

where $\rho'_{\max}$ and $\rho'_{\min}$ denote the maximum and minimum demand values in Π'. But $\rho'_{\max} < 2\rho_{\max}$, $\rho'_{\max} \leq 1$, and $\rho'_{\min} \geq \rho_{\min}$ from the construction of Π'. Therefore $\rho'_{\max} - \rho'_{\min} \leq 1 - \rho_{\min}$ and $\rho'_{\max} - \rho'_{\min} \leq 2\rho_{\max} - \rho_{\min}$. Dividing the upper bound on the flow by $zu_e \geq 1$ yields the claimed guarantees on congestion. □

5. Minimizing congestion with arbitrary capacities. We examine here UFP in its full generality, when demands lie in $(0, 1]$ and capacities in $(0, \infty)$. In particular, let $1/D$ be the minimum demand value for a real $D \geq 1$. We assume without loss of generality that $u_e \geq 1/D$ for all e, since the optimal solution will not use any edge violating this condition. Our approximation algorithm will have time complexity polynomial in $\lfloor \log D \rfloor$. In subsequent sections we return to the original balance condition on the capacity values as stated in section 2.

The essential modification required in algorithm PARTITION is at Step 1. We compute a feasible fractional solution such that the flow of commodity i through edge e is set to 0 if $\rho_i > u_e$. These constraints can be easily enforced, for example, in a linear programming formulation. The resulting problem formulation is a valid relaxation of the original problem and the unsplittable optimum is not affected by the additional

constraints. A second modification is that no subproblem is constructed for a range of the form $(0, \alpha_1]$. The first subinterval of the decomposition is $[a_1, a_2]$ with $a_1 = 1/D$. The analysis of the performance guarantee is largely the same. Let L_PARTITION be the modified algorithm as outlined above. We maintain the notation of PARTITION for the decomposition parameters.

LEMMA 5.1. *Algorithm* L_PARTITION *runs in polynomial time and outputs an unsplittable routing g with total flow through edge e at most* $\max_{2\le i\le\nu}\{r(\alpha_{i-1}, \alpha_i)\}u_e + \sum_{i=2}^{\nu}\alpha_i$, *where* $u_e \in (\alpha_{\nu-1}, \alpha_\nu]$, $\nu \le \xi$.

Proof. The proof is similar to that of Lemma 3.3. The additional observation is that because of the new constraints in the fractional relaxation, during Step 2 of the algorithm edge e is not included in any graph G_i with $i \ge \nu$. □

THEOREM 5.1. *Given* UFP Π *with arbitrary capacities and minimum demand value* $\rho_{\min} = 1/D$, $D \ge 1$, *algorithm* L_PARTITION *finds a* $(5.8285 - 1.4144\rho_{\min})$*-approximation for congestion. The algorithm runs in polynomial time.*

Proof. We instantiate the partitioning scheme as follows. The parameter $r > 1$, to be chosen later, is the ratio of the geometrically decreasing intervals. (In the proof of Theorem 3.1, $r = 2$.) We partition the interval $[1/D, 1]$ of demands into $O(\log_r D)$ geometrically increasing subintervals:

$$[1/D, 1/r^{\lfloor \log_r D\rfloor}], \ldots, (1/r^{i+1}, 1/r^i], \ldots, (1/r, 1].$$

By Lemma 5.1 the total flow through an edge e such that $1/r^{i+1} < u_e \le 1/r^i$ is at most

$$ru_e + \sum_{j=i}^{\lfloor \log_r D\rfloor} 1/r^j = ru_e + \frac{r}{r^i(r-1)} - \frac{1}{(r-1)r^{\lfloor \log_r D\rfloor}} \le ru_e + \frac{r^2 u_e}{r-1} - \frac{u_e}{(r-1)D}.$$

Thus we obtain a congestion of $\frac{2r^2-r}{r-1} - \frac{1}{(r-1)D}$. Selecting $r = 1.707$, yields $5.8285 - 1.4144\rho_{\min}$. This analysis hinges on the fact that $u_e \le 1$. The total flow through an edge e with $u_e > 1$ will be

$$ru_e + \sum_{j=0}^{\lfloor \log_r D\rfloor} 1/r^j < ru_e + \frac{r}{r-1}u_e,$$

which is at most $4.12143u_e$ when $r = 1.707$. Since $\rho_{\min} \le 1$, $4.12143 < 5.8285 - 1.4144\rho_{\min}$. □

6. Minimum-cost unsplittable flow. In this section we examine the problem of finding a minimum-cost unsplittable flow for $(G, s, \mathcal{T})$ when a cost $c_e \ge 0$ is associated with each edge e. The cost $c(P)$ of a path P is $\sum_{e\in P} c_e$. The *cost of a routing* is defined as $\sum_{t_i\in\mathcal{T}} \rho_i c(P_i)$. Similarly we define the *cost of an unsplittable flow* to be the cost of the corresponding routing. The optimization problem we consider is defined as follows: find among all routings of minimum congestion the one of minimum cost. Given that this problem has two objectives, we give a bicriteria (α, β)-approximation which finds an unsplittable routing with (i) congestion β times the optimum congestion z and (ii) cost α times the minimum cost of an unsplittable flow which is feasible with respect to the capacity function zu.

We show how to modify algorithm PARTITION to invoke a minimum-cost flow subroutine instead of maximum flow routines. Since PARTITION works on both directed and undirected graphs the same holds for the new algorithm. First we need the analogue of Theorem 2.1.

THEOREM 6.1. *Let $(G = (V, E, u), s, \mathcal{T})$ be a* UFP *with costs on an arbitrary network G with all demands ρ_i, $1 \le i \le k$, equal to the same value σ and edge capacities u_e equal to integral multiples of σ for all $e \in E$. There is a maximum fractional flow of minimum cost that is an unsplittable flow. Moreover, this unsplittable flow can be found in polynomial time.*

This theorem is an easy consequence of the well-known successive shortest path algorithm for minimum-cost flow (developed independently by [17, 16, 3]), and it is a corollary of the integrality property of minimum-cost flow, with integral units scaled by σ.

Two lemmas for the analysis follow. They generalize Lemmas 3.1 and 3.2 to accommodate costs.

LEMMA 6.1. *Given a minimum-cost* UFP *with demands in the interval $(0, \alpha]$ and a fractional solution f with cost $c(f)$, there is an algorithm, α-*C_ROUTING, *which finds an unsplittable routing of cost at most $c(f)$, such that the flow through an edge is at most $n\alpha$. The running time of the algorithm is $O(n \log n + m)$.*

Proof. Algorithm α-C_ROUTING works as follows. Find shortest paths from s to all the sinks in G using Dijkstra's algorithm [4, 10]. For each sink t_i route on the shortest path from s flow equal to the demand ρ_i. □

LEMMA 6.2. *Let a minimum-cost* UFP Π *have demands in the interval $(a, b]$, arbitrary capacities, and a fractional solution f of cost $c(f)$. There is an algorithm,* C_INTERVAL_ROUTING, *which finds in polynomial time an unsplittable routing of cost at most $r(a, b)c(f)$ such that the flow through an edge e is at most $r(a, b)u_e + b$.*

Proof. The proof is similar to that of Lemma 3.2, but we use Theorem 6.1 instead of Theorem 2.1. Round up all the demands to b and call Π' the resulting problem. By multiplying the flow f_e and the capacity u_e on each edge e by at most $r(a, b)$, we obtain a feasible fractional solution f' for Π'; the cost of f' is at most $r(a, b)c(f)$. Now add at most b to each edge capacity so that it becomes a multiple of b. Find an unsplittable routing by using Theorem 6.1. The cost of the unsplittable routing is at most equal to the cost of the fractional solution f'. □

Algorithm C_PARTITION takes the same steps as PARTITION, with two differences in Steps 1 and 3. At Step 1 of C_PARTITION a fractional minimum-cost solution f is found. At Step 3 we invoke C_INTERVAL_ROUTING instead of INTERVAL_ROUTING. C_INTERVAL_ROUTING is implemented with the successive shortest paths algorithm for minimum-cost flow. Finally algorithm α-C_ROUTING is used to compute a routing for graph G_1 of the decomposition. We keep the same notation for the partitioning scheme. The proof is very similar to that of Lemma 3.3, and we omit it.

LEMMA 6.3. *Algorithm* C_PARTITION *outputs an unsplittable routing g with congestion at most*

$$n\alpha_1 + \max_{2 \le i \le \xi} \{r(\alpha_{i-1}, \alpha_i)\} + \sum_{i=2}^{\xi} \alpha_i$$

and cost at most $\max_{2 \le i \le \xi}\{r(\alpha_{i-1}, \alpha_i)\}c(f)$, where $c(f)$ is the minimum cost of a fractional solution. The running time of C_PARTITION *is $O(T_2(n, m) + nm + m\xi)$ where $T_2(n, m)$ is the time to solve a fractional minimum-cost flow problem.*

Using the same partitioning scheme as in Theorem 3.1 we obtain the following.

THEOREM 6.2. *Given a minimum-cost* UFP Π, *we can obtain a simultaneous $(2, 4 + \varepsilon)$-approximation for cost and congestion, for any $\varepsilon > 0$. The running time of the algorithm is $O(T_2(n, m) + nm + m \log(n/\varepsilon))$ where $T_2(n, m)$ is the time to solve a fractional minimum-cost flow problem.*

We observe that the approximation factor for the cost in the above algorithm is the ratio of the subintervals used in the partitioning scheme. By increasing the constant for congestion we can improve upon the cost approximation.

THEOREM 6.3. *Given a minimum-cost* UFP Π *we can obtain a simultaneous* $(1+\delta, 2+\frac{\delta^2+1}{\delta}+\varepsilon)$*-approximation for cost and relative congestion for any* $\delta, \varepsilon > 0$. *The running time of the algorithm is* $O(T_2(n,m) + m\log(n/\varepsilon))$, *where* $T_2(n,m)$ *is the time to solve a fractional minimum-cost flow problem.*

Proof. The new algorithm is the same as C_PARTITION except for the partitioning scheme. The interval $(0,1]$ of demands is partitioned into ξ geometrically increasing subintervals

$$(0, 1/(\delta+1)^{\xi-1}], \ldots, (1/(\delta+1)^{i+1}, 1/(\delta+1)^i], \ldots, (1/(\delta+1)^2, 1/(\delta+1)], (1/(\delta+1), 1]$$

such that $1/(\delta+1)^{\xi-1} \le \varepsilon/n$. According to this scheme, by Lemma 6.3 the approximation ratio for the cost will be $1+\delta$ and the approximation ratio for the congestion will be at most

$$\begin{aligned} 1 + \delta + \sum_{i=0}^{\xi-2} \frac{1}{(\delta+1)^i} + \varepsilon \\ \le 1 + \delta + \frac{1+\delta}{\delta} + \varepsilon \\ = 2 + \frac{\delta^2+1}{\delta} + \varepsilon. \end{aligned}$$

□

Currently the best time bound $T_2(n,m)$ for fractional minimum-cost flow is

$$O(\min\{nm\log(n^2/m)\log(nC), nm(\log\log U)\log(nC), (m\log n)(m + n\log n)\})$$

due to [14, 1, 33]. Finally, by modifying the congestion algorithm of Theorem 4.2, it is easy to see how to obtain a $(2,3)$ simultaneous approximation for cost and congestion. The essential modification to H_PARTITION lies in the use of the minimum-cost analogue to Lemma 4.1.

LEMMA 6.4. *Let* $\Pi = (G, s, \mathcal{T})$ *be a minimum-cost* UFP *in which all demands have value* $1/2^x$ *for* $x \in N$ *and all capacities are multiples of* $1/2^{x+1}$, *and let* f *be a fractional flow solution of cost* $c(f)$ *such that the flow* f_e *through any edge is a multiple of* $1/2^{x+1}$. *We can find in polynomial time an unsplittable routing* g *of cost at most* $c(f)$ *such that the flow* g_e *through an edge* e *is at most* $f_e + 1/2^{x+1}$.

THEOREM 6.4. *Let* Π *be a minimum-cost* UFP, *with maximum and minimum demand values* $\rho_{\max}$ *and* $\rho_{\min}$, *respectively. We can obtain in polynomial time a simultaneous*

$$(2, \min\{3 - \rho_{\min}, 2 + 2\rho_{\max} - \rho_{\min}\})$$

approximation for cost and congestion.

7. Maximizing the routable demand. In this section we turn to the maximization problem of finding a routable subset of sinks with maximum total demand. Let the *value* of a (partial) routing be the sum of the demands of the commodities being routed. We seek a routing of maximum value such that the capacity constraints are satisfied. We obtain a .075 approximation which is slightly improved in the next section to .0769 as an indirect application of the algorithm for minimizing the number

of rounds; the latter algorithm uses subroutines from matroid theory. In contrast to most of the results in this paper, the cut condition is a necessary prerequisite for the .0769-approximation. Therefore the .075-approximation is our most generally applicable algorithm for maximum demand. The exposition in the present section introduces in a direct fashion the general technique while the algorithm invokes only maximum flow subroutines. Note that if the value definition was relaxed to take into account paths carrying flow to commodities without fully satisfying the respective demands, a c-approximation, $c > 1$, for congestion would imply a $1/c$-approximation for this relaxed maximization problem.

Let $(G, s, \mathcal{T})$ be the given UFP. Consider splitting the interval $(0, 1]$ of demands into two intervals, $(0, \alpha]$ and $(\alpha, 1]$, for an appropriate *breakpoint* α. We will give separate constant-factor approximation algorithms for each of the two resulting problems. At least one of the two intervals contains at least half of the total demand in $\mathcal{T}$, so we obtain a constant-factor approximation for the entire problem. This was also the high-level approach used in [23]. In fact, in order to route the demands in the subinterval $(\alpha, 1]$ we will use an intuitive algorithm developed by Kleinberg [23]. For the demands in $(0, \alpha]$, we show how a partitioning scheme, similar to the one used for minimizing congestion, can give a simple algorithm with a constant performance guarantee. For our partitioning scheme to succeed in the maximization setting, it is essential to have the maximum demand bounded away from 1. Hence the different treatment for $(0, \alpha]$ and $(\alpha, 1]$.

Let $\alpha^f(G, s, \mathcal{T})$ denote the total demand routed by a fractional solution f. For any given UFP we will use $\alpha^f(G, s, \mathcal{T})$, with f a fractional solution of maximum value, as an upper bound to measure the quality of our approximation.

The following lemma, shown in [23], will be used for demands in $(\alpha, 1]$. The spirit of the proof is similar to that of Lemma 3.2.

LEMMA 7.1 (see [23]). *Given a* UFP $\Pi = (G, s, \mathcal{T})$ *with demands in the interval* $(a, b]$, *there is an algorithm,* K_ROUTING, *which finds a partial unsplittable flow* g *of value at least* $\frac{1}{2}[r(a, b)]^{-1}\alpha^f(G, s, \mathcal{T})$. *The algorithm runs in polynomial time.*

Adapting the proof of Lemma 7.1 we can show a slightly different lemma, which will be also of use in our algorithm. We include the proof for the sake of completeness.

LEMMA 7.2. *Given a* UFP $\Pi = (G, s, \mathcal{T})$ *with demands in the interval* $(a, b]$ *and capacities that are multiples of* b, *there is an algorithm,* M_ROUTING, *which finds, in polynomial time, a partial unsplittable flow* g *of value at least* $[r(a, b)]^{-1}\alpha^f(G, s, \mathcal{T})$, *where* f *is a fractional solution to* Π.

Proof. We outline algorithm M_ROUTING. Round down all demands to a. Multiply all capacities in G by $[r(a, b)]^{-1} < 1$. Let $\Pi' = (G', s, \mathcal{T}')$ be the resulting problem. We show that Π' has a fractional solution f' of value $\alpha^{f'}(G', s, \mathcal{T}') \geq [r(a, b)]^{-1}\alpha^f(G, s, \mathcal{T})$: multiply the flow pushed by f on each edge by $[r(a, b)]^{-1}$ and let f' be the resulting fractional flow. Clearly f' does not route more than a units of flow to any sink in $\mathcal{T}'$ and respects the capacities in G', so it is feasible for Π'.

We now show how to obtain a partial unsplittable solution for Π of value at least $\alpha^{f'}(G', s, \mathcal{T}')$. In G all capacities are multiples of b so in G' they are multiples of a. But all demands in G' are a so by Theorem 2.1 there is an unsplittable flow g' of value at least $\alpha^{f'}(G', s, \mathcal{T}')$. Multiplying the flow on each path in g' by a factor of at most $r(a, b)$ gives a partial unsplittable flow g for Π, respecting the capacities in G and of value at least $\alpha^{f'}(G', s, \mathcal{T}')$. ∎

We now outline our approach for the second half of the problem, namely the demands in $(0, \alpha]$. Our aim is to use a partitioning scheme as in the congestion case:

ALGORITHM M_PARTITION$(G=(V,E,c), s, \mathcal{T}, \xi, \alpha_1, \ldots, \alpha_\xi, x)$

***Step* 1.** Find a feasible fractional solution f.

***Step* 2.** Let x be a constant, $0 < x < 1$. Define a partition of the $(0,1]$ interval into ξ consecutive subintervals $(0,\alpha_1], (\alpha_1,\alpha_2], \ldots, (\alpha_{\xi-1},\alpha_\xi]$, $\alpha_\xi = 1$, such that $\sum_{i=1}^{\xi-1} \alpha_i \leq x$. Construct $\xi - 1$ copies of G where the set of sinks in G_i, $2 \leq i \leq \xi$, is the subset $\mathcal{T}_i$ of $\mathcal{T}$ with demands in the interval $(\alpha_{i-1}, \alpha_i]$. Using flow decomposition determine for each edge e the amount c_e^i of u_e used by f to route flow to sinks in $\mathcal{T}_i$.

***Step* 3.** Set the capacity of edge e in G_i, $2 \leq i \leq \xi - 1$, equal to the smallest multiple of α_i greater than or equal to $(1-x)c_e^i$. Set the capacity of edge e in G_ξ to u_e.

***Step* 4.** Invoke M_ROUTING on each G_i, $2 \leq i \leq \xi - 1$, to obtain a partial unsplittable flow g^i. Invoke K_ROUTING on G_ξ to obtain a partial unsplittable flow g^ξ.

***Step* 5.** Set partial routing g to be the union of the path sets g^i, $2 \leq i \leq \xi - 1$. Of the two partial routings g and g^ξ output the one of greater value.

FIG. 8. *Algorithm* M_PARTITION.

partition the interval $(0, \alpha]$ of demand values into subintervals and generate an appropriate subproblem for each subinterval. We would like to find a near-optimal solution to each subproblem, by exploiting Lemma 7.2. Ideally, we could then combine these near-optimal solutions to the subproblems to obtain a close-to-maximum unsplittable flow for the original problem. However, in the congestion version, only a fraction of the capacity of an edge is assigned initially to each subproblem, a fraction determined by flow decomposition. On the other hand, in order to make use of Lemma 7.2 on each subproblem, we require capacities to be multiples of the maximum demand. In the congestion algorithm, adding the necessary amount to each capacity only increased the approximation ratio. In the current setting we are not allowed to increase the original input capacities, as this would violate feasibility. To circumvent this difficulty we scale down the fractional solution to the original problem. On each edge a constant fraction of the edge capacity is then left unused and can provide the required extra capacity for the subproblems. The scaling of the fractional solution by a $1-x$ factor, for appropriately chosen x, will incur a scaling by the same factor to the approximation ratio, as the value of the final routing is given in terms of the value $\alpha^f(G, s, \mathcal{T})$ of the maximum flow with which we begin. We give the general algorithm in Figure 8. It takes as input a UFP $(G, s, \mathcal{T})$ together with a set of parameters $x, \xi, \alpha_1, \ldots, \alpha_\xi$ that will determine the exact partitioning scheme. The breakpoint α will be $\alpha_{\xi-1}$ with α_ξ set to 1. Subsequently we show how to choose these parameters so as to optimize the approximation ratio. The algorithm outputs a partial routing.

By Lemma 7.1, partial routing g^ξ computed in Step 4 is a partial unsplittable flow, which respects the capacities. We need to establish this fact for partial routing g computed in Step 5 as well and also get an estimate of its value. Note that algorithm M_PARTITION does not route any demands in $(0, \alpha_1]$. Let $\alpha^f(G, s, \mathcal{T}_i)$ denote the flow routed by the fractional solution to demands in the interval $(\alpha_{i-1}, \alpha_i]$.

LEMMA 7.3. *Partial routing g found by algorithm* M_PARTITION *is a partial*

unsplittable flow of value at least

$$\min_{2\le i\le \xi-1}\{[r(\alpha_{i-1},\alpha_i)]^{-1}\}(1-x)\alpha^f\left(G,s,\bigcup_{2\le i\le \xi-1}\mathcal{T}_i\right).$$

Moreover, g respects the capacity constraints in G. The running time of M_PARTITION *is $O(T_1(n,m)+nm+m\xi)$, where $T_1(n,m)$ is the time to solve a fractional maximum flow problem.*

Proof. We examine first the value of g. Let f_i^x be the fractional optimal flow on G_i. By Lemma 7.2 the value of the partial routing g_i, $2\le i\le \xi-1$, found at Step 5, is at least $[r(\alpha_{i-1},\alpha_i)]^{-1}f_i^x$. By the capacity assignment in G_i, $\alpha^{f_i^x}(G,s,\mathcal{T}_i)\ge (1-x)\alpha^f(G,s,\mathcal{T}_i)$. Hence the value of the partial routing g^i is at least

$$(1-x)[r(\alpha_{i-1},\alpha_i)]^{-1}\alpha^f(G,s,\mathcal{T}_i).$$

Since $\alpha^f(G,s,\bigcup_{2\le i\le \xi-1}\mathcal{T}_i)=\sum_{i=2}^{i=\xi-1}\alpha^f(G,s,\mathcal{T}_i)$, the claim on the value follows.

For the capacity constraints, the aggregate capacity used in all partial routings g^i, $2\le i\le \xi-1$, on any edge e, is by Step 4 at most $(1-x)\sum_{i=2}^{i=\xi-1}c_e^i+\sum_{i=2}^{i=\xi-1}\alpha_i$. By Step 2, $\sum_{i=2}^{\xi-1}\alpha_i\le x$ and $\sum_{i=2}^{i=\xi-1}c_e^i\le u_e$. Thus, the aggregate capacity used by g is at most $(1-x)u_e+x\le u_e$. The running time follows in the same manner as in Lemma 3.3. ∎

It remains to choose the parameters x,ξ, and the α_i of the partitioning and to account for the "missing" flow $\alpha^f(G,s,\mathcal{T}_1)$. Without loss of generality we assume that there is one demand in $\mathcal{T}$ of value 1. Otherwise we can rescale the interval of demands. Thus $\alpha^f(G,s,\mathcal{T})$ is at least 1.

THEOREM 7.1. *Let x_1,x_2,x_3 be constants in $(0,1)$ such that $x_3=x_1(1-x_2)$. Given a* UFP $\Pi=(G,s,\mathcal{T})$ *and choosing parameters based on x_1,x_2,x_3, algorithm* M_PARTITION *finds a $\beta(1-\varepsilon)$-approximation for maximum routable demand for any $0<\varepsilon<\beta$ and $\beta=\min\{(1-x_1)x_2/2,x_3/4\}$. The running time of the algorithm is $O(T_1(n,m)+nm+m\log(n/\varepsilon))$ where $T_1(n,m)$ is the time to solve a fractional maximum flow problem.*

Proof. At Step 3 of M_PARTITION, partition the interval $(0,1]$ of demand values into ξ geometrically increasing subintervals

$$(0,x_3(x_2)^{\xi-2}],\ldots,(x_3(x_2)^{i+1},x_3(x_2)^i],\ldots,(x_3x_2,x_3],(x_3,1]$$

such that $x_3(x_2)^{\xi-2}\le\varepsilon/n$. Thus it suffices for ξ to be $\Theta(\log n/\varepsilon)$. The flow $\alpha^f(G,s,\mathcal{T}_1)$ is at most ε so

$$\alpha^f\left(G,s,\bigcup_{2\le i\le \xi}\mathcal{T}_i\right)\ge\alpha^f(G,s,\mathcal{T})-\varepsilon\ge(1-\varepsilon)\alpha^f(G,s,\mathcal{T}).$$

Achieving therefore a β-approximation to $\alpha^f(G,s,\bigcup_{2\le i\le \xi}\mathcal{T}_i)$ will yield the claimed $\beta(1-\varepsilon)$ ratio.

Set parameter x in the algorithm to x_1. Note that by the choice of x_3 the sum $\sum_{i=0}^{i=\xi-2}x_3(x_2)^i$ is at most x_1 as required by the algorithm. By Lemma 7.3, the partial unsplittable flow g found by M_PARTITION is a $(1-x_1)x_2$-approximation to $\alpha^f(G,s,\bigcup_{2\le i\le \xi-1}\mathcal{T}_i)$. By Lemma 7.1, the partial unsplittable flow g^ξ is a $x_3/2$-approximation to $\alpha^f(G,s,\mathcal{T}_\xi)$. Choosing of the two the one of greatest value at Step 5 yields a β-approximation to $\alpha^f(G,s,\bigcup_{2\le i\le \xi}\mathcal{T}_i)$. ∎

By choosing x_1,x_2, and x_3 to be $4/10,1/4$, and $3/10$, respectively, we obtain $(1-x_1)x_2=x_3/2=0.15$. Accordingly, $\beta=.075$.

8. Minimizing the number of rounds. In this section we examine the problem of routing in rounds. A partitioning of the set of sinks into a minimum number of communication rounds is sought so that the set of terminals assigned to each round is routable under the capacity constraints. The requirement to satisfy the capacity constraints leads to a high-level treatment similar to the one for the maximum demand metric. We split the interval of demand values into $(0, \alpha]$ and $(\alpha, 1]$, for an appropriate breakpoint α, and give a constant-factor approximation for each of the two resulting problems. Demands in $(0, \alpha]$ will be processed under a partitioning scheme that will generate subproblems. In the subproblems, we relax the capacities to allow congestions of up to $1/(1-x)$ in order to free up capacity that we can reallocate conveniently among subproblems. This scaling will be reflected in an increase on the approximation ratio by $1/(1-x)$. Intuitively, since we are using paths from the fractional solution in the unsplittable routings, when the fractional solution carries $(1-x)$ times less flow for each commodity, we need more rounds.

Recall that the input UFP comes with the assumption that a fractional solution with relative congestion 1 exists, i.e., the cut condition holds. In this section we will generate subproblems on which this assumption does not hold. Therefore we introduce notation $\zeta^f(G, s, \mathcal{T})$ for the congestion of a *fractional* solution f to a UFP $(G, s, \mathcal{T})$. Note that if f is a fractional solution of minimum congestion for a problem $(G, s, \mathcal{T})$, the quantity $\lceil \zeta^f(G, s, \mathcal{T}) \rceil$ is a lower bound on the minimum number of rounds, which we denote by $\chi(G, s, \mathcal{T})$. Our analysis bounds the number of rounds against $\zeta^f(G, s, \mathcal{T})$, which we have assumed to be 1 (because of the cut condition). But since $\chi(G, s, \mathcal{T}) \geq \lceil \zeta^f(G, s, \mathcal{T}) \rceil$, the approximation ratio holds even if we drop this assumption. For the sake of clarity we include $\zeta^f(G, s, \mathcal{T})$ in the approximation guarantees because we find that the rounds metric is the least intuitive of the three metrics as far as validity of the approximations without the cut condition is concerned.

We employ a subroutine that we call KR_ROUTING (see Lemma 5.3 in [23]) and a variant R_ROUTING to deal with subproblems having demands in a bounded range. The subroutines are similar in spirit to the subroutines K_ROUTING and M_ROUTING used for the maximum demand metric; however, their basic ingredient is not Theorem 2.1. Maximum flow integrality is not useful in the rounds setting and instead a result from [23] is used, given as Theorem 8.1 below. The proof of the latter theorem uses results from matroid theory.

THEOREM 8.1 (see [23]). *Given a* UFP $(G, s, \mathcal{T})$ *with all demands equal to* σ *and all capacities multiples of* σ, $\lceil \zeta^f(G, s, \mathcal{T}) \rceil = \chi(G, s, \mathcal{T})$. *Moreover, a routing in* $\chi(G, s, \mathcal{T})$ *rounds can be found in polynomial time.*

LEMMA 8.1 (see [23]). *Let* $\Pi = (G, s, \mathcal{T})$ *be a* UFP *with demands in the interval* $(a, b]$ *and* f *a corresponding fractional solution. There is a polynomial-time algorithm,* KR_ROUTING, *which routes* Π *in at most* $\lceil 2r(a, b)\zeta^f(G, s, \mathcal{T}) \rceil$ *rounds.*

Again we can adapt the proof of Lemma 8.1 to show a slightly different result.

LEMMA 8.2. *Let* $\Pi = (G, s, \mathcal{T})$ *be a* UFP *with demands in the interval* $(a, b]$, *and capacities multiples of* b, *and* f *a corresponding fractional solution not necessarily respecting capacities. There is a polynomial-time algorithm,* R_ROUTING, *which routes* Π *in at most* $\lceil r(a, b)\zeta^f(G, s, \mathcal{T}) \rceil$ *rounds.*

Proof. We outline algorithm R_ROUTING. Round all the demands up to b. Call Π' the resulting problem and f' a corresponding fractional solution not necessarily respecting capacities. In Π' all demands are equal to b and all capacities are multiples of b; therefore by Theorem 8.1 we can route in $\chi(\Pi') = \lceil \zeta^{f'}(\Pi') \rceil$ rounds and use the paths from this routing to route Π. Clearly, the paths used in each round respect the

capacity constraints in $(G, s, \mathcal{T})$. This completes the description of the algorithm.

It remains to demonstrate a suitable f', which will help us to upper bound $\zeta^{f'}(\Pi')$ in terms of $\zeta^f(G, s, \mathcal{T})$. Decompose the flow f for Π and consider the paths routing demand to a particular sink t_i. Multiply the flow on each path by the same constant (no greater than b/a) so that b units of flow are routed to each t_i. Do this for all sinks. Let f' be the resulting fractional solution; f' satisfies all the demands for Π' and has congestion $\zeta^{f'}(\Pi')$ at most $r(a, b)\zeta^f(G, s, \mathcal{T})$. □

Our algorithm R_Partition and its analysis are very similar to M_Partition up to the subroutine level. When dealing with $(\alpha_1, \alpha_{\xi-1}]$, we scale down the fractional capacity on each edge by $(1 - x)$, thereby inducing congestion of up to $1/(1 - x)$ in the subproblems, in order that we can reallocate capacity among the subproblems to satisfy the hypotheses of Lemma 8.2. Steps 1 through 3 are exactly the same for both algorithms. In Step 4 routines R_Routing and KR_Routing are invoked on G_i, $2 \le i \le \xi - 1$, and G_ξ, respectively. Algorithm R_Routing outputs for each G_i, $2 \le i \le \xi - 1$, a set of paths $\mathcal{P}_{ij}$ to be used on round j. Let j^* be the maximum number of rounds output from R_Routing for any G_i, $2 \le i \le \xi - 1$. During round j, $1 \le j \le j^*$, we route along all paths in $\bigcup_i \mathcal{P}_{ij}$. Subsequently we route the demands in $(0, \alpha_1]$ and $(\alpha_{\xi-1}, 1]$ in two separate sets of rounds. Recall that $\zeta^f(G, s, \mathcal{T}_i)$ is the minimum congestion for routing demands fractionally in $\mathcal{T}_i$ on the original unscaled capacities and thus is equal to 1. The following lemma accounts for the rounds needed to route all demands except for the ones in the $(0, \alpha_1]$.

Lemma 8.3. *Algorithm* R_Partition *runs in polynomial time and routes the demands in* $(\alpha_1, 1]$ *in at most*

$$\max_{2 \le i \le \xi - 1} \left\{ \left\lceil \frac{1}{1 - x} r(\alpha_{i-1}, \alpha_i) \zeta^f(G, s, \mathcal{T}_i) \right\rceil \right\} + \left\lceil 2r(a_{\xi-1}, 1)\zeta^f(G, s, \mathcal{T}_\xi) \right\rceil$$

rounds.

Proof. For the number of rounds to route demands in $(\alpha_i, \alpha_{i+1}]$, $1 \le i \le \xi - 2$, we note that Lemma 8.2 applies in the corresponding subproblems. A fractional solution satisfying all demands in subproblem G_i would have to potentially introduce congestion given that at Step 3 we assign capacity to edge e in G_i which is as low as $(1 - x)c_e^i$. There is a fractional solution f_i to G_i with congestion at most $\frac{1}{1-x}\zeta^f(G, s, \mathcal{T}_i)$, obtained by setting $f_e^i = c_e^i$. Therefore, the number of rounds for a subproblem is at most $\lceil \frac{1}{1-x} r(\alpha_{i-1}, \alpha_i)\zeta^f(G, s, \mathcal{T}_i) \rceil$. By the same argument as in Lemma 7.3, the aggregate capacity on any edge e used on all these subproblems does not exceed u_e. Thus during the same round j, $1 \le j \le j^*$, we can route all paths in $\bigcup_{2 \le i \le \xi - 1} \mathcal{P}_{ij}$.

To route the demands in $(\alpha_{\xi-1}, 1]$ we need by Lemma 8.1 at most an additional $\lceil 2r(a_{\xi-1}, 1)\zeta^f(G, s, \mathcal{T}_\xi) \rceil$ number of rounds. □

By choosing $\alpha_1 \le 1/n$, all the demand in $(0, \alpha_1]$ can be routed in one round, by Lemma 3.1.

Theorem 8.2. *Let* x_1, x_2, x_3 *be constants in* $(0, 1)$ *so that* $x_3 = x_1(1 - x_2)$. *Given a* Ufp $\Pi = (G, s, \mathcal{T})$, *we can obtain in polynomial time a* β*-approximation for minimum number of rounds where* $\beta = \lceil 1/(1 - x_1)x_2 \rceil + \lceil 2/x_3 \rceil + 1$.

Proof. The partitioning scheme is the same as in the proof of Theorem 7.1 with $\varepsilon = 1$. The demands in $(0, x_3(x_2)^{\xi-2}]$ are small enough to be routed in one round using the algorithm α-Routing from Lemma 3.1. Substituting x_1, x_2, x_3 in the number of rounds given by Lemma 8.3 and adding one extra round for the demands in $(0, x_3(x_2)^{\xi-2}]$ complete the proof. □

By choosing x_1, x_2, x_3 to be $1/2, 1/2$, and $1/4$, respectively, we obtain $\beta = 13$.

Routing all commodities in x rounds implies that during one round at least a $1/x$ fraction of the total demand is routed. If for the problem $\Pi = (G, s, \mathcal{T})$ we begin with $\zeta^f(G, s, \mathcal{T}) = 1$, i.e., the cut condition is met, Theorem 8.2 implies that we can route all commodities in at most 13 rounds. During one of those at least $1/13 = .0769$ of the total demand is routed.

COROLLARY 8.1. *Given a* UFP $\Pi = (G, s, \mathcal{T})$ *which satisfies the cut condition, we can obtain in polynomial time a* .0769*-approximation for maximizing routable demand.*

Observe that Theorem 7.1 guarantees routing at least .075 of $\alpha^f(G, s, \mathcal{T})$, which is an upper bound on the maximum routable demand but less than the total demand if the cut condition is not met. We do not see how to apply Theorem 8.2 to obtain a guarantee for maximum demand when the cut condition is violated.

9. A hardness result for unsplittable flow. In this section we consider the hardness of approximation for minimum congestion unsplittable flow on a directed network with 2 sources (2-UFP). We give a gap-preserving reduction from the NP-complete problem 3-D MATCHING [18]. Our reduction establishes that it is NP-hard to achieve an approximation ratio better than 2 for 2-UFP. In the 3-D MATCHING problem, we are given a set $M \subseteq A \times B \times C$, where A, B, and C are disjoint sets each of cardinality n. The objective is to find a *perfect matching,* i.e., a subset $M' \subseteq M$ such that $|M'| = n$ and no two elements of M' agree in any coordinate. In our reduction we use ideas from Theorem 5 in [32].

LEMMA 9.1. *Given an instance* I *of* 3-D MATCHING, *we can obtain in polynomial time an instance* I' *of* 2-UFP *on a directed network such that*

$$I \in \text{3-D MATCHING} \Rightarrow OPT(I') = 1,$$

$$I \notin \text{3-D MATCHING} \Rightarrow OPT(I') \geq 2.$$

Proof. Let $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, and $C = \{c_1, \ldots, c_n\}$ so that the m triples in I are of the form (a_i, b_j, c_k). We show how to construct a directed network G for the corresponding instance I' of unsplittable flow. G contains two source vertices s_d and s_c. For each a_i occurring in t_i triples in I, G contains $t_i - 1$ vertices called the *dummies.* We denote as $\bar{d_{i\tau}}$, $1 \leq \tau \leq t_i - 1$, the dummy vertices corresponding to a_i. For each element b_j and c_k in I, we have, respectively, a vertex b_j and $\bar{c_k}$ in I'. We refer collectively to the $2n$ vertices b_j, $\bar{c_k}$, $1 \leq j, k \leq n$, as *b-* and *c-type vertices,* respectively. There are m commodities corresponding to terminal pairs $(s_c, \bar{c_k})$, $1 \leq k \leq n$, and $(s_d, \bar{d_{i\tau}})$, $1 \leq i \leq n, 1 \leq \tau \leq t_i - 1$, and they have demand 1 each. We use the bar symbol to emphasize that a vertex has a nonzero demand. All the edges in G have capacity 1.

For each of the m triples in I there will be a *triple gadget* (see Figure 9) in G. Let the μth triple be (a_i, b_j, c_k). The gadget for μ contains two dedicated vertices: x_μ and y_μ. The edges of the gadget are (x_μ, y_μ), (b_j, x_μ), $(y_\mu, \bar{c_k})$ and there is an edge from y_μ to each $\bar{d_{i\tau}}$, $1 \leq \tau \leq t_i - 1$. This completes the description of a gadget. For convenience let us identify a gadget with the triple it represents. Finally, an edge is directed from s_d to each x_μ, $1 \leq \mu \leq m$, and from s_c to each b_j, $1 \leq j \leq n$.

Since all demands and capacities are 1, each feasible unsplittable flow is simply a set of disjoint source-sink paths each carrying unit flow. Note that each flow path must pass through some gadget μ and in particular through the edge (x_μ, y_μ) of the gadget.

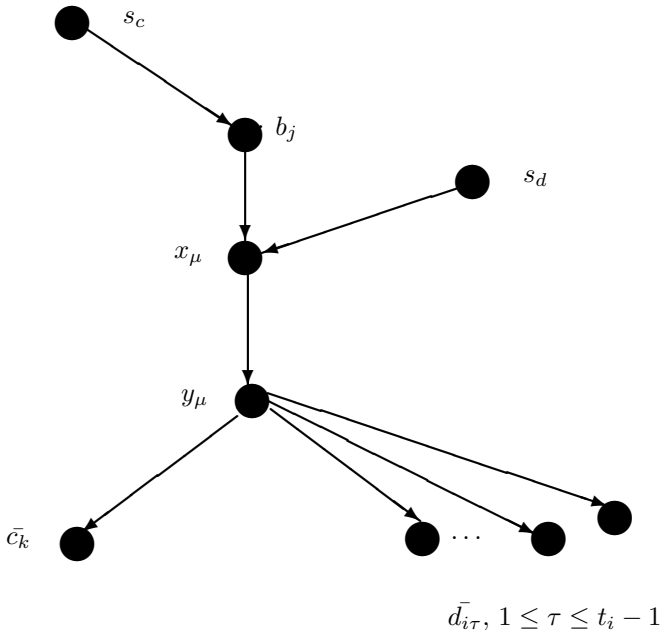


FIG. 9. *Gadget corresponding to the μth triple (a_i, b_j, c_k) together with the edges connecting it to the two sources.*

If I contains a three-dimensional (3-D) perfect matching M', it is straightforward to construct an unsplittable flow g in G with congestion 1. Flow g uses the n gadgets corresponding to triples in M' to satisfy the demands of vertices $\bar{c_k}$, $1 \le k \le n$. The remaining gadgets are used to route 1 unit of flow each (via vertex y_μ for the μth gadget) to a dummy vertex. Since M' is a matching, only one of the t_i gadgets associated with a_i, for any i, is used to route flow to a c-type vertex. As a result, for any i, there are $t_i - 1$ gadgets available to route the dummy vertices associated with a_i.

We show now that if I' has an unsplittable flow g', which satisfies the capacity constraints, then I contains a 3-D perfect matching. Let Γ be the set of n gadgets used by g' to route flow to c-type vertices. We claim that Γ forms a 3-D perfect matching in I. For each $1 \le i \le n$, each dummy vertex $\bar{d_{i\tau}}$ $(1 \le \tau \le t_i - 1)$ requires one flow path, so only one gadget in Γ corresponds to α_i. Each node b_i has in-degree 1, so only one gadget in Γ contains b_i, and the demand at each node c_i is 1, so only one gadget in Γ contains c_i. □

The ensuing theorem is an immediate consequence of Lemma 9.1.

THEOREM 9.1. *No ρ-approximation algorithm, $\rho < 2$, exists for* 2-UFP *on directed graphs unless $P = NP$. The result holds even when all capacities and demands are equal to* 1.

10. Restricted sets of demands and applications to scheduling. In this section we examine connections between unsplittable flow and scheduling problems. Consider the scheduling problem $\mathcal{S}$ defined as follows. A set J of jobs is to be scheduled on a set M of parallel nonidentical machines. A job j can be scheduled to run with

processing time p_j on a set of machines $M(j)$ and has processing time ∞ on all machines in $M - M(j)$. In other words, it is technologically infeasible for job j to run on machines in $M - M(j)$. The objective is to find a schedule which minimizes the *makespan,* i.e., the maximum completion time of a job. $\mathcal{S}$ is a special case of minimizing makespan on unrelated machines. In the unrelated machine setting, job j has a machine-dependent processing time p_{ij} on $i \in M$. The best approximation algorithm known for $\mathcal{S}$ is the 2-approximation for unrelated machine scheduling [32, 38]. From a straightforward modification to Theorem 5 in [32], the following hardness result is obtained.

THEOREM 10.1 (see [32]). *Unless $P = NP$, no approximation better than $3/2$ exists for $\mathcal{S}$ with processing times from the set $\{1, 2, \infty\}$.*

Kleinberg [23] gave an approximation-preserving reduction from $\mathcal{S}$ to minimum congestion unsplittable flow on a three-level directed graph G. A source vertex has edges directed to vertices representing the machines. The vertex set of G contains also one vertex for each job. Machine vertex i has an edge directed to job vertex j if and only if $i \in M(j)$. The edges out of the source have capacity T and the edges into the job vertices have infinite capacity. Finally each job vertex has demand equal to p_j. An unsplittable routing in G where at most ρT amount of flow is pushed through any edge corresponds to a schedule with makespan ρT for $\mathcal{S}$. See Figure 2 for an example network.

Consider a UFP on an arbitrary network with demands p and $2p$, for some $p > 0$. This includes the family of networks obtained for the scheduling problem with processing times $\{1, 2, \infty\}$. As a corollary to Theorem 4.1 and Corollary 4.1 we obtain a tight approximation ratio of $3/2$. The lower bound comes from Theorem 10.1.

COROLLARY 10.1. *Given a* UFP $\Pi = (G, s, \mathcal{T})$ *with demands from the set $\{p, 2p\}$, $0 < p \leq 1/2$, there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge e is at most $zu_e + p$, where z is the optimal congestion. If all capacities are multiples of p the flow is at most $u_e + p$. Thus the approximation ratio for congestion is at most $3/2$. This is best possible, unless $P = NP$.*

COROLLARY 10.2. *For problem $\mathcal{S}$ with processing times from $\{p, 2p, \infty\}$ there is a polynomial-time algorithm, which outputs a schedule within an additive p of the optimum makespan. The approximation ratio is at most $3/2$ and this is best possible, unless $P = NP$.*

It is instructive to consider the action taken by algorithm 2H_PARTITION on the UFP resulting from scheduling problem $\mathcal{S}$. The algorithm splits each job of processing time $2p$ into two virtual jobs each of processing time p. Then a schedule g_1 with optimum makespan is computed for the resulting instance. In the scheduling context, when all the jobs have the same processing time, an optimum schedule can be found by solving an assignment problem. Schedule g_1 corresponds to a half-integral superoptimal solution for $\mathcal{S}$. In the second phase of 2H_PARTITION this half-integral solution is rounded to an integral one.

The results of Corollary 10.1 can be generalized to the following.

COROLLARY 10.3. *Given a* UFP $\Pi = (G, s, \mathcal{T})$, *with demands from the set $\{p, Cp\}$, $C > 1, 1/C \geq p > 0$, there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge e is at most $zu_e + (C-1)p$, where z is the optimal congestion. If all capacities are multiples of p the flow is at most $u_e + (C-1)p$. Thus the approximation ratio for congestion is at most $2 - \frac{1}{C}$.*

We now proceed to examine the case in which the demands lie in the interval $[p, Cp]$, $p > 0$. Lemma 3.2 would give in this setting a $(C+1)$-approximation for

congestion, since the minimum capacity is now Cp. We show how to achieve a C-approximation, the same as the ratio of the interval.

THEOREM 10.2. *Given a* UFP $\Pi = (G, s, \mathcal{T})$ *with demands from the interval* $[p, Cp]$, $C > 1, p > 0$, *there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge* e *is at most* Cu_e.

Proof. As in the proof of Theorem 4.1 we assume without loss of generality that Π has an unsplittable routing with congestion 1. Otherwise one can use the algorithm we propose as a C-relaxed decision procedure [15] in conjunction with a binary search for the optimum congestion to obtain the claimed approximation. The algorithm is as follows. Round all demands down to p. Call the resulting problem Π'. Since an unsplittable flow solution exists for Π, one exists for Π' as well. Rounding down the capacities of edges to the closest multiple of p does not affect the existence of this solution. But for Π' all demands are p and all capacities are multiples of p. Therefore, by Theorem 2.1 we can find an unsplittable flow solution g' in polynomial time. To obtain from g' an unsplittable routing for Π it suffices to increase the flow along paths in g' that lead to sinks in $\mathcal{T}$ with demands more than p. The increase will be at most by a multiplicative factor of C on each path, hence the approximation theorem. ∎

Interestingly, the approach in Theorem 10.2 does not seem to yield an improvement on the result of Lemma 3.3. For problem Π' in the proof above, the capacities are already multiples of p, an assumption we cannot make in the setting of Lemma 3.2.

Acknowledgments. Thanks to Aravind Srinivasan for useful discussions and to Michel Goemans for sending us a copy of [5]. We also wish to thank the two anonymous referees for their comments that substantially improved the presentation and led to a strengthening of Theorem 5.1.

REFERENCES

[1] R. K. AHUJA, A. V. GOLDBERG, J. B. ORLIN, AND R. E. TARJAN, *Finding minimum-cost flows by double scaling*, Math. Program., 53 (1992), pp. 243–266.

[2] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] R. G. BUSAKER AND P. J. GOWEN, *A Procedure for Determining Minimal-Cost Flow Network Patterns*, Technical report ORO-15, Operational Research Office, Johns Hopkins University, Baltimore, MD, 1961.

[4] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 260–271.

[5] Y. DINITZ, N. GARG, AND M. X. GOEMANS, *On the single-source unsplittable flow problem*, Combinatorica, 19 (1999), pp. 1–25.

[6] P. ELIAS, A. FEINSTEIN, AND C. E. SHANNON, *Note on maximum flow through a network*, IRE Trans. Information Theory, IT-2 (1956), pp. 117–199.

[7] L. R. FORD AND D. R. FULKERSON, *Maximal flow through a network*, Canad. J. Math., 8 (1956), pp. 399–404.

[8] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[9] A. FRANK, *Packing paths, cuts and circuits—a survey*, in Paths, Flows and VLSI-Layout, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds., Springer-Verlag, Berlin, 1990, pp. 49–100.

[10] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.

[11] N. GARG, V. VAZIRANI, AND M. YANNAKAKIS, *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica, 18 (1997), pp. 3–20.

[12] A. GOLDBERG AND S. RAO, *Beyond the flow decomposition barrier*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 2–11.

[13] A. V. GOLDBERG AND R. E. TARJAN, *A new approach to the maximum flow problem*, J. ACM, 35 (1988), pp. 921–940.

[14] A. V. Goldberg and R. E. Tarjan, *Solving minimum-cost flow problems by successive approximation*, Math. Oper. Res., 15 (1990), pp. 430–466.
[15] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: Theoretical and practical results*, J. ACM, 34 (1987), pp. 144–162.
[16] M. Iri, *A new method of solving transportation-network problems*, J. Oper. Res. Soc. Japan, 3 (1960), pp. 27–87.
[17] W. S. Jewell, *Optimal Flow through Networks*, Technical report 8, Operations Research Center, MIT, Cambridge, MA, 1958.
[18] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[19] V. King, S. Rao, and R. Tarjan, *A faster deterministic maximum flow algorithm*, J. Algorithms, 17 (1994), pp. 447–474.
[20] P. Klein, S. A. Plotkin, C. Stein, and E. Tardos, *Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts*, SIAM J. Comput., 23 (1994), pp. 466–487.
[21] P. Klein, S. Rao, A. Agrawal, and R. Ravi, *Approximation through multicommodity flow*, Combinatorica, 15 (1995), pp. 187–202.
[22] J. M. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, MIT, Cambridge, MA, 1996.
[23] J. M. Kleinberg, *Single-source unsplittable flow*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 68–77.
[24] J. M. Kleinberg and E. Tardos, *Approximations for the disjoint paths problem in high-diameter planar networks*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 26–35.
[25] J. M. Kleinberg and E. Tardos, *Disjoint paths in densely-embedded graphs*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 52–61.
[26] S. G. Kolliopoulos, *Exact and Approximation Algorithms for Network Flow and Disjoint-Path Problems*, Ph.D. thesis, Dartmouth College, Hanover, NH, 1998.
[27] S. G. Kolliopoulos and C. Stein, *Improved approximation algorithms for unsplittable flow problems*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 426–435.
[28] S. G. Kolliopoulos and C. Stein, *Approximating disjoint-path problems using greedy algorithms and packing integer programs*, in Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1412, R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, eds., Springer-Verlag, New York, 1998, pp. 153–168.
[29] S. G. Kolliopoulos and C. Stein, *Experimental evaluation of approximation algorithms for single-source unsplittable flow*, in Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1610, G. Cornuéjols, R. E. Burkard, and G. J. Woeginger, eds., Springer-Verlag, New York, 1999, pp. 328–344.
[30] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas, *Fast approximation algorithms for multicommodity flow problems*, J. Comput. System Sci., 50 (1995), pp. 228–243.
[31] T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, 1988, pp. 422–431.
[32] J. K. Lenstra, D. B. Shmoys, and E. Tardos, *Approximation algorithms for scheduling unrelated parallel machines*, Math. Program., 46 (1990), pp. 259–271.
[33] J. B. Orlin, *A faster strongly polynomial minimum cost flow algorithm*, Oper. Res., 41 (1993), pp. 338–350.
[34] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.
[35] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[36] N. Robertson and P. D. Seymour, *Outline of a disjoint paths algorithm*, in Paths, Flows and VLSI-Layout, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds., Springer-Verlag, Berlin, 1990, pp. 293–328.
[37] A. Schrijver, *Homotopic routing methods*, in Paths, Flows and VLSI-Layout, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds., Springer-Verlag, Berlin, 1990, pp. 329–371.
[38] D. B. Shmoys and E. Tardos, *An approximation algorithm for the generalized assignment problem*, Math. Program., 62 (1993), pp. 461–474.
[39] M. Skutella, *Approximating the single-source unsplittable min-cost flow problem*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2000.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 947-986

PROOF TECHNIQUES FOR CRYPTOGRAPHIC PROCESSES*

MICHELE BOREALE†, ROCCO DE NICOLA†, AND ROSARIO PUGLIESE†

Abstract. Contextual equivalences for cryptographic process calculi, like the spi-calculus, can be used to reason about correctness of protocols, but their definition suffers from quantification over all possible contexts. Here, we focus on two such equivalences, namely may-testing and barbed equivalence, and investigate tractable proof methods for them. To this aim, we design an enriched labelled transition system, where transitions are constrained by the knowledge the environment has of names and keys. The new transition system is then used to define a trace equivalence and a weak bisimulation equivalence that avoid quantification over contexts. Our main results are soundness and completeness of trace and weak bisimulation equivalence with respect to may-testing and barbed equivalence, respectively. They lead to more direct proof methods for equivalence checking. The use of these methods is illustrated with a few examples concerning implementation of secure channels and verification of protocol correctness.

Key words. process calculi, reasoning about security, semantics, formal methods

AMS subject classification. 68Q85

PII. S0097539700377864

1. Introduction.

Recently, there has been much interest in using formal methods for analyzing cryptographic protocols. Here, we focus on a specific approach, which aims at modeling protocols as concurrent processes, described as terms of a process calculus (e.g., the spi-calculus [7, 5], a cryptographic version of the π-calculus [17, 18]). As an example, consider the very simple protocol where two principals A and B share a private key k, and A wants to send B a datum d encrypted under k through a public channel c:

$$\text{Message 1} \qquad A \to B\colon \{d\}_k \text{ on } c.$$

This informal notation can be expressed in the spi-calculus as follows:

$$\begin{aligned} A(d) &\stackrel{\text{def}}{=} \overline{c}\{d\}_k.\mathbf{0}, \\ B &\stackrel{\text{def}}{=} c(x).F(x), \\ P(d) &\stackrel{\text{def}}{=} (\nu\, k)(A(d) \mid B). \end{aligned}$$

Here, $\overline{c}\{d\}_k.$ means *output* of message $\{d\}_k$ at channel c and $\mathbf{0}$ stands for *termination*. The prefix $c(x).$ indicates the intention to *input* a message at channel c and to bind it to x, and $F(x)$ is some expression describing the behavior of B after the reception of x. The whole protocol $P(d)$ is the *parallel composition* $A(d) \mid B$, with the *restriction* $(\nu\, k)$ indicating that the key k is known only to $A(d)$ and B.

The main advantage of this kind of description is that process calculi have formal yet simple semantics that permit rendering rigorously such notions as "attacker" and "secrecy." Continuing with the example above, a way of asserting that $P(d)$ keeps d secret is requiring that $P(d)$ be *equivalent* to $P(d')$ for every other d'. An appropriate notion of equivalence is *may-testing* [11, 8, 7]; its intuition is precisely that no external context (which in the present setting can be read as "attacker") may notice any

*Received by the editors September 11, 2000; accepted for publication (in revised form) March 5, 2001; published electronically February 20, 2002. This paper is an extended and revised version of [9].

http://www.siam.org/journals/sicomp/31-3/37786.html

†Dipartimento di Sistemi e Informatica, Università di Firenze, I-50134, Firenze, Italy (boreale@dsi.unifi.it, denicola@dsi.unifi.it, pugliese@dsi.unifi.it).

difference when running in parallel with $P(d')$ or $P(d)$. A similar intuition is supported by other contextual equivalences, like *barbed equivalence* [19]. While rigorous and intuitive, the definitions of these equivalences suffer from universal quantification over contexts (attackers) that makes equivalence checking very hard. It is then important to devise proof techniques that avoid such quantification. Results in this direction have been obtained for traditional process calculi (for example, in CCS [11, 15] and in π-calculus [17, 18], may-testing is easily proven to coincide with trace equivalence), but little has been done for cryptographic calculi.

In this paper, we consider may-testing and barbed equivalence for a variant of the spi-calculus with shared-key encryption primitives [7]. We develop an "environment-sensitive" labeled transition system (lts), whose transitions are constrained by the knowledge that the environment has of names and keys. A trace-based equivalence and a purely coinductive notion of weak bisimulation, which avoid quantification over contexts, are defined on the new lts, and it is shown that they are in agreement with may-testing and barbed equivalence, respectively. A more detailed account of our work follows.

The handling of names is a crucial aspect in the semantics of process calculi. Let us first consider the nature of the transitions in the noncryptographic π-calculus. There are three kinds of basic moves which correspond to output of a message, input of a message, and internal computation. An output transition like $P \xrightarrow{(\nu\, b)\overline{a}\langle b\rangle} P'$ says that process P passes a *new* (or *fresh*; $(\nu\,\cdot)$ means "new") name b to the environment along channel a and becomes P' in doing so. The environment can use at will names it has got to know. For example, the two-step sequence $P \xrightarrow{(\nu\, b)\overline{a}\langle b\rangle} P' \xrightarrow{b\, c} P''$ (where $b\,c$ means "input c along b") is possible. In general, if a process is ready to perform some action, the environment will always be able to react. The reason is that, in the π-calculus, environment and process share at each stage the *same* knowledge of names. Thus, to determine whether two processes are, say, may-testing equivalent, it is sufficient to establish that they can perform the *same* sequences of (input/output) actions.

The correspondence between environment and process knowledge is lost when moving to the spi-calculus, i.e., when adding encryption and decryption primitives to the π-calculus. Indeed, two new facts must be taken into account.

(a) When the environment receives a new name encrypted under a fresh key, it does not acquire the knowledge of that name immediately. For instance, after P outputs a new name b encrypted under a fresh key k (written $P \xrightarrow{(\nu\, b,k)\overline{a}\langle\{b\}_k\rangle} P'$), name b is part of the knowledge of P' but not part of the knowledge of the environment. Thus, if P' is willing to input something at b (say $P' = b(c).P''$), the environment cannot satisfy P''s expectations: a sequence like $P \xrightarrow{(\nu\, b,k)\overline{a}\langle\{b\}_k\rangle} P' \xrightarrow{b\, c} P''$ (that is possible in the traditional-style transition system) cannot be considered as meaningful in the spi-calculus. For similar reasons, a sequence like $P \xrightarrow{(\nu\, b,k)\overline{a}\langle\{b\}_k\rangle} P' \xrightarrow{a'\, b} P''$, where the environment is expected to send back the cleartext b, cannot be considered as meaningful.

(b) Equivalent processes need not exhibit the same sequences of transitions. The process that performs the single output $(\nu\, k)\overline{a}\langle\{b\}_k\rangle$ and terminates, and the one that performs $(\nu\, k)\overline{a}\langle\{c\}_k\rangle$ and terminates, are equivalent because messages $\{b\}_k$ and $\{c\}_k$ cannot be distinguished by the environment (as it

cannot open something encrypted with k). However, the two messages could be distinguished if the environment got the key k. Thus, the two processes $(\nu\, k)\overline{a}\{b\}_k.\,\overline{a}k.\,\mathbf{0}$ and $(\nu\, k)\overline{a}\{c\}_k.\,\overline{a}k.\,\mathbf{0}$ are not equivalent.

To cope with these issues and recover the correspondence between environment and process actions, we introduce an enriched lts that explicitly keeps track of the environment's knowledge. The states of the new lts are *configurations* $\sigma \rhd P$, where P is a process and σ is the current environment's knowledge, modeled as a mapping from a set of variables to a set of messages. Informally, σ plays the role of a database storing the messages received by the environment; each entry of the database is referenced by a distinct variable. Transitions represent interactions between the environment and the process and take the form

$$\sigma \rhd P \overset{\mu}{\underset{\delta}{\longmapsto}} \sigma' \rhd P',$$

where μ is the action of process P and δ is the "complementary" *environment action*.

We have three different kinds of situations.

1. The process performs an output and the environment an input. As a consequence, the environment's knowledge gets updated. For instance,

$$\sigma \rhd P \overset{(\nu\, \widetilde{b})\overline{a}\langle M\rangle}{\underset{z(x)}{\longmapsto}} \sigma[M/x] \rhd P'.$$

   Here $\sigma[M/x]$ is the update of σ with the new entry $[M/x]$ for a fresh variable x. Moreover, $\widetilde{b}$ is the set of new names the process has just created. For the transition to take place, channel a must belong to the knowledge of σ, which in this case amounts to saying that $\sigma(z) = a$.
2. The process performs an input and the environment an output. As discussed previously, messages from the environment cannot be arbitrary. They must be built, via encryption and decryption, using only the knowledge stored in σ. Thus, a transition might be

$$\sigma \rhd P \overset{a\, M}{\underset{(\nu\, \widetilde{b})\overline{z}\langle\zeta\rangle}{\longmapsto}} \sigma[\widetilde{b}/\widetilde{b}] \rhd P'.$$

   Informally, $\widetilde{b}$ is the set of new names the environment has just created and added to its knowledge, while ζ is an expression describing how M has been built out of σ and $\widetilde{b}$. For example, if $\sigma(x_1) = \{c\}_k$, $\sigma(x_2) = k$, and $M = c$, then ζ might be $\mathtt{dec}_{x_2}(x_1)$, indicating that message c results from decrypting the x_1-entry using the x_2-entry as a key. Again, a must belong to the knowledge of σ; thus $\sigma(z) = a$.
3. The process performs an internal move and the environment does nothing:

$$\sigma \rhd P \overset{\tau}{\underset{-}{\longmapsto}} \sigma \rhd P'.$$

When defining trace and bisimulation semantics (section 3) on the top of the new lts, the point of view is taken that *equivalent configurations should exhibit the same environment actions*. As an example, take σ with entries $\sigma(x) = a$, $\sigma(y) = b$, and $\sigma(z) = c$ and consider the configurations $C_1 \overset{\text{def}}{=} \sigma \rhd (\nu\, k)\overline{a}\{b\}_k.\,\mathbf{0}$ and $C_2 \overset{\text{def}}{=} \sigma \rhd (\nu\, k)\overline{a}\{c\}_k.\,\mathbf{0}$.

These configurations are both trace and bisimulation equivalent, because the only transitions they have are

$$C_1 \xmapsto[x(w)]{(\nu\, k)\overline{a}\langle\{b\}_k\rangle} \sigma[\{b\}_k/w] \rhd \mathbf{0}$$

and

$$C_2 \xmapsto[x(w)]{(\nu\, k)\overline{a}\langle\{c\}_k\rangle} \sigma[\{c\}_k/w] \rhd \mathbf{0},$$

which exhibit the same environment action, $x(w)$. On the other hand, as discussed above, $C_3 \stackrel{\text{def}}{=} \sigma \rhd (\nu\, k)\overline{a}\{b\}_k.\, \overline{a}k.\, \mathbf{0}$ and $C_4 \stackrel{\text{def}}{=} \sigma \rhd (\nu\, k)\overline{a}\{c\}_k.\, \overline{a}k.\, \mathbf{0}$ should not be regarded as equivalent. Indeed, after two steps, C_3 reaches a state where the environment is $\sigma[\{b\}_k/w][k/v]$, which cannot be considered "equivalent" to the environment reachable from C_4, i.e., $\sigma[\{c\}_k/w][k/v]$: The decryption of the entry w, which is now possible because k is known, yields distinct names, b and c, in the two cases. Equivalence on environments is a notion crucial to our approach and will be formalized in terms of logical equivalence. For both trace and bisimulation equivalence, we shall insist that *matching transitions should take equivalent environments to equivalent environments.* We shall show that these equivalences imply their contextual counterparts (soundness); hence the former can be used as proof techniques for the latter. The converse implication (completeness) is also proven for trace equivalence. As to bisimulation, we establish completeness relative to a broad class of processes (which includes all the *image-finite* ones [15]).

Trace and bisimulation equivalences avoid quantification over contexts and only require considering transitions of the enriched lts. As such, they make reasoning on processes much easier than the contextual definitions. While trace semantics are sufficient for expressing many security properties (especially those of secrecy and authenticity [7]), bisimulation is sometimes preferable because it embodies a notion of fairness and is supported by a nice, purely coinductive proof technique. The latter can be enhanced by tailoring, as we do, some "up to" techniques [22, 10] to the cryptographic setting. Another advantage of our semantics is the congruence rules that make compositional proofs possible. The use of trace and bisimulation semantics as proof techniques is illustrated with a few examples; some of them concern the problem of implementing secure channels using encrypted public channels (like in [4]). Some of the equalities we establish are hard and lengthy to prove if relying on the original, contextual definitions (see, e.g., the secure channel implementations in section 5).

The rest of the paper is organized as follows. The language is presented in section 2; there, we also introduce the contextual semantics: may-testing and barbed equivalences. Section 3 introduces the new lts for the spi-calculus and, based on that, trace and bisimulation semantics. In section 4, we establish soundness and completeness of trace semantics with respect to may-testing and of bisimulation with respect to barbed equivalence. Section 5 presents a number of properties of trace and bisimulation semantics and their applications. In section 6, we present the extension of the theory to a richer calculus, which permits handling pairs of messages. Comparisons to related works and a few concluding remarks are reported in section 7. The most technical proofs and definitions are relegated to appendices A, B, and C.

2. The language. We first present syntax and (conventional) operational semantics of the language, which is a variant of the spi-calculus. We then define the

contextual semantics: may-testing and barbed equivalence. In the definition of the language, there are a few implicit assumptions on the underlying system of shared-key cryptography. We make them explicit below:

1. A message M encrypted under a key k, written $\{M\}_k$, can only be decrypted using k. The only way to produce the ciphertext $\{M\}_k$ is to encrypt M under k. If k is secret, the attacker cannot guess or forge k (*perfect encryption*).
2. There is enough redundancy in the structure of messages to tell whether decryption of a message with a given key has actually succeeded or not.
3. There is enough redundancy in the structure of messages to tell their role (key or compound ciphertext).
4. The only way to form a new key is to get a fresh name from a primitive set of *names*.

These assumptions are quite common in the literature. In particular, the first two are also found in the original spi-calculus [7]. The third and fourth assumptions might be regarded as a limitation over the practice of crypto-protocols, where new keys are sometimes formed by assembling pieces of old messages together (especially if random bits are considered as an expensive resource). However, many interesting protocols do fulfill these assumptions.

2.1. Syntax. The syntax of the calculus is summarized in Figure 1. A countable set $\mathcal{N}$ of *names* $a, b \ldots, h, k, \ldots, x, y, z \ldots$ is assumed. Names can be used as communication channels, primitive data, or encryption keys: we do not distinguish between these three kinds of objects (in notation, we prefer letters $h, k, \ldots$ when we want to stress the use of a name as a key). In the standard π-calculus, names are the only transmissible objects. In the spi-calculus the possibility has been added to communicate *messages* obtained via shared-key encryption: message $\{M\}_k$ represents the ciphertext obtained by encrypting message M under key k, using a shared-key encryption system. Encryptions can be arbitrarily nested. *Expressions* are obtained by applying encryption and decryption operators to names and ciphertexts. For example, the result of evaluating $\mathtt{dec}_\eta(\zeta)$ is the text obtained by decrypting the ciphertext ζ using the value of η as a key. Expressions are also used to represent dummy terms that can be generated at run but do not represent proper messages (such as $\{a\}_{\{b\}_k}$, where a compound term $\{b\}_k$ is used as a key instead of an atomic name). Logical *formulae* generalize the usual equality operator of the π-calculus with a predicate $name(\cdot)$, which tests for the format of the argument (plain name or a compound ciphertext), and with a `let` construct that binds the value of some expression ζ to a name z. *Processes* are built using a set of operators which include those from the standard π-calculus, plus two new operators: boolean guard and a `let` construct. An informal explanation of the operators might be the following:

- $\mathbf{0}$ is the process that does nothing.
- $\eta(x).P$ represents input of a generic message x along η: the only useful case is when η is a name; otherwise the whole process is stuck.
- $\overline{\eta}\zeta.P$ represents output of ζ along η: the only useful case is when η is a name and ζ is a message; otherwise the whole process is stuck.
- $P + Q$ can behave either as P or Q: the choice might be triggered either by the environment, or by internal computations of P or Q.
- $P \mid Q$ is the parallel execution of P and Q.
- $(\nu\, a)P$ creates a new name a which is only known to P.
- $!\,P$ behaves like unboundedly many copies of P running in parallel, i.e., $P \mid P \mid P \mid \cdots$.

$a, b \ldots, h, k, \ldots, x, y, z \ldots$			*names* $\mathcal{N}$
M, N	::=	$a \mid \{M\}_k$	*messages* $\mathcal{M}$
η, ζ	::=	$a \mid \{\eta\}_\zeta \mid \mathtt{dec}_\eta(\zeta)$	*expressions* $\mathcal{Z}$
ϕ, ψ	::=	$\sharp \mid name(\zeta) \mid [\zeta = \eta] \mid \mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi \mid \phi \wedge \psi \mid \neg\phi$	*formulae* Φ
P, Q	::=		*processes* $\mathcal{P}$
		$\mathbf{0}$	(*null*)
	|	$\eta(x).\, P$	(*input prefix*)
	|	$\overline{\eta}\zeta.\, P$	(*output prefix*)
	|	$P + Q$	(*nondeterministic choice*)
	|	$P \mid Q$	(*parallel composition*)
	|	$(\nu\, a)P$	(*restriction*)
	|	$!\, P$	(*replication*)
	|	ϕP	(*boolean guard*)
	|	$\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$	(*encryption/decryption*)

It is assumed that $\mathtt{dec}_\cdot(\cdot)$ does not occur in $name(\zeta)$, $[\zeta = \eta]$, $\eta(x).$, and $\overline{\eta}\zeta.$. Operators $a(x).\cdot$, $(\nu\, a)\cdot$, and $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \cdot$ are *binders*, with the obvious scope, for names x, a, and z, respectively. In $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \cdot$, it is assumed that z does not appear in ζ.

FIG. 1. *Syntax of the calculus.*

- ϕP behaves like P if the formula ϕ is logically true; otherwise it is stuck.
- $\mathtt{let}\ z = \zeta\ \mathtt{in}\ P$ attempts evaluation of ζ: if the evaluation succeeds, the result is bound to z within P; otherwise the whole process is stuck.

There are a few differences from Abadi and Gordon's spi-calculus [7]. In particular, the differences are as follows:

- Our decryption keys cannot be compound messages, as already noted.
- For decryption, we use a "let" construct instead of the "case" construct of [7]. This enables us to write process expressions in a more compact form; for instance, the spi-calculus process $\mathtt{case}\ M\ \mathtt{of}\ \{k\}_h\ \mathtt{in}\ (\mathtt{case}\ N\ \mathtt{of}\ \{z\}_k\ \mathtt{in}\ P)$ can be written as $\mathtt{let}\ z = \mathtt{dec}_{\mathtt{dec}_h(M)}(N)\ \mathtt{in}\ P$ in our syntax.
- We have included a nondeterministic choice $+$, which is sometimes useful for specification purposes. Another, technical, reason for including $+$ in the language is that this operator appears to be necessary when proving coincidence of barbed equivalence and bisimilarity.
- We do not consider public key and hash functions, which are present in the original spi-calculus.

A minor difference is that the syntax above does not mention *tuples*, which we have preferred to treat separately (in section 6). This permits a cleaner presentation of the overall approach.

We shall often abbreviate $\alpha.\mathbf{0}$ as α, where α is an input or output prefix, and $(\nu\, a)(\nu\, b)P$ as $(\nu\, a, b)P$. We shall use the tilde $\widetilde{\cdot}$ to denote tuples of objects (e.g., $\widetilde{x}$ is a generic tuple of names); this will sometimes be written as $\widetilde{x}_{i \in I}$, for an appropriate index-set I. If $\widetilde{x} = (x_1, \ldots, x_n)$ and $\widetilde{y} = (y_1, \ldots, y_m)$, then $\widetilde{x}\widetilde{y}$ will denote the tuple

$(x_1, \ldots, x_n, y_1, \ldots, y_m)$. When convenient, we shall regard a tuple simply as a set (writing, e.g., $\widetilde{x} \subseteq S$ to mean that all components of $\widetilde{x}$ are in S). All our notations are extended to tuples componentwise. In particular, if $\widetilde{k} = (k_1, \ldots, k_n)$, then $\{M\}_{\widetilde{k}}$ means $\{\cdots\{M\}_{k_1}\cdots\}_{k_n}$ and $\mathtt{dec}_{\widetilde{k}}(M)$ means $\mathtt{dec}_{k_n}(\cdots\mathtt{dec}_{k_1}(M)\cdots)$.

Notions of *free names* of a process P, $fn(P)$, of *bound names* of P, $bn(P)$, and of alpha-equivalence arise as expected; $n(P)$ is $fn(P) \cup bn(P)$. Often, we shall write $fn(P, Q)$ in place of $fn(P) \cup fn(Q)$ (similarly for $bn(\cdot)$ and $n(\cdot)$). Similar notations are used for formulae, expressions, and messages.

A *substitution* σ is a finite partial map from $\mathcal{N}$ to the set of messages $\mathcal{M}$. The domain and proper codomain of σ are written $dom(\sigma)$ and $range(\sigma)$, respectively. We let $n(\sigma) = dom(\sigma) \cup (\cup_{M \in range(\sigma)} n(M))$. Given a tuple of distinct names $\widetilde{x} = (x_1, \ldots, x_n)$ and a tuple of messages $\widetilde{M} = (M_1, \ldots, M_n)$, the substitution mapping each x_i to M_i will be sometimes written as $[\widetilde{M}/\widetilde{x}]$ or $[{}^{M_i}/_{x_i}]_{i \in 1..n}$. When $\widetilde{x} \cap dom(\sigma) = \emptyset$, we write $\sigma[\widetilde{M}/\widetilde{x}]$ for the substitution σ', which is the union of σ and $[\widetilde{M}/\widetilde{x}]$ (in this case we say that σ' *extends* σ). For a given $V \subseteq_{\text{fin}} \mathcal{N}$, we write ϵ_V for the substitution with $dom(\epsilon_V) = V$ that acts as the identity on V. For any term (name/expression/formula/process) t, $t\sigma$ denotes the term obtained by simultaneously replacing each $x \in fn(t) \cap dom(\sigma)$ with $\sigma(x)$, with renaming of bound names of t possibly involved to avoid captures.

2.2. Operational semantics. The (conventional) operational semantics defined below only accounts for process intentions. In fact, the subsequent definitions of contextual equivalences use only the part of this semantics that describes "internal computation" (denoted by $\xrightarrow{\tau}$) of processes.

First, we need two evaluation functions: one for expressions, the other for formulae. The *evaluation function for expressions*, $\widehat{\cdot} : \mathcal{Z} \to \mathcal{M} \cup \{\bot\}$ (where $\bot$ is a distinct symbol) is defined by induction on ζ as follows:

- $\widehat{a} = a$,
- $\widehat{\{\zeta_1\}_{\zeta_2}} = \begin{cases} \{M\}_k & \text{if } \widehat{\zeta_1} = M \text{ and } \widehat{\zeta_2} = k \in \mathcal{N}, \text{ for some } M \text{ and } k, \\ \bot & \text{otherwise,} \end{cases}$
- $\widehat{\mathtt{dec}_{\zeta_2}(\zeta_1)} = \begin{cases} M & \text{if } \widehat{\zeta_1} = \{M\}_k \text{ and } \widehat{\zeta_2} = k \in \mathcal{N}, \text{ for some } M \text{ and } k, \\ \bot & \text{otherwise.} \end{cases}$

Note that the evaluation of an expression "fails," i.e., returns the value $\bot$, whenever an encryption/decryption with something different from a name is attempted, or whenever a decryption with something different from the encryption key is attempted. For instance, the evaluation of $\{a\}_{\{b\}_c}$ and $\mathtt{dec}_b(\{a\}_c)$ is $\bot$, while $\mathtt{dec}_c(\{a\}_c)$ evaluates to a.

The *evaluation function for formulae*, $[\![\cdot]\!] : \Phi \to \{tt, ff\}$, is defined by induction on ϕ. The only nonstandard clauses are those for $name(\zeta)$ and for $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi$:

- $[\![name(\zeta)]\!] = \begin{cases} tt & \text{if } \zeta \in \mathcal{N}, \\ ff & \text{otherwise,} \end{cases}$
- $[\![\mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi]\!] = \begin{cases} [\![\phi[\widehat{\zeta}/z]]\!] & \text{if } \widehat{\zeta} \neq \bot, \\ ff & \text{otherwise.} \end{cases}$

For any substitution σ, $\sigma \models \phi$ means that $[\![\phi\sigma]\!] = tt$.

The operational semantics is defined by the early-style inference rules of Figure 2. All rules but the last two are standard from π-calculus. Rule (GUARD) says that process ϕP behaves like P, provided that ϕ evaluates to true; otherwise, process ϕP is stuck. Rule (LET) attempts evaluation of expression ζ: if the evaluation succeeds,

(Inp) $a(x).\,P \xrightarrow{a\,M} P[M/x]$ (Out) $\overline{a}M.\,P \xrightarrow{\overline{a}\langle M\rangle} P$

(Sum) $\dfrac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}$ (Rep) $\dfrac{P \mid !\,P \xrightarrow{\mu} P'}{!\,P \xrightarrow{\mu} P'}$

(Par) $\dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}$ (Com) $\dfrac{P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle M\rangle} P' \qquad Q \xrightarrow{a\,M} Q'}{P \mid Q \xrightarrow{\tau} (\nu\,\widetilde{b})(P' \mid Q')}$

(Res) $\dfrac{P \xrightarrow{\mu} P'}{(\nu\,c)P \xrightarrow{\mu} (\nu\,c)P'}\ c \notin n(\mu)$ (Open) $\dfrac{P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle M\rangle} P'}{(\nu\,c)P \xrightarrow{(\nu\,\widetilde{b}c)\overline{a}\langle M\rangle} P'}\ c \neq a, c \in n(M) - \widetilde{b}$

(Guard) $\dfrac{[\![\,\phi\,]\!] = tt \qquad P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}$ (Let) $\dfrac{\widehat{\zeta} \neq \bot \qquad P[\widehat{\zeta}/z] \xrightarrow{\mu} P'}{\texttt{let}\ z = \zeta\ \texttt{in}\ P \xrightarrow{\mu} P'}$

FIG. 2. *Operational semantics (symmetric versions of* (Sum)*,* (Par)*, and* (Com) *omitted).*

then process $\texttt{let}\ z = \zeta\ \texttt{in}\ P$ behaves like process $P[\widehat{\zeta}/z]$; otherwise, $\texttt{let}\ z = \zeta\ \texttt{in}\ P$ is stuck.

Process *actions* (i.e., labels of the transition system), ranged over by $\mu, \lambda, \ldots$, can be of three forms: τ (internal action), $a\,M$ (input at a where message M is received), and $(\nu\,\widetilde{b})\overline{a}\langle M\rangle$ (output at a where message M containing the fresh, private names $\widetilde{b}$ is sent). We shall write $\overline{a}\langle M\rangle$ instead of $(\nu\,\widetilde{b})\overline{a}\langle M\rangle$ whenever $\widetilde{b} = \emptyset$. Input and output actions will be called *visible* actions. We use s to range over sequences of visible actions (traces), and write $\Longrightarrow$ or $\stackrel{\epsilon}{\Longrightarrow}$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$ and, inductively, $\stackrel{s}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\mu}\stackrel{s'}{\Longrightarrow}$ when $s = \mu \cdot s'$. $P \stackrel{s}{\Longrightarrow}$ will stand for "there is P' such that $P \stackrel{s}{\Longrightarrow} P'$ for some P."

From now on, we shall adopt the following convention.

Convention 2.1. We identify alpha-equivalent processes and formulae. Moreover, both in actions and in sequences of actions, *we shall assume that bound names can be freely renamed with fresh names*. In particular, we shall always assume that bound names are distinct from each other and from the free names and are not touched by substitutions.

2.3. May-testing and barbed equivalence. We instantiate the general framework of may-testing [11] to our calculus. *Observers*, ranged over by $O, O', \ldots$, are processes that can perform a distinct "success" action ω. Informally, the latter is used to signal that the observed process has passed a test. For instance, the observer $(\nu\,b)\overline{a}b.\,b(x).\,[x = c]\omega$, when run in parallel with any process, tests for the ability of the process to receive a new name b on channel a and then to send name c along b. The may-testing preorder is defined in terms of the ability of processes to pass tests proposed by observers. Since we work in a nondeterministic setting, a process *may* or *may not* pass a specific test. If one interprets "passing a test" as "revealing a piece of information," then two processes that may pass the same tests may potentially reveal the same information to observers. As such, they should be considered as equivalent from a security point of view. We can formalize this concept solely in terms of sequences of internal computations ($\Longrightarrow$) and success action (ω).

DEFINITION 2.2 (may-testing preorder). *$P \lesssim Q$ if, for every observer O, $P \mid O \stackrel{\omega}{\Longrightarrow}$ implies $Q \mid O \stackrel{\omega}{\Longrightarrow}$.*

The equivalence obtained as the kernel of the preorder $\lesssim$ is denoted by $\simeq$ ($\simeq = \lesssim \cap \lesssim^{-1}$) and is called *testing equivalence*.

The intuition behind barbed equivalence [19] is somehow similar to that of testing, but is based on a notion of step-by-step simulation between two processes. Throughout the rest of the paper, we say that a process P *commits to* a, and write $P \downarrow a$, if $P \xrightarrow{a\,M}$ or $P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle M\rangle}$ for some M and $\widetilde{b}$. We also write $P \Downarrow a$ if $P \Longrightarrow P'$ and $P' \downarrow a$ for some P'.

DEFINITION 2.3 (barbed equivalence). *A symmetric relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a* barbed bisimulation *if whenever $P\mathcal{S}Q$ holds true, then*

1. *for each P', if $P \xrightarrow{\tau} P'$, then there is Q' such that $Q \Longrightarrow Q'$ and $P'\mathcal{S}Q'$, and*
2. *for each a, if $P \downarrow a$, then $Q \Downarrow a$.*

Barbed bisimilarity*, written $\dot{\cong}$, is the largest barbed bisimulation relation. Two processes P and Q are* barbed equivalent*, written $P \cong Q$, if for all R we have that $P \mid R \dot{\cong} Q \mid R$.*

It is worthwhile to notice that neither $\lesssim$ nor $\cong$ are (pre)congruences, because they are not preserved by input prefix. This is standard in name-passing languages (see, e.g., [17, 18, 10]).

3. Trace and bisimulation semantics. In this section we shall first introduce an "environment-sensitive" lts for our calculus and then define trace and bisimulation equivalences over the new lts.

3.1. An environment-sensitive lts. States are *configurations* of the form $\sigma \rhd P$, where P is a process and where substitution σ represents the environment (from now on, terms "substitution" and "environment" will be used interchangeably). Transitions take the form

$$\sigma \rhd P \stackrel{\mu}{\underset{\delta}{\longmapsto}} \sigma' \rhd P'$$

and represent atomic interactions between process P and environment σ, μ is the process action (i.e., input, output, or τ), and δ is the complementary *environment action*. The latter can be of three forms, output, input and "no action":

$$\delta ::= (\nu\,\widetilde{b})\overline{\eta}\langle\zeta\rangle \quad | \quad \eta(x) \quad | \quad -\,.$$

The upper transition labels are not strictly necessary for the development of our theory. However, they are useful because they show the process action that triggers the transition.

Free names and bound names of δ are defined as expected, in particular $bn(\eta(x)) = \{x\}$. The *visible* environment actions are input and output. We shall use u to range over sequences of visible environment actions. The inference rules for the transition relation $\stackrel{\mu}{\underset{\delta}{\longmapsto}}$ are displayed in Figure 3. Note that judgements from the conventional transition system are used in the premises.

In rule (E-OUT), the environment receives a message M and updates its knowledge accordingly. For the transition to take place, channel a must belong to the knowledge of the environment; thus η is some expression describing how a can be obtained out of σ (this is what $\widehat{\eta\sigma} = a$ means). In rule (E-INP), the environment

It is assumed that $n(\eta) \subseteq dom(\sigma)$ and that names in $\widetilde{b}$ are fresh for σ and P.

$$\text{(E-Out)}\quad \frac{P \xrightarrow{(\nu\,\widetilde{b})\overline{a}\langle M\rangle} P' \qquad \widehat{\eta\sigma} = a}{\sigma \rhd P \overset{(\nu\,\widetilde{b})\overline{a}\langle M\rangle}{\underset{\eta(x)}{\longmapsto}} \sigma[M/x] \rhd P'} \qquad \text{(E-Tau)}\quad \frac{P \xrightarrow{\tau} P'}{\sigma \rhd P \overset{\tau}{\underset{-}{\longmapsto}} \sigma \rhd P'}$$

$$\text{(E-Inp)}\quad \frac{P \xrightarrow{a\,M} P' \qquad \widehat{\eta\sigma} = a \qquad M = \widehat{\zeta\sigma} \qquad \widetilde{b} \stackrel{\text{def}}{=} (n(\zeta) - dom(\sigma))}{\sigma \rhd P \overset{a\,M}{\underset{(\nu\,\widetilde{b})\overline{\eta}\langle\zeta\rangle}{\longmapsto}} \sigma[\widetilde{b}/\widetilde{b}] \rhd P'}$$

FIG. 3. *Rules for the environment-sensitive lts.*

sends a message M to the process. Expression ζ describes how message M is built out of σ and $\widetilde{b}$. The update $[\widetilde{b}/\widetilde{b}]$ records the creation of the new names $\widetilde{b}$.[1] As in the previous rule, a must belong to the knowledge of σ.

In the following, we write $\Longmapsto$ to denote the reflexive and transitive closure of $\overset{\tau}{\underset{-}{\longmapsto}}$ and, inductively, write $\overset{s}{\underset{u}{\Longmapsto}}$ for $\Longmapsto \overset{\mu}{\underset{\delta}{\longmapsto}} \overset{s'}{\underset{u'}{\Longmapsto}}$ when $s = \mu \cdot s'$ and $u = \delta \cdot u'$.

3.2. Trace and bisimulation semantics. In order to define observational semantics based on $\overset{\mu}{\underset{\delta}{\longmapsto}}$, we have to precisely define when two environments, represented by substitutions σ and σ', can be considered as equivalent. Informally, two environments σ and σ' are equivalent whenever they are logically indistinguishable.

DEFINITION 3.1 (equivalence on environments). *Two substitutions σ and σ' are* equivalent, *written $\sigma \sim \sigma'$, if $dom(\sigma) = dom(\sigma')$ and, for each formula ϕ with $fn(\phi) \subseteq dom(\sigma)$, it holds that $\sigma \models \phi$ if and only if $\sigma' \models \phi$.*

This logical characterization is difficult to check, as it contains a quantification on all formulae. Below, we shall give an equivalent definition that is easy to check. To do this, we start by making precise the concept of knowledge of an environment σ, that is, all the information that can be deduced from σ.

DEFINITION 3.2 (decryption closure and knowledge). *Let W be a set of messages. The* decryption closure *of W, written $dc(W)$, is the set of messages defined inductively as follows:*

(i) *$W \subseteq dc(W)$, and*
(ii) *if $k \in dc(W)$ and $\{M\}_k \in dc(W)$, then $M \in dc(W)$.*

The knowledge *of W, written $kn(W)$, is the set of names in $dc(W)$, i.e., $kn(W) \stackrel{\text{def}}{=} dc(W) \cap \mathcal{N}$. Let σ be a substitution; we let $dc(\sigma) \stackrel{\text{def}}{=} dc(range(\sigma))$ and $kn(\sigma) \stackrel{\text{def}}{=} kn(range(\sigma))$.*

Note that $kn(\sigma)$ can be computed in a finite number of steps. For instance, it is not difficult to see that $kn([\{b\}_l/w, \{a\}_k/x, \{k\}_h/y, h/z]) = \{a, h, k\}$. Next, we list some notational shorthand.

- Given any substitution $\sigma = [M_i/x_i]_{i \in I}$ and $i \in I$, we denote by $core(\sigma, x_i)$ what is left of M_i after decrypting as much as possible using the keys in $kn(\sigma)$. Formally, we define $core(\sigma, x_i)$ as the message N such that, for some $\widetilde{k} \subseteq kn(\sigma)$, it holds $M_i = \{N\}_{\widetilde{k}}$ and either N is a name, or $N = \{N'\}_h$

[1] Actually, any update $[\widetilde{b}/\widetilde{x}]$—with $\widetilde{x}$ a tuple of fresh names—could have been used instead.

for some N' and $h \notin kn(\sigma)$. For example, given $\sigma = [\{a\}_{hk}/x_1, k/x_2]$, then $core(\sigma, x_1) = \{a\}_h$ and $core(\sigma, x_2) = k$.

- Given a tuple $\widetilde{x} = x_{i \in I}$ and a tuple of indices $\widetilde{\jmath} = (j_1, \ldots, j_k) \subseteq I$, we let $\widetilde{x}[\widetilde{\jmath}]$ denote the tuple $(x_{j_1}, \ldots, x_{j_k})$. For instance, if $\widetilde{x} = (x_1, x_2, x_3)$ and $\widetilde{\jmath} = (3, 1, 1, 2)$, then $\widetilde{x}[\widetilde{\jmath}] = (x_3, x_1, x_1, x_2)$.

We are now set to give an alternative definition of equivalence on environments. The intuition behind the definition below is that, for any two equivalent environments, it should not be possible to tell apart two messages M_i and M'_i referenced by the same variable x_i by (a) trying decryption with different keys, (b) format mismatch (name vs. compound ciphertext), or (c) syntactic comparison.

DEFINITION 3.3 (equivalence on environments: alternative definition). *Let $\sigma = [M_i/x_i]_{i \in I}$ and $\sigma' = [M'_i/x_i]_{i \in I}$ be two substitutions with the same domain. For each $i \in I$, let $N_i = core(\sigma, x_i)$ and $N'_i = core(\sigma', x_i)$, and let $\widetilde{N} = N_{i \in I}$ and $\widetilde{N}' = N'_{i \in I}$. We write $\sigma \sim' \sigma'$ if for each $i \in I$ the following three conditions hold:*

(a) *for some tuple of indices $\widetilde{\jmath_i} \subseteq I$, it holds that $M_i = \{N_i\}_{\widetilde{N}[\widetilde{\jmath_i}]}$ and $M'_i = \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath_i}]}$;*

(b) *$N_i \in \mathcal{N}$ if and only if $N'_i \in \mathcal{N}$;*

(c) *for each $j \in I$, it holds that $N_i = N_j$ if and only if $N'_i = N'_j$.*

As an example, $\sigma_1 = [b/x_1, c/x_2, \{b\}_k/x_3]$ and $\sigma_2 = [b/x_1, c/x_2, \{c\}_k/x_3]$ are equivalent. On the contrary, $\sigma_3 = \sigma_1[k/x_4]$ and $\sigma_4 = \sigma_3[k/x_4]$ are not equivalent, because $core(\sigma_3, x_3) = b = core(\sigma_3, x_1)$, while $core(\sigma_4, x_3) = c \neq core(\sigma_4, x_1)$; thus condition (c) is violated.[2] Also note that equivalent environments need not have the same $kn(\cdot)$: for instance, the environments $[\{b\}_{kh}/x_1, h/x_2]$ and $[\{b\}_{k'h'}/x_1, h'/x_2]$ are equivalent, though they have different knowledge. Environment pairs of this sort may arise when comparing two processes (this is due to the interplay between encryption and restriction—see Example 3.9).

The following theorem, whose proof can be found in Appendix A, allows us to freely interchange the use of $\sim$ and of $\sim'$ in the rest of the paper.

THEOREM 3.4 (coincidence of $\sim$ and $\sim'$). *For any two substitutions σ and σ', it holds that $\sigma \sim \sigma'$ if and only if $\sigma \sim' \sigma'$.*

We are now ready to define a trace-based preorder. Recall that the bound names of s and u below are assumed to be fresh. A similar remark applies to μ and δ in Definition 3.8.

DEFINITION 3.5 (trace preorder). *Let $\sigma_1 \sim \sigma_2$. Given two processes P and Q, we write $(\sigma_1, \sigma_2) \vdash P \ll Q$ if, whenever $\sigma_1 \rhd P \stackrel{s}{\underset{u}{\Longrightarrow}} \sigma'_1 \rhd P'$, there are s', σ'_2, and Q' such that $\sigma_2 \rhd Q \stackrel{s'}{\underset{u}{\Longrightarrow}} \sigma'_2 \rhd Q'$ and $\sigma'_1 \sim \sigma'_2$.*

Note that, when comparing configurations, just the lower transition labels are considered, while the upper labels (in s and s') are ignored. We give them to help in reading the definition. We revise below the example given in the introduction.

EXAMPLE 3.6. *Define $\sigma = [a/x, b/y, c/z]$. Then $\sigma \rhd (\nu\, k)\overline{a}\{b\}_k$ and $\sigma \rhd (\nu\, k)\overline{a}\{c\}_k$ are $\ll$-equivalent. On the contrary, $\sigma \rhd (\nu\, k)\overline{a}\{b\}_k.\overline{a}k$ and $\sigma \rhd (\nu\, k)\overline{a}\{c\}_k.\overline{a}k$ are not*

[2] In general, once $kn(\sigma_1)$ and $kn(\sigma_2)$ have been computed, $\sigma_1 \sim' \sigma_2$ can be very easily checked. In particular, the existential on (a) does not imply any search among the tuples $\widetilde{\jmath}$. Given $i \in I$, we just choose any tuple $\widetilde{\jmath_i}$ s.t. $M_i = \{N_i\}_{\widetilde{N}[\widetilde{\jmath}_i]}$ and check whether $M'_i = \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath}_i]}$. If this is the case, then condition (a) is verified for i; otherwise, we can immediately conclude that $\sigma \not\sim' \sigma'$. Indeed, if condition (a) were validated by a different tuple $\widetilde{\jmath}'$, then we would get that $\widetilde{N}[\widetilde{\jmath_i}] = \widetilde{N}[\widetilde{\jmath}'_i]$, but $\widetilde{N}'[\widetilde{\jmath_i}] \neq \widetilde{N}'[\widetilde{\jmath}'_i]$; thus condition (c) would be violated.

related, because $\sigma[\{b\}_k/v,\, k/w] \not\sim \sigma[\{c\}_k/v,\, k/w]$ *for any fresh* v, w.

EXAMPLE 3.7. *A subtler example. Consider*

$$P \stackrel{\text{def}}{=} (\nu\, a, k)\overline{c}\{k\}_k.\overline{c}a \qquad \text{and} \qquad Q \stackrel{\text{def}}{=} (\nu\, a, k)\overline{c}\{k\}_{ak}.\overline{c}a.$$

It is easy to check that, for $\sigma \stackrel{\text{def}}{=} [c/c]$, *it holds both that* $(\sigma, \sigma) \vdash P \ll Q$ *and that* $(\sigma, \sigma) \vdash Q \ll P$. *The difference between* P *and* Q *is that* Q*'s first message contains a private name* (a) *that is later disclosed to the environment; however, the environment cannot detect this difference without the key* k, *which is never disclosed. Indeed,* P *and* Q *are may (and barbed) equivalent.*

More examples will be given in section 5. Let us switch to bisimulation. In what follows, $\sigma \rhd P \stackrel{\widehat{\mu}}{\stackrel{\Longrightarrow}{\scriptstyle\delta}} \sigma' \rhd P'$ stands for $\sigma \rhd P \stackrel{\mu}{\stackrel{\Longrightarrow}{\scriptstyle\delta}} \sigma' \rhd P'$ if $\mu \neq \tau$, and for $\sigma \rhd P \Longrightarrow \sigma' \rhd P'$ if $\mu = \tau$ (the $\widehat{\cdot}$ defined here has of course nothing to do with the evaluation function on expressions defined in the previous section). We say that a pair of configurations $(\sigma_1 \rhd P, \sigma_2 \rhd Q)$ is *compatible* if σ_1 and σ_2 are equivalent. A relation $\mathcal{R}$ is *compatible* if it only contains compatible pairs of configurations. Given a binary relation $\mathcal{R}$, we write $(\sigma_1, \sigma_2) \vdash P\,\mathcal{R}\,Q$ if $(\sigma_1 \rhd P\,,\, \sigma_2 \rhd Q) \in \mathcal{R}$.

DEFINITION 3.8 (weak bisimulation). *Let* $\mathcal{R}$ *be a binary compatible relation of configurations. We say that* $\mathcal{R}$ *is a* weak bisimulation *if, whenever* $(\sigma_1, \sigma_2) \vdash P\,\mathcal{R}\,Q$ *and* $\sigma_1 \rhd P \stackrel{\mu}{\stackrel{\longmapsto}{\scriptstyle\delta}} \sigma_1' \rhd P'$, *there are* μ', σ_2', *and* Q' *such that* $\sigma_2 \rhd Q \stackrel{\widehat{\mu'}}{\stackrel{\Longrightarrow}{\scriptstyle\delta}} \sigma_2' \rhd Q'$ *and* $(\sigma_1', \sigma_2') \vdash P'\,\mathcal{R}\,Q'$, *and the converse on the transitions of* Q *and* P. Bisimilarity, *written* $\approx$, *is the largest weak bisimulation relation.*

EXAMPLE 3.9. *This example shows that, when establishing process equivalence, pairs of equivalent environments with different* $kn(\cdot)$ *may arise, even when starting from a pair of identical environments. Consider*

$$P \stackrel{\text{def}}{=} (\nu\, a, k)\overline{c}\{k\}_k.\overline{c}a.\overline{a}c \qquad \text{and} \qquad Q \stackrel{\text{def}}{=} (\nu\, a, k)\overline{c}\{k\}_{ak}.\overline{c}a.\overline{a}c,$$

and let $\sigma \stackrel{\text{def}}{=} [c/c]$. *It is easy to see that* $(\sigma, \sigma) \vdash P \approx Q$. *In particular, the move*

$$\sigma \rhd P \stackrel{(\nu\, k)\overline{c}\langle\{k\}_k\rangle}{\stackrel{\longmapsto}{\scriptstyle c(x)}} \sigma[\{k\}_k/x] \rhd (\nu\, a)\overline{c}a.\overline{a}c \stackrel{\text{def}}{=} P'$$

is matched by

$$\sigma \rhd Q \stackrel{(\nu\, k,a)\overline{c}\langle\{k\}_{ak}\rangle}{\stackrel{\longmapsto}{\scriptstyle c(x)}} \sigma[\{k\}_{ak}/x] \rhd \overline{c}a.\overline{a}c \stackrel{\text{def}}{=} Q',$$

where $(\sigma[\{k\}_k/x], \sigma[\{k\}_{ak}/x]) \vdash P' \approx Q'$. *Thus, the move*

$$\sigma[\{k\}_k/x] \rhd P' \stackrel{(\nu\, b)\overline{c}\langle b\rangle}{\stackrel{\longmapsto}{\scriptstyle c(y)}} \sigma[\{k\}_k/x,\, b/y] \rhd \overline{b}c \stackrel{\text{def}}{=} P''$$

(where we have alpha-renamed a *into a fresh* b*) must be matched by*

$$\sigma[\{k\}_{ak}/x] \rhd Q' \stackrel{\overline{c}\langle a\rangle}{\stackrel{\longmapsto}{\scriptstyle c(y)}} \sigma[\{k\}_{ak}/x,\, a/y] \rhd \overline{a}c \stackrel{\text{def}}{=} Q'',$$

where $(\sigma[\{k\}_k/x,\, b/y], \sigma[\{k\}_{ak}/x,\, a/y]) \vdash P'' \approx Q''$. *In particular, the process action* $\overline{b}\langle c\rangle$ *originating from* P'' *is matched by the process action* $\overline{a}\langle c\rangle$ *originating from* Q''.

To end the section, we note that bisimilarity $\approx$ is strictly included in the trace preorder $\ll$. This fact holds for labelled transition systems in general; it is not specific to cryptography. As an example, it is easily checked that if $V = \{a, b, c\}$, then

$$(\epsilon_V, \epsilon_V) \vdash P \stackrel{\text{def}}{=} \overline{a}a.(\nu\, z)((\overline{z}z.\overline{b}b \,|\, \overline{z}z.\overline{c}c) \,|\, z(x)) \ll (\nu\, z)((\overline{z}z.\overline{a}a.\overline{b}b \,|\, \overline{z}z.\overline{a}a.\overline{c}c) \,|\, z(x)) \stackrel{\text{def}}{=} Q$$

but

$$(\epsilon_V, \epsilon_V) \vdash P \not\approx Q.$$

Indeed, $\epsilon_V \rhd P$ has an $\overline{a}a$-move that $\epsilon_V \rhd Q$ cannot match.

4. Soundness and completeness. In this section we show the agreement between the contextual semantics of section 2 and the semantics based on the environment-sensitive lts of section 3. More precisely, we will prove that

- the trace preorder ($\ll$) coincides with the may-testing preorder ($\sqsubseteq_{\approx}$), and
- bisimilarity ($\approx$) is included in barbed equivalence ($\cong$), while the opposite inclusions hold for the class of structurally image-finite processes (defined later in this section).

The inclusions $\ll \subseteq$, $\sqsubseteq_{\approx}$, and $\approx \subseteq \cong$ will be referred to as *soundness*, while the opposite inclusions will be referred to as *completeness*. There are a few basic ingredients for the proofs of soundness and completeness, which we list below. First, it is technically convenient to introduce a notion of structural equivalence, $\equiv$, in the same vein of [16].

DEFINITION 4.1 (structural equivalence). Structural equivalence *is the least equivalence relation $\equiv$ over processes that is preserved by parallel composition and restriction and satisfies the structural laws of* [16], *i.e.,*

- the monoid laws for parallel composition*:* $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, *and* $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$,
- the laws for restriction*:* $(\nu\, b)\mathbf{0} \equiv \mathbf{0}$, $(\nu\, a)(\nu\, b)P \equiv (\nu\, b)(\nu\, a)P$, *and* $(\nu\, a)(P|Q) \equiv P \mid (\nu\, a)Q$ *if* $a \notin fn(P)$,
- the law for replication*:* $!P \equiv P \mid !P$,

plus the law

$$(\texttt{let}\ z = \zeta\ \texttt{in}\ P) \equiv P[\widehat{\zeta}/z] \quad \text{if } \widehat{\zeta} \neq \perp.$$

A property of structural equivalence that we shall use extensively in what follows is that $\equiv$ commutes with $\xrightarrow{\mu}$; i.e., if $P \equiv Q$ and $P \xrightarrow{\mu} P'$, then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \equiv Q'$ (the proof goes by inspection of the rules; see also [16]).

The key to soundness is the following proposition that relates equivalence on environments ($\sim$) to the (conventional) operational semantics of Figure 2 (its proof can be found in Appendix B).

PROPOSITION 4.2. *Consider two equivalent substitutions, σ_1 and σ_2. Let R be any process or observer such that $fn(R) \subseteq dom(\sigma_1)$.*

(1) *Suppose that $R\sigma_1 \xrightarrow{(\nu\, b)\overline{a}\langle M\rangle} R_1$. Then* (i) *there is η such that (s.t.) $n(\eta) \subseteq dom(\sigma_1)$ and $\widehat{\eta\sigma_1} = a$;* (ii) *there are ζ and R' s.t. $fn(\zeta, R') \subseteq dom(\sigma_1) \cup \widetilde{b}$, $M = \widehat{\zeta\sigma_1}$, and $R_1 \equiv R'\sigma_1$;* (iii) *it holds that $R\sigma_2 \xrightarrow{(\nu\, \widetilde{b})\overline{a'}\langle M'\rangle} R_2$, where $a' = \widehat{\eta\sigma_2}$, $M' = \widehat{\zeta\sigma_2}$, and $R_2 \equiv R'\sigma_2$.*

(2) *Suppose that* $R\sigma_1 \xrightarrow{a\,M} R_1$. *Then* (i) *there is* η *s.t.* $n(\eta) \subseteq dom(\sigma_1)$ *and* $\widehat{\eta\sigma_1} = a$; (ii) *given any fresh* y *and* $\sigma_1' \stackrel{\text{def}}{=} \sigma_1[M/y]$, *there is* R' *s.t.* $fn(R') \subseteq dom(\sigma_1')$ *and* $R_1 \equiv R'\sigma_1'$; (iii) *for any* M', *it holds that* $R\sigma_2 \xrightarrow{a'\,M'} R_2$, *where* $a' = \widehat{\eta\sigma_2}$, $R_2 \equiv R'\sigma_2'$, *and* $\sigma_2' \stackrel{\text{def}}{=} \sigma_2[M'/y]$.

(3) *Suppose that* $R\sigma_1 \xrightarrow{\mu} R_1$ *with* $\mu = \tau$ *or* $\mu = \omega$. *Then* (i) *there is* R' *s.t.* $fn(R') \subseteq dom(\sigma_1)$ *and* $R_1 \equiv R'\sigma_1$; (ii) *we have* $R\sigma_2 \xrightarrow{\mu} R_2$, *where* $R_2 \equiv R'\sigma_2$.

The main ingredient for completeness is the notion of a *characteristic formula* of an environment σ, written ϕ_σ. The exact definition of ϕ_σ can be found in Appendix A. Here, we only wish to recall the properties that will be used below. It holds that $n(\phi_\sigma) \subseteq dom(\sigma)$ and that $\sigma \models \phi_\sigma$; moreover, the following crucial theorem (whose proof can be found in Appendix A) says that ϕ_σ characterizes all and only those environments σ' equivalent to σ.

THEOREM 4.3. *Let* σ *and* σ' *be two substitutions such that* $dom(\sigma) = dom(\sigma')$. *We have that* $\sigma \sim \sigma'$ *if and only if* $\sigma' \models \phi_\sigma$.

We will now proceed to prove soundness and completeness, first for may-testing and then for barbed equivalence.

4.1. May-testing and trace semantics.

Soundness. It is convenient to prove soundness of $\ll$ with respect to a notion more general than $\sqsubseteq$, which is introduced below.

DEFINITION 4.4 (generalized may-testing). *For equivalent* σ_1 *and* σ_2, *we write* $(\sigma_1, \sigma_2) \vdash P \sqsubseteq Q$ *if, for each observer* O *with* $fn(O) \subseteq dom(\sigma_1)$, $P \mid O\sigma_1 \stackrel{\omega}{\Longrightarrow}$ *implies* $Q \mid O\sigma_2 \stackrel{\omega}{\Longrightarrow}$.

The above definition subsumes that of may preorder, as $P \sqsubseteq Q$ holds if and only if $(\epsilon_V, \epsilon_V) \vdash P \sqsubseteq Q$ for some $V \supseteq fn(P, Q)$. The "only if" part of this statement is trivial. To see that the "if" part is true, use the fact that, for any R and O, $R \mid O \stackrel{\omega}{\Longrightarrow}$ if and only if $(\nu\,\widetilde{b})(R \mid O) \equiv R \mid (\nu\,\widetilde{b})O \stackrel{\omega}{\Longrightarrow}$, where $\widetilde{b} = fn(O) - fn(R)$.

We need some properties of sequences of transitions. To state these properties we introduce some additional notions.

NOTATION 4.5. *Given two process actions* μ *and* λ, *we write* $\mu \underline{compl}\, \lambda$ *when* $\mu = a\,M$ *and* $\lambda = (\nu\,\widetilde{b})\overline{a}\langle M\rangle$, *or vice versa, for some* a, $\widetilde{b}$, *and* M. *The notation extends to sequences of visible actions of the same length (*$s \underline{compl}\, r$*) as expected.*

Note that whenever $P \mid O \stackrel{\omega}{\Longrightarrow}$, we can find s and r (possibly empty) s.t. $P \stackrel{s}{\Longrightarrow}$, $O \stackrel{r\omega}{\Longrightarrow}$, and $s \underline{compl}\, r$. Conversely, $P \stackrel{s}{\Longrightarrow}$ and $O \stackrel{r\omega}{\Longrightarrow}$ with $s \underline{compl}\, r$ can be composed to yield $P \mid O \stackrel{\omega}{\Longrightarrow}$.

DEFINITION 4.6. *Given an environment action* δ *and an environment* σ, $\widehat{\delta\sigma}$ *is the process action defined as*

$$\widehat{\delta\sigma} = \begin{cases} (\nu\,\widetilde{b})\overline{a}\langle M\rangle & \text{if } \delta = (\nu\,\widetilde{b})\overline{\eta}\langle\zeta\rangle,\ \widehat{\eta\sigma} = a, \text{ and } M = \widehat{\zeta\sigma}, \\ a\,M & \text{if } \delta = \eta(x),\ \widehat{\eta\sigma} = a, \text{ and } M = x\sigma, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

Given a trace u *with* $n(u) \subseteq dom(\sigma)$, $\widehat{u\sigma}$ *is defined as expected.*

For proving the soundness theorem we make use of the following three auxiliary lemmas whose proofs can be found in Appendix B. The first lemma is a generalization of Proposition 4.2 to sequences of transitions.

LEMMA 4.7. *Suppose $\sigma_1 \sim \sigma_2$ and let O be any observer such that $fn(O) \subseteq dom(\sigma_1)$. Suppose that $O\sigma_1 \stackrel{r}{\Longrightarrow} O_1$, where, for some u and σ_1' extending σ_1, it is $r = \widehat{u\sigma_1'}$. Then, there is O' with $fn(O') \subseteq dom(\sigma_1')$ such that $O_1 \equiv O'\sigma_1'$. Furthermore, for any σ_2' extending σ_2 and such that $\sigma_2' \sim \sigma_1'$, it holds that $O\sigma_2 \stackrel{r'}{\Longrightarrow} \equiv O'\sigma_2'$, with $r' = \widehat{u\sigma_2'}$.*

The next lemma gives a sufficient condition to infer the existence of a sequence of transitions $\sigma \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma' \rhd P'$.

LEMMA 4.8. *Consider σ, P, and any observer O with $fn(O) \subseteq dom(\sigma)$. Suppose that $P \stackrel{s}{\Longrightarrow} P'$ and that $O\sigma \stackrel{r}{\Longrightarrow}$ with $s \underline{compl}\, r$. Then there are u and σ' extending σ such that $r = \widehat{u\sigma'}$ and $\sigma \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma' \rhd P'$.*

Finally, a simple result relates the form of s to that of u in a sequence of transitions $\stackrel{s}{\underset{u}{\models\!\Longrightarrow}}$.

LEMMA 4.9. *Suppose that $\sigma \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma' \rhd P'$. Then it holds that $P \stackrel{s}{\Longrightarrow} P'$, that σ' extends σ, and that $s \underline{compl}\, r$, where $r = \widehat{u\sigma'}$.*

We can now state and prove the soundness theorem.

THEOREM 4.10 (soundness of trace semantics). *If $(\sigma_1, \sigma_2) \vdash P \ll Q$, then $(\sigma_1, \sigma_2) \vdash P \stackrel{\cdot}{\lesssim} Q$.*

Proof. Suppose that $(\sigma_1, \sigma_2) \vdash P \ll Q$ and let O be any observer with $fn(O) \subseteq dom(\sigma_1)$. Suppose that $P \mid O\sigma_1 \stackrel{\omega}{\Longrightarrow}$; we have to show that $Q \mid O\sigma_2 \stackrel{\omega}{\Longrightarrow}$. Since $P \mid O\sigma_1 \stackrel{\omega}{\Longrightarrow}$ we can find P', s, and r such that

$$P \stackrel{s}{\Longrightarrow} P' \text{ and } O\sigma_1 \stackrel{r\omega}{\Longrightarrow} \text{ with } s \underline{compl}\, r.$$

(As usual, we suppose that $bn(s, r)$ are fresh.) Due to Lemma 4.8, we can find u and σ_1' extending σ_1 such that

$$\sigma_1 \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma_1' \rhd P' \text{ and } r = \widehat{u\sigma_1'}\,.$$

Thus, by hypothesis, there are s', σ_2' extending σ_2 and Q' such that

$$\sigma_2 \rhd Q \stackrel{s'}{\underset{u}{\models\!\Longrightarrow}} \sigma_2' \rhd Q' \text{ with } \sigma_1' \sim \sigma_2'\,.$$

Moreover, due to Lemma 4.9, it holds that $Q \stackrel{s'}{\Longrightarrow} Q'$ and $s' \underline{compl}\, r'$, where $r' = \widehat{u\sigma_2'}$. From Lemma 4.7 applied to $O\sigma_1 \stackrel{r\omega}{\Longrightarrow}$ and to σ_2', we get that $O\sigma_2 \stackrel{r'\omega}{\Longrightarrow}$. From this fact, $Q \stackrel{s'}{\Longrightarrow} Q'$, and $s' \underline{compl}\, r'$, we get the wanted $Q \mid O\sigma_2 \stackrel{\omega}{\Longrightarrow}$. □

Completeness. We begin by introducing some notation. We write $\langle\overline{\eta}\zeta\rangle.P$ instead of $\texttt{let}\ z_1 = \eta\ \texttt{in}\ (\texttt{let}\ z_2 = \zeta\ \texttt{in}\ \overline{z_1}z_2.P)$ and $\langle\eta(x)\rangle.P$ instead of $\texttt{let}\ z = \eta\ \texttt{in}\ z(x).P$ (z, z_1, and z_2 fresh). The *output-bound names* of δ, $obn(\delta)$, are defined as follows: $obn((\nu\,\widetilde{b})\overline{\eta}\zeta) = \widetilde{b}$ and $obn(\delta) = \emptyset$ if δ is not an output action. This notation extends to traces ($obn(u)$) as expected. Given u and σ, we say that u *is consistent with* σ if (i) $n(u) \subseteq dom(\sigma)$; (ii) σ extends $[\widetilde{b}/\widetilde{b}]$, where $\widetilde{b} = obn(u)$; and (iii) $\widehat{u\sigma}$ is defined.

Based on ϕ_σ, we can define a class of canonical observers $o(u, \sigma)$, depending on specific u and σ.

DEFINITION 4.11 (canonical observers). *Consider u consistent with σ. The observers $o(u,\sigma)$ are defined by induction on u as follows:*

$$\begin{array}{rcl} o(\epsilon,\sigma) & \stackrel{\text{def}}{=} & \phi_\sigma \omega, \\ o((\nu\, \widetilde{b})\overline{\eta}\langle\zeta\rangle \cdot u, \sigma) & \stackrel{\text{def}}{=} & (\nu\, \widetilde{b})\, \langle\overline{\eta}\zeta\rangle.o(u,\sigma), \\ o(\eta(x)\cdot u,\sigma) & \stackrel{\text{def}}{=} & \langle\eta(x)\rangle.o(u,\sigma). \end{array}$$

Note that $fn(o(u,\sigma)) \subseteq dom(\sigma) - bn(u)$. We now need two technical lemmas.

LEMMA 4.12. *Consider u consistent with σ. Let σ_0 be the restriction of σ to $dom(\sigma) - bn(u)$.*

1. *We have $o(u,\sigma)\sigma_0 \stackrel{r\omega}{\Longrightarrow}$, where $r = \widehat{u\sigma}$.*
2. *Consider any $\sigma_0' \sim \sigma_0$. If $o(u,\sigma)\sigma_0' \stackrel{r\omega}{\Longrightarrow}$, then (up to the renaming of bound names of r) $r = \widehat{u\sigma'}$ for some σ' extending σ_0' and s.t. $\sigma \sim \sigma'$.*

Proof. Both parts are proven by straightforward induction on u and relying on Theorem 4.3 for the base case. □

LEMMA 4.13. *Consider P, σ, and σ' extending σ. Suppose that $P \stackrel{s}{\Longrightarrow} P'$, with $s \underline{\ compl\ } \widehat{u\sigma'}$, for some u consistent with σ' and s.t. $bn(u) = dom(\sigma') - dom(\sigma)$. Then $\sigma \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma' \rhd P'$.*

Proof. The proof may be made by straightforward induction based on u, which relies on Proposition 4.2. □

We are now ready to prove completeness.

THEOREM 4.14 (completeness for may-testing). *If $(\sigma_1,\sigma_2) \vdash P \sqsubseteq Q$, then $(\sigma_1,\sigma_2) \vdash P \ll Q$.*

Proof. Suppose that $(\sigma_1,\sigma_2) \vdash P \sqsubseteq Q$ and that $\sigma_1 \rhd P \stackrel{s}{\underset{u}{\models\!\Longrightarrow}} \sigma_1' \rhd P'$, where as usual $bn(s,u)$ are taken fresh. We have to show that for some s', σ_2', and Q' it holds that $\sigma_2 \rhd Q \stackrel{s'}{\underset{u}{\models\!\Longrightarrow}} \sigma_2' \rhd Q'$ and $\sigma_1' \sim \sigma_2'$.

Due to Lemma 4.9, we know that $s \underline{\ compl\ } r \stackrel{\text{def}}{=} \widehat{u\sigma_1'}$. Furthermore, it is easy to show that u is consistent with σ_1'. Since σ_1 is the restriction of σ_1' to $dom(\sigma_1') - bn(u)$, by virtue of Lemma 4.12(1) we get that $o(u,\sigma_1')\sigma_1 \stackrel{r\omega}{\Longrightarrow}$. From this, $P \stackrel{s}{\Longrightarrow} P'$, and $s \underline{\ compl\ } r$, we get that $P \mid o(u,\sigma_1')\sigma_1 \stackrel{\omega}{\Longrightarrow}$. Thus, by hypothesis, we also have $Q \mid o(u,\sigma_1')\sigma_2 \stackrel{\omega}{\Longrightarrow}$. This implies that there are s', Q', and r' such that

$$Q \stackrel{s'}{\Longrightarrow} Q' \text{ and } o(u,\sigma_1')\sigma_2 \stackrel{r'\omega}{\Longrightarrow} \text{ with } s' \underline{\ compl\ } r' .$$

From this and Lemma 4.12(2), we obtain that $r' = \widehat{u\sigma_2'}$, for some σ_2' extending σ_2 and s.t. $\sigma_1' \sim \sigma_2'$. From this fact, $Q \stackrel{s'}{\Longrightarrow} Q'$, and Lemma 4.13, we get the desired $\sigma_2 \rhd Q \stackrel{s'}{\underset{u}{\models\!\Longrightarrow}} \sigma_2' \rhd Q'$. □

4.2. Barbed equivalence and bisimilarity.

Soundness. It is convenient to generalize Definition 2.3 of barbed equivalence.

DEFINITION 4.15 (generalized barbed equivalence). *Let $\sigma_1 = [M_i/x_i]_{i\in I}$ and $\sigma_2 = [M_i'/x_i]_{i\in I}$ be equivalent substitutions. For each $i \in I$, let $N_i = core(\sigma_1, x_i)$ and $N_i' = core(\sigma_2, x_i)$. A binary relation $\mathcal{S}$ of processes is a (σ_1,σ_2)-*barbed bisimulation *if, whenever $P\mathcal{S}Q$ holds true,*

1. *for each P', if $P \stackrel{\tau}{\longrightarrow} P'$, then there is Q' such that $Q \Longrightarrow Q'$ and $P'\mathcal{S}Q'$, and*

2. *for each* $i \in I$, *if* $P \downarrow N_i$, *then* $Q \Downarrow N'_i$

and the converse on the transitions and commitments of Q *and* P.

Two processes P *and* Q *are* (σ_1, σ_2)-barbed bisimilar, *written* $(\sigma_1, \sigma_2) \vdash P \mathrel{\dot{\cong}} Q$, *if* (P, Q) *belongs to the largest* (σ_1, σ_2)*-barbed bisimulation.*

Two processes P *and* Q *are* (σ_1, σ_2)-barbed equivalent, *written* $(\sigma_1, \sigma_2) \vdash P \cong Q$, *if for all* R *with* $fn(R) \subseteq dom(\sigma_1)$ *we have that* $(\sigma_1, \sigma_2) \vdash P \mid R\sigma_1 \mathrel{\dot{\cong}} Q \mid R\sigma_2$.

Note the differences of this definition from Definition 2.3. First, the "barbs" (the $\downarrow$ predicate) are only checked relative to those names that are known to the environments, which are precisely those N_i's and those N'_i's that are in $\mathcal{N}$. Second, for each $i \in I$, names N_i and N'_i are not required to be the same, but are just required to be the "cores" of the same environment entry x_i. Most important, $\mathrel{\dot{\cong}}$ is just closed under those contexts that can be obtained via instantiation with σ_1 and σ_2. Of course, $P \cong Q$ holds if and only if $(\epsilon_V, \epsilon_V) \vdash P \cong Q$ for some V containing $fn(P, Q)$.

The purely coinductive formulation of $\approx$ given in Definition 3.8 gives us a powerful proof technique when proving equalities between two processes: it is sufficient to exhibit *any* bisimulation relation containing the given pair. This technique can be enhanced using the so-called *up to* techniques (similar to those in, e.g., [22, 10]), which often permit one to reduce the size of the relation to exhibit. We introduce below some useful up to techniques, which will be used in later proofs and examples. *Up to structural equivalence* allows one to freely identify structurally equivalent processes; *up to weakening* permits discarding environment entries, while *up to contraction* permits adding redundant (hence harmless) entries to the environments. Finally, *up to restriction* and *up to parallel composition* permit cutting away top-level restrictions and common parallel contexts, respectively, in process derivatives.

DEFINITION 4.16 (up to techniques). *Given a compatible relation* $\mathcal{R}$, *define* $\mathcal{R}_t$, *for* $t \in \{s, w, c, r, p\}$, *as the least binary relations over configurations that satisfy the following rules:*

- *up to structural equivalence:*
$$\frac{P' \equiv P,\ Q' \equiv Q \text{ and } (\sigma_1, \sigma_2) \vdash P' \,\mathcal{R}\, Q'}{(\sigma_1, \sigma_2) \vdash P \,\mathcal{R}_s\, Q};$$

- *up to weakening:*
$$\frac{(\sigma_1[\widetilde{M}/\widetilde{x}], \sigma_2[\widetilde{M'}/\widetilde{x}]) \vdash P \,\mathcal{R}\, Q}{(\sigma_1, \sigma_2) \vdash P \,\mathcal{R}_w\, Q};$$

- *up to contraction:*
$$\frac{(\sigma_1, \sigma_2) \vdash P \,\mathcal{R}\, Q,\ \perp \notin \widetilde{\widetilde{\zeta}\sigma_1} \text{ and } n(\widetilde{\zeta}) - dom(\sigma_1) \text{ are fresh for } \sigma_1,\ \sigma_2,\ P, \text{ and } Q}{(\sigma_1[\widetilde{\widetilde{\zeta}\sigma_1}/\widetilde{y}], \sigma_2[\widetilde{\widetilde{\zeta}\sigma_2}/\widetilde{y}]) \vdash P \,\mathcal{R}_c\, Q};$$

- *up to restriction:*
$$\frac{\widetilde{h} \cap n(\sigma_1) = \emptyset\ ,\ \widetilde{k} \cap n(\sigma_2) = \emptyset \text{ and } (\sigma_1, \sigma_2) \vdash P \,\mathcal{R}\, Q}{(\sigma_1, \sigma_2) \vdash (\nu\, \widetilde{h}) P \,\mathcal{R}_r\, (\nu\, \widetilde{k}) Q};$$

- *up to parallel composition:*
$$\frac{A \equiv P \mid R\sigma_1,\ B \equiv Q \mid R\sigma_2 \text{ and } (\sigma_1, \sigma_2) \vdash P \,\mathcal{R}\, Q \text{ and } fn(R) \subseteq dom(\sigma_1)}{(\sigma_1, \sigma_2) \vdash A \,\mathcal{R}_p\, B}.$$

A relation $\mathcal{R}$ is a weak bisimulation up to structural equivalence *if $\mathcal{R}$ satisfies the definition of weak bisimulation (Definition* 3.8*), but with the condition on the derivatives "$(\sigma_1', \sigma_2') \vdash P' \,\mathcal{R}\, Q'$" replaced by the weaker "$(\sigma_1', \sigma_2') \vdash P' \,\mathcal{R}_s\, Q'$." Weak bisimulation* up to weakening, contraction, restriction, *and* parallel composition *are defined similarly.*

Thus an up to technique t is essentially a functional $(\cdot)_t$ from compatible relations to compatible relations. We say that an up to technique t is *sound* if, whenever $\mathcal{R}$ is a bisimulation up to t, then $\mathcal{R} \subseteq \approx$. Our first task is therefore to prove that the techniques we have defined above are sound. Next, it should be obvious that different up to techniques can be combined to get new techniques. As an example, weak bisimulation up to parallel composition and contraction is defined by replacing $(\sigma_1', \sigma_2') \vdash P' \mathcal{R} Q'$ with $(\sigma_1', \sigma_2') \vdash P' (\mathcal{R}_p)_c Q'$ in Definition 3.8. Formally, a combination of techniques is a composition of the corresponding functionals (see [22]). The results of [22] ensure that, if the techniques we have introduced in Definition 4.16 are sound, then any combination of them is sound too.[3] The next proposition, whose proof can be found in Appendix B, states the soundness of our up to techniques.

PROPOSITION 4.17. *Let $\mathcal{R}$ be a weak bisimulation up to structural equivalence (resp., weakening, contraction, restriction, parallel composition). Then $\mathcal{R} \subseteq \mathcal{R}_s \subseteq \approx$ (resp., $\mathcal{R} \subseteq \mathcal{R}_t \subseteq \approx$, for $t = w, c, r, p$).*

We are now ready to prove the soundness theorem.

THEOREM 4.18 (soundness of weak bisimilarity). *Let P and Q be processes and σ_1 and σ_2 be equivalent substitutions. If $(\sigma_1, \sigma_2) \vdash P \approx Q$, then $(\sigma_1, \sigma_2) \vdash P \cong Q$.*

Proof. Note that $\approx$ is trivially a weak bisimulation up to parallel composition; thus $(\approx)_p \subseteq \approx$ due to Proposition 4.17. Hence $(\sigma_1, \sigma_2) \vdash P \approx Q$ implies $(\sigma_1, \sigma_2) \vdash P \mid R\sigma_1 \approx Q \mid R\sigma_2$ for each R with $fn(R) \subseteq dom(\sigma_1)$. Since $\approx$ is finer than $\dot{\cong}$, this fact implies the wanted $(\sigma_1, \sigma_2) \vdash P \cong Q$. □

Completeness. We shall prove completeness of $\approx$ relative to a class of processes that have an image-finiteness property (defined below). This property makes the proof relatively simple (and not far from, e.g., the proof for asynchronous bisimilarity given in [3]). On the other hand, the class of processes enjoying the property is broad enough to ensure that $\approx$ is a fairly general proof technique. At present, we do not know whether the proof can be extended to the full language.

Formally, a process P is *structurally image-finite* if, for each visible trace s, the set of equivalence classes $\{P' : P \stackrel{s}{\Longrightarrow} P'\}/\equiv$ is finite. Note that this notion is slightly more general than the usual image-finiteness (as considered, e.g., in [3]). This is due to our use of $\equiv$ to quotient the set of s- derivatives. As an example, process $P \stackrel{\text{def}}{=} !\, a(x).\bar{b}x$ is structurally image-finite but not image-finite; indeed, for each s, P has infinitely many s-derivatives, which are, however, finite up to structural equivalence (use the law $!\, P \mid P \equiv !\, P$). On the contrary, process $Q \stackrel{\text{def}}{=} !\, (\tau.\overline{a} + \tau.\overline{b})$ is not structurally image-finite (hence not image-finite); for instance, for any $n, m \geq 0$, Q has an ϵ-derivative of the form $\overline{a} \mid \cdots \mid \overline{a} \mid \overline{b} \mid \cdots \mid \overline{b} \mid Q$, with n $\overline{a}$'s and m $\overline{b}$'s in parallel.

Next, it is convenient to introduce a chain of relations $\approx_i$, $i \geq 0$, which are used to approximate bisimilarity $\approx$ over configurations. In what follows, $|E|$ is used to denote the syntactic size of some term E. Given any σ, we let $rk(\sigma) \stackrel{\text{def}}{=} |\phi_\sigma|$.

[3] Technically, our techniques enjoy the "respectfulness" property of [22] with respect to the transition relation $\vdash \stackrel{\mu}{\underset{\delta}{\longrightarrow}}$.

DEFINITION 4.19. *The binary relations $\approx_i$ over configurations are defined by induction on i as follows:*

- $(\sigma_1, \sigma_2) \vdash P \approx_0 Q$ *if* $\sigma_1 \sim \sigma_2$;
- $(\sigma_1, \sigma_2) \vdash P \approx_i Q$, *for* $i > 0$ *and* $\sigma_1 \sim \sigma_2$ *if, whenever* $\sigma_1 \rhd P \stackrel{\mu}{\underset{\delta}{\longmapsto}} \sigma_1' \rhd P'$ *with* $|\delta| \leq i$ *and* $rk(\sigma_1') \leq i$, *there are* μ', σ_2', *and* Q' *s.t.* $\sigma_2 \rhd Q \stackrel{\hat{\mu}}{\underset{\delta}{\Longmapsto}} \sigma_2' \rhd Q'$, $rk(\sigma_2') \leq i$, *and* $(\sigma_1', \sigma_2') \vdash P' \approx_{i-1} Q'$, *and the converse on the transitions of* Q *and* P.

We let $\approx_\omega \stackrel{\text{def}}{=} \cap_{i \geq 0} \approx_i$.

The following is a variation on a standard result for bisimulation (see, e.g., [15, 21]).

LEMMA 4.20. *Let* P *and* Q *be structurally image-finite processes. Then* $(\sigma_1, \sigma_2) \vdash P \approx Q$ *if and only if* $(\sigma_1, \sigma_2) \vdash P \approx_\omega Q$.

We now show that $\cong$ implies $\approx_\omega$, from which completeness of $\approx$ for structurally image-finite processes will follow. In order to do this, we exploit the formula ϕ_σ and define a class of canonical contexts $R_{i,\sigma}$, depending on some $i \geq 0$ and σ, which can be used to test whether two configurations are related by $\approx_i$. In what follows, we shall use some of the process notation introduced at the beginning of the subsection on completeness for may-testing. Furthermore, we shall sometimes omit the object part of action prefixes, writing, e.g., $\overline{c}$ instead of $\overline{c}x$, when x is not relevant. We will let $\tau.P$ denote the process $(\nu\, c)(c.P \,|\, \overline{c})$ $(c \notin fn(P))$ and, for any finite set of processes $\{P_1, \ldots, P_k\}$, we will let $\sum \{P_1, \ldots, P_k\}$ denote the process $P_1 + \cdots + P_k$ (the exact way the summands are arranged does not matter).

DEFINITION 4.21 (canonical contexts). *Define the processes* $R_{i,\sigma}$, *for* $i \geq 0$, *by induction on* i *as follows.* $R_{0,\sigma} \stackrel{\text{def}}{=} \mathbf{0}$ *and, for* $i > 0$,

$$
\begin{array}{rcl}
R_{n,\sigma} & \stackrel{\text{def}}{=} & \sum \left\{ R_\eta^{inp} + R_\eta^{out} \;:\; n(\eta) \subseteq dom(\sigma),\, \widehat{\eta\sigma} \in \mathcal{N} \text{ and } |\eta| < i \right\} + R_\epsilon + \overline{e}_i, \text{where} \\
R_\eta^{inp} & \stackrel{\text{def}}{=} & \langle \eta(x) \rangle . \sum \left\{ \phi_{\sigma'} \big(\overline{f}_{\eta,\phi_{\sigma'},i} + \tau.R_{i-1,\sigma'} \big) \;: \right. \\
 & & \qquad \left. \text{there is } M \text{ s.t. } \sigma' = \sigma[M/x], \text{ and } rk(\sigma') \leq i \right\}, \\
R_\eta^{out} & \stackrel{\text{def}}{=} & \sum \left\{ (\nu\, \widetilde{b}) \langle \overline{\eta} \zeta \rangle . \big(\overline{g}_{\eta,\zeta,i} + \tau.R_{i-1,\sigma'} \big) \;: \right. \\
 & & \qquad \left. fn\big((\nu\, \widetilde{b})\big) \overline{\eta} \langle \zeta \rangle \subseteq dom(\sigma), \sigma' = \sigma[\widetilde{b}/\widetilde{b}] \text{ and } |\zeta| < i \right\}, \\
R_\epsilon & \stackrel{\text{def}}{=} & \tau.\big(\overline{h}_i + \tau.R_{i-1,\sigma} \big),
\end{array}
$$

where the names e_j, $f_{\eta,\phi_{\sigma'},j}$, $g_{\eta,\zeta,j}$, *and* h_j *(*$0 \leq j \leq i$*) are all distinct and fresh.*

Note that, in the above definition, the sum in R_η^{inp} is finite because, for fixed $\widetilde{x}$ and i, there are finitely many ϕ_σ's and $R_{i,\sigma}$ s.t. $dom(\sigma) = \widetilde{x}$ and $rk(\sigma) \leq i$ (this is formally proven by induction on i). The sum in R_η^{out} is finite as well, because actions $(\nu\, \widetilde{b})\overline{\eta}\langle\zeta\rangle$ are considered up to alpha-equivalence. Note also that $fn(R_{i,\sigma}) \subseteq dom(\sigma) \cup \widetilde{l}$, where $\widetilde{l}$ is the set of all names e_j, $f_{\eta,\phi_{\sigma'},j}$, $g_{\eta,\zeta,j}$, and h_j $(0 \leq j \leq i)$ occurring in $R_{i,\sigma}$.

An easy lemma on characteristic formulae follows; its proof can be found in Appendix B.

LEMMA 4.22. *If* $\sigma \sim \sigma'$, *then* $rk(\sigma) = rk(\sigma')$.

We are now ready to prove completeness.

THEOREM 4.23 (completeness for barbed equivalence). *Let* P *and* Q *be structurally image-finite processes. If* $(\sigma_1, \sigma_2) \vdash P \cong Q$, *then* $(\sigma_1, \sigma_2) \vdash P \approx Q$.

Proof. By virtue of Lemma 4.20, it is sufficient to prove that for each $i \geq 0$ it holds $(\sigma_1, \sigma_2) \vdash P \approx_i Q$. Consider R_{i,σ_1} and let $\rho_1 \stackrel{\text{def}}{=} \sigma_1[\widetilde{l}/\widetilde{l}]$ and $\rho_2 \stackrel{\text{def}}{=} \sigma_2[\widetilde{l}/\widetilde{l}]$, where $\widetilde{l}$ is the set of all names e_j, $f_{\eta,\phi_{\sigma'},j}$, $g_{\eta,\zeta,j}$, and h_j $(0 \leq j \leq i)$ occurring in R_{i,σ_1}

(we suppose that $\widetilde{l}$ has been chosen fresh for σ_1, σ_2, P, and Q). We prove that if $(\rho_1,\rho_2) \vdash (P \mid R_{i,\sigma_1}\sigma_1) \mathrel{\dot\cong} (Q \mid R_{i,\sigma_1}\sigma_2)$, then $(\sigma_1,\sigma_2) \vdash P \approx_i Q$. From this fact and $(\sigma_1,\sigma_2) \vdash P \cong Q$, the thesis will follow (note that $(\sigma_1,\sigma_2) \vdash P \cong Q$ trivially implies $(\rho_1,\rho_2) \vdash P \cong Q$).

We proceed by induction on i. The case $i = 0$ is trivial; thus suppose $i > 0$. We only consider the case of a (C-Inp) transition, that is,

$$\sigma_1 \rhd P \xmapsto[\eta(x)]{(\nu\,\widetilde{b})\overline{a}\langle M\rangle} \sigma_1' \rhd P' \tag{1}$$

with $|\eta(x)| \le i$, $\sigma_1' = \sigma_1[^M\!/_x]$, and $rk(\sigma_1') \le i$, as the other cases are similar or easier. We show the existence of a transition of $\sigma_2 \rhd Q$ that matches this one. From (1) above, we can infer that

$$P \mid R_{i,\sigma_1}\sigma_1 \xrightarrow{\tau} (\nu\,\widetilde{b})\left(P' \mid \left(\phi_{\sigma_1'}\left(\overline{f}_{\eta,\phi_{\sigma_1'},i} + \tau.R_{i-1,\sigma_1'}\right)\right)\sigma_1'\right) \stackrel{\text{def}}{=} A.$$

Since $(\rho_1,\rho_2) \vdash P \mid R_{i,\sigma_1}\sigma_1 \mathrel{\dot\cong} Q \mid R_{i,\sigma_1}\sigma_2$ and $A \downarrow f_{\eta,\phi_{\sigma_1'},i}$, we deduce the existence of a transition

$$Q \mid R_{i,\sigma_1}\sigma_2 \Longrightarrow (\nu\,\widetilde{b}')\left(Q' \mid \left(\phi_{\sigma_1'}\left(\overline{f}_{\eta,\phi_{\sigma_1'},i} + \tau.R_{i-1,\sigma_1'}\right)\right)\sigma_2'\right) \stackrel{\text{def}}{=} B$$

with $(\rho_1,\rho_2) \vdash A \mathrel{\dot\cong} B$ and $B \downarrow f_{\eta,\phi_{\sigma_1'},i}$, where $\sigma_2' = \sigma_2[^{M'}\!/_x]$, for some M' s.t. $Q \xRightarrow{(\nu\,\widetilde{b}')\overline{a'}\langle M'\rangle} Q'$ ($a' = \widehat{\eta\sigma_2}$). Hence

$$\sigma_2 \rhd Q \xRightarrow[\eta(x)]{(\nu\,\widetilde{b}')\overline{a'}\langle M'\rangle} \sigma_2' \rhd Q'. \tag{2}$$

Moreover, since $B \downarrow f_{\eta,\phi_{\sigma_1'},i}$, it holds that $\sigma_2' \models \phi_{\sigma_1'}$.

Now, from $A \xrightarrow{\tau} \equiv (\nu\,\widetilde{b})\big((P' \mid R_{i-1,\sigma_1'})\sigma_1'\big) \stackrel{\text{def}}{=} A'$ we deduce that $B \Longrightarrow B'$ with $(\rho_1,\rho_2) \vdash A' \mathrel{\dot\cong} B'$. Since $A' \downarrow e_{i-1}$, it must hold that $B' \Downarrow e_{i-1}$; hence we must have $B' \equiv (\nu\,\widetilde{b}')\big((Q'' \mid R_{i-1,\sigma_1'})\sigma_2'\big)$ with $Q' \Longrightarrow Q''$. We can strip the restrictions $(\nu\,\widetilde{b})$ and $(\nu\,\widetilde{b}')$ away from $(\rho_1,\rho_2) \vdash A' \mathrel{\dot\cong} B'$ and deduce that

$$(\rho_1,\rho_2) \vdash (P' \mid R_{i-1,\sigma_1'}\sigma_1') \mathrel{\dot\cong} (Q'' \mid R_{i-1,\sigma_1'}\sigma_2'),$$

which, by induction, implies that $(\sigma_1',\sigma_2') \vdash P' \approx_{i-1} Q''$. Now, from $Q' \Longrightarrow Q''$ and transition (2), we deduce

$$\sigma_2 \rhd Q \xRightarrow[\eta(x)]{(\nu\,\widetilde{b}')\overline{a'}\langle M'\rangle} \sigma_2' \rhd Q''\,.$$

We show that this transition matches (1). Indeed, we have that $\sigma_1' \sim \sigma_2'$ (by $\sigma_2' \models \phi_{\sigma_1'}$ and Theorem 4.3), that $rk(\sigma_2') = rk(\sigma_1') \le i$ (Lemma 4.22), and that $(\sigma_1',\sigma_2') \vdash P' \approx_{i-1} Q''$. □

5. Applications. In this section we first give some properties which are useful when reasoning on cryptographic processes and then use them in a few examples.

Let **rel** $\in \{\approx, \ll\}$.

(C-INP) Suppose that for all ζ such that $\widetilde{y} \stackrel{\text{def}}{=} (n(\zeta) - dom(\sigma_1))$ are fresh and $\widehat{\zeta\sigma_1} \neq \bot$ it holds that $(\sigma_1[\widetilde{y}/\widetilde{y}], \sigma_2[\widetilde{y}/\widetilde{y}]) \vdash P[\widehat{\zeta\sigma_1}/x]$ **rel** $Q[\widehat{\zeta\sigma_2}/x]$.
Suppose $a_i = \widehat{\eta\sigma_i}$ $(i = 1, 2)$ with $n(\eta) \subseteq dom(\sigma_1)$.
Then $(\sigma_1, \sigma_2) \vdash a_1(x).P$ **rel** $a_2(x).Q$.

(C-OUT) Suppose that $(\sigma_1[M_1/x], \sigma_2[M_2/x]) \vdash P$ **rel** Q and that $a_i = \widehat{\eta\sigma_i}$ $(i = 1, 2)$ with $n(\eta) \subseteq dom(\sigma_1)$.
Then $(\sigma_1[M_1/x], \sigma_2[M_2/x]) \vdash \overline{a}_1 M_1.P$ **rel** $\overline{a}_2 M_2.Q$.

(C-PAR) Suppose that $fn(R) \subseteq dom(\sigma_1)$ and $(\sigma_1, \sigma_2) \vdash P$ **rel** Q.
Then $(\sigma_1, \sigma_2) \vdash P \mid R\sigma_1$ **rel** $Q \mid R\sigma_2$.

(C-RES) Suppose that $(\sigma_1, \sigma_2) \vdash P$ **rel** Q, that $\widetilde{k} \cap n(\sigma_1) = \emptyset$, and that $\widetilde{h} \cap n(\sigma_2) = \emptyset$.
Then $(\sigma_1, \sigma_2) \vdash (\nu\,\widetilde{k})P$ **rel** $(\nu\,\widetilde{h})Q$.

FIG. 4. *Some congruence rules for* $\ll$ *and* $\approx$.

5.1. Some useful laws. We start by stating some simple properties.

PROPOSITION 5.1. *Let* **rel** $\in \{\approx, \ll\}$.

- (Reflexivity) *For any* σ *and* P, $(\sigma, \sigma) \vdash P$ **rel** P.
- (Transitivity) *If* $(\sigma_1, \sigma_2) \vdash P$ **rel** Q *and* $(\sigma_2, \sigma_3) \vdash Q$ **rel** R, *then* $(\sigma_1, \sigma_3) \vdash P$ **rel** R.
- (Weakening) *Suppose that* $(\sigma_1[M/x], \sigma_2[N/x]) \vdash P$ **rel** Q. *Then* $(\sigma_1, \sigma_2) \vdash P$ **rel** Q.
- (Contraction) *Suppose that* $(\sigma_1, \sigma_2) \vdash P$ **rel** Q *and consider any* ζ *such that* $n(\zeta) \subseteq dom(\sigma_1)$ *and* $\widehat{\zeta\sigma_1} \neq \bot$. *Then* $(\sigma_1[\widehat{\zeta\sigma_1}/x], \sigma_2[\widehat{\zeta\sigma_2}/x]) \vdash P$ **rel** Q.
- (Structural equivalence) *Suppose that* $P \equiv Q$. *Then for any* σ, $(\sigma, \sigma) \vdash P$ **rel** Q.

Proof. Reflexivity, transitivity, and structural equivalence are trivial. The other cases are consequences of Proposition 4.17 for $\approx$. For $\ll$, the proof becomes trivial when one switches to the original definition of $\lesssim$; for contraction, note that $O\sigma_i[\widehat{\zeta\sigma_i}/x] \equiv (\texttt{let}\ x = \zeta\ \texttt{in}\ O)\sigma_i$ for $i = 1, 2$ and $fn(O) \subseteq dom(\sigma_i)$. ∎

Some congruence laws are listed in Figure 4. These laws are very useful (especially (C-PAR) and (C-RES)) because they permit a kind of compositional reasoning, as we shall see in later examples in this section.

PROPOSITION 5.2. *The laws listed in Figure* 4 *are correct.*

Proof. The proof for (C-INP) and (C-OUT) is trivial. Laws (C-PAR) and (C-RES) are a consequence of Proposition 4.17 in the case of $\approx$. In the case of $\ll$, the proof becomes trivial if one switches to the original definition of $\lesssim$. ∎

We shall also need a few rules to reason on environments. They are given in the following two lemmas (whose proofs can be found in Appendix A). The first lemma characterizes $kn(\sigma)$ in terms of the expressions that can be formed using the variables in $dom(\sigma)$.

LEMMA 5.3. *Let* σ *be an environment. Then* $kn(\sigma) = \{\widehat{\zeta\sigma} \in \mathcal{N} : n(\zeta) \subseteq dom(\sigma)\}$.

The next lemma is about the effect of evaluating the same expression ζ under two

equivalent environments, σ and σ'.

LEMMA 5.4. *Let $\sigma = [M_i/x_i]_{i\in I}$ and $\sigma' = [M'_i/x_i]_{i\in I}$ be two substitutions s.t. $\sigma \sim' \sigma'$. Define $\widetilde{N} = core(\sigma, x_i)_{i\in I}$ and $\widetilde{N}' = core(\sigma', x_i)_{i\in I}$. For each ζ s.t. $n(\zeta) \subseteq \widetilde{x}$, either*

(a) *$\widehat{\zeta\sigma} = \widehat{\zeta\sigma'} = \bot$, or*

(b) *there are $i \in I$ and a tuple $\widetilde{\jmath} \subseteq I$ such that $\widehat{\zeta\sigma} = \{N_i\}_{\widetilde{N}[\widetilde{\jmath}]}$ and $\widehat{\zeta\sigma'} = \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath}]}$.*

We end this subsection with a small example (borrowed from [6]) that shows the use of our congruence laws.

EXAMPLE 5.5. *Let us consider the processes $P \stackrel{\text{def}}{=} (\nu\, k)\overline{c}\{d\}_k.\, c(x).\, [x = k]\overline{c}\{d\}_k$ and $Q \stackrel{\text{def}}{=} (\nu\, k)\overline{c}\{d\}_k.\, c(x)$. Process P creates a private key k, sends d encrypted under k, listens for an input, and if it receives k, then resends $\{d\}_k$. Process Q behaves like P, but, after the reception of one message, it becomes stuck. Since k is a private key that is never disclosed to the environment, P will never receive k back at c; as a consequence the matching $[x = k]$ will never become true. Therefore P and Q should be considered as equivalent. Let $V = fn(P, Q) = \{c, d\}$; we want to show that $(\epsilon_V, \epsilon_V) \vdash P \simeq Q$.*

We can prove that $(\epsilon_V, \epsilon_V) \vdash Q \ll P$ by simply noting that traces of Q are also traces of P. To prove that $(\epsilon_V, \epsilon_V) \vdash P \ll Q$, let z be any fresh name and let $\sigma \stackrel{\text{def}}{=} \epsilon_V[\{d\}_k/z]$. The crucial step is showing that

$$(\sigma, \sigma) \vdash c(x).\, [x = k]\overline{c}\{d\}_k \ll c(x).$$

Indeed, for any ζ with $\widetilde{y} \stackrel{\text{def}}{=} (n(\zeta) - dom(\sigma))$ fresh, it holds that $\widehat{\zeta\sigma[\widetilde{y}/\widetilde{y}]} = \widehat{\zeta\sigma} \neq k$ (because of $k \notin kn(\sigma)$ and of Lemma 5.3); hence we have $(\sigma[\widetilde{y}/\widetilde{y}], \sigma[\widetilde{y}/\widetilde{y}]) \vdash [\widehat{\zeta\sigma} = k]\overline{c}\{d\}_k \ll \mathbf{0}$. The thesis for this step follows by using (C-INP). *Now, using* (C-OUT), *we have that*

$$(\sigma, \sigma) \vdash \overline{c}\{d\}_k.\, c(x).\, [x = k]\overline{c}\{d\}_k \ll \overline{c}\{d\}_k.\, c(x).$$

The thesis follows from this fact, using first weakening and then (C-RES).

5.2. Secure channels implementation. In the following examples, we show the use of our framework for proving security properties of communication protocols. In the same vein of [1, 4], the idea is that of implementing communication on secure (private) channels by means of encrypted communication on public channels. Let us consider the π-calculus process:

$$P \stackrel{\text{def}}{=} (\nu\, c)(\overline{c}d \mid c(z).\, R),$$

where c does not occur in R. Process P creates a private channel c which is used to transmit name d. Communication on c is secure because no execution context knows c. Since P consists of two concurrent subprocesses, the actual implementation could allocate them onto two different computers, whose interconnections are not guaranteed to be secure. Communication on c has to be implemented in terms of lower-level, encrypted communication on some public channel, say p. Thus, process P might be implemented as

$$I_P \stackrel{\text{def}}{=} (\nu\, k_c)(\overline{p}\{d\}_{k_c} \mid p(x).\mathtt{let}\ z = \mathtt{dec}_{k_c}(x)\ \mathtt{in}\ R) \qquad (k_c,\, x \notin fn(R)).$$

In I_P, name k_c is a private encryption key that corresponds to channel c. The behavior of the process is as follows: the process $\overline{p}\{d\}_{k_c}$ sends d encrypted under k_c, while $p(x).\texttt{let } z = \texttt{dec}_{k_c}(x) \texttt{ in } R$ tries to decrypt a ciphertext x received at p. If the decryption succeeds, a cleartext is obtained and bound to z and the process behaves like R, otherwise the process is stuck. Note that this implementation does not guarantee that d will eventually be passed to R: message $\{d\}_{k_c}$ could be captured by some context (attacker) listening at p. The last example of this section shall present an implementation that solves this problem.

A secrecy property. Assume that R keeps z secret under any context; i.e., for every d and d', and $\sigma_1 \sim \sigma_2$, it holds that $(\sigma_1, \sigma_2) \vdash R[d/z] \simeq R[d'/z]$. Under this hypothesis, we want to prove that the implementation scheme for P preserves secrecy. To see this, we consider a generic d', let $Q \stackrel{\text{def}}{=} (\nu\, c)(\overline{c}d' \mid c(z).\, R)$, and show that

$$(\epsilon_V, \epsilon_V) \vdash I_P \simeq I_Q,$$

where I_Q is the obvious implementation of Q and $V = fn(I_P, I_Q)$. In order to prove this, let y be any fresh name and define $\sigma_1 \stackrel{\text{def}}{=} \epsilon_V[\{d\}_{k_c}/y]$ and $\sigma_2 \stackrel{\text{def}}{=} \epsilon_V[\{d'\}_{k_c}/y]$. First, rule (C-Inp) allows one to prove that

$$(\sigma_1, \sigma_2) \vdash (p(x).\texttt{let } z = \texttt{dec}_{k_c}(x) \texttt{ in } R) \simeq (p(x).\texttt{let } z = \texttt{dec}_{k_c}(x) \texttt{ in } R).$$

To prove this, one exploits two facts: (1) for any ζ s.t. $n(\zeta) - dom(\sigma_1)$ are fresh, if $\widehat{\zeta\sigma_1} = \{M\}_{k_c}$, then $M = d$ and $\widehat{\zeta\sigma_2} = \{d'\}_{k_c}$ (by Lemma 5.4(b) and $k_c \notin kn(\sigma_1)$); (2) $R[d/z]$ is may-equivalent to $R[d'/z]$ under σ_1 and σ_2. These facts and (C-Par) can be used to infer that

$$\begin{aligned}&(\sigma_1, \sigma_2) \vdash (\overline{p}\{d\}_{k_c} \mid p(x).\texttt{let } z = \texttt{dec}_{k_c}(x) \texttt{ in } R)\\ &\simeq (\overline{p}\{d'\}_{k_c} \mid p(x).\texttt{let } z = \texttt{dec}_{k_c}(x) \texttt{ in } R).\end{aligned}$$

Finally, the desired claim follows by applying weakening and then (C-Res) (with $(\nu\, k_c)$) to the equality above.

Preservation of may semantics. Here we show that the previous implementation scheme also preserves may semantics. We relax the hypothesis that R keeps name z secret and, for the sake of simplicity, assume that $R \stackrel{\text{def}}{=} \overline{b}z$. In π-calculus, process P is may-equivalent to process $\overline{b}d$. We want to show that the implementations of P and of process $\overline{b}d$ are still may-equivalent, under the assumption that the communication channel p is both *asynchronous* and *noisy*. Thus, the actual implementation also includes a buffer $B \stackrel{\text{def}}{=}\!!p(x).\overline{p}x$ and a noise generator $N \stackrel{\text{def}}{=}\!!(\nu\, k)\overline{p}\{k\}_k$ for p. Both noise and asynchrony are necessary to prevent the execution context from detecting traffic on the public channel p. Let $V = fn(I_P, \overline{b}d, N, B)$. To sum up, we want to show that

$$(\epsilon_V, \epsilon_V) \vdash (I_P \mid N \mid B) \simeq (\overline{b}d \mid N \mid B). \tag{3}$$

(Note that this equation is *not* valid for $\approx$.) We do this in two steps.

- First, we prove that $(\epsilon_V, \epsilon_V) \vdash \overline{b}d \mid N \mid B \ll I_P \mid N \mid B$. The only possible trace for the configuration $\epsilon_V \rhd \overline{b}d$ is

$$\epsilon_V \rhd \overline{b}d \stackrel{\overline{b}d}{\underset{b(x)}{\longmapsto}} \epsilon_V[d/x] \rhd \mathbf{0}.$$

Configuration $\epsilon_V \rhd I_P$ can simulate the action above by first communicating $\{d\}_{k_c}$ on p, and then decrypting $\{d\}_{k_c}$:

$$\epsilon_V \rhd I_P \stackrel{\overline{b}d}{\stackrel{\Longrightarrow}{b(x)}} \epsilon_V[d/x] \rhd (\nu\, k_c)\mathbf{0}.$$

Hence, $(\epsilon_V, \epsilon_V) \vdash \overline{b}d \ll I_P$, and the thesis follows by applying law (C-Par) in Figure 4.

- Let us now prove that $(\epsilon_V, \epsilon_V) \vdash I_P \mid N \mid B \ll \overline{b}d \mid N \mid B$. Let y be any fresh name and let $\sigma \stackrel{\text{def}}{=} \epsilon_V[\{d\}_{k_c}/y]$.
  - The crucial step is showing that

    $$(\sigma, \sigma) \vdash p(x).\texttt{let}\ z = \texttt{dec}_{k_c}(x)\ \texttt{in}\ \overline{b}z \ll p(x).\overline{b}d. \tag{4}$$

    Indeed, taking any ζ such that $\widetilde{w} \stackrel{\text{def}}{=} (n(\zeta) - dom(\sigma))$ are fresh names and such that $\widehat{\zeta\sigma} \neq \perp$, we have $(\sigma[\widetilde{w}/\widetilde{w}], \sigma[\widetilde{w}/\widetilde{w}]) \vdash \texttt{let}\ z = \texttt{dec}_{k_c}(\widehat{\zeta\sigma})\ \texttt{in}\ \overline{b}z \ll \overline{b}d$. In fact, the only case in which $\texttt{dec}_{k_c}(\widehat{\zeta\sigma})$ does not evaluate to $\perp$ is when $\widehat{\zeta\sigma} = \{d\}_{k_c}$ (Lemma 5.4(b) and $k_c \notin kn(\sigma)$), which implies $\texttt{let}\ z = \texttt{dec}_{k_c}(\widehat{\zeta\sigma})\ \texttt{in}\ \overline{b}z \equiv \overline{b}d$. Then (4) above follows, using (C-Inp).
  - Now, using (C-Par) and (4) above, we have that

    $$(\sigma, \sigma) \vdash \overline{p}\{d\}_{k_c} \mid (p(x).\texttt{let}\ z = \texttt{dec}_{k_c}(x)\ \texttt{in}\ \overline{b}z) \ll \overline{p}\{d\}_{k_c} \mid p(x).\overline{b}d;$$

    hence, by weakening and (C-Res), we have that

    $$\begin{aligned}(\epsilon_V, \epsilon_V) \vdash\ & (\nu\, k_c)(\overline{p}\{d\}_{k_c} \mid p(x).\texttt{let}\ z = \texttt{dec}_{k_c}(x)\ \texttt{in}\ \overline{b}z) \\ & \ll (\nu\, k_c)(\overline{p}\{d\}_{k_c} \mid p(x).\overline{b}d) \equiv (\nu\, k_c)(\overline{p}\{d\}_{k_c}) \mid p(x).\overline{b}d.\end{aligned}$$

    In the last step we have used a structural law for restriction ($(\nu\, a)(A_1|A_2) \equiv ((\nu\, a)A_1) \mid A_2$ if $a \notin fn(A_2)$). Using (C-Par) again, we can put the context $N \mid B$ in parallel with the two processes:

    $$\begin{aligned}(\epsilon_V, \epsilon_V) \vdash\ & (\nu\, k_c)(\overline{p}\{d\}_{k_c} \mid p(x).\texttt{let}\ z = \texttt{dec}_{k_c}(x)\ \texttt{in}\ \overline{b}z) \mid N \mid B \\ & \ll (\nu\, k_c)(\overline{p}\{d\}_{k_c}) \mid p(x).\overline{b}d \mid N \mid B.\end{aligned}$$

    Now, $(\nu\, k_c)(\overline{p}\{d\}_{k_c})$ in the right-hand side above can be turned into a particle of noise, because $(\epsilon_V, \epsilon_V) \vdash (\nu\, k_c)(\overline{p}\{d\}_{k_c}) \simeq (\nu\, k)(\overline{p}\{k\}_k)$. Using the structural law for replication ($!A \equiv A \mid !A$), this particle of noise can be absorbed by N; hence

    $$(\epsilon_V, \epsilon_V) \vdash (\nu\, k_c)(\overline{p}\{d\}_{k_c}) \mid p(x).\overline{b}d \mid N \mid B \ \simeq\ p(x).\overline{b}d \mid N \mid B.$$

    Moreover, as an instance of a general law for asynchronous channels, we have that

    $$(\epsilon_V, \epsilon_V) \vdash\ p(x).\overline{b}d \mid B \ll \overline{b}d \mid B,$$

    and the thesis easily follows by (C-Par).

Ensuring message delivery. We consider a more sophisticated implementation scheme for process P and prove that (under a fairness assumption) this scheme guarantees that a message sent on channel c is eventually delivered. Again, we implement c with an asynchronous and noisy public channel p. This time, however, we need a more complex source of noise, $N \stackrel{\text{def}}{=} !(\nu\, k)!\overline{p}\{k\}_k$. Note the difference from the previous example: N can now spawn at any time a process $(\nu\, k)!\overline{p}\{k\}_k$ which emits a constant noise $\{k\}_k$ at p. The buffer B for p is still $B \stackrel{\text{def}}{=} !p(x).\overline{p}x$.

In this example, we shall use recursive definitions of agent constants of the kind $A \Longleftarrow S$, where A is an agent constant that may appear in the process expression S (these can be taken as primitive—the theory extends smoothly—or can be coded up using replication like in [15]). We also use the shorthand "$\texttt{let}\ z = \zeta\ \texttt{in}\ A\ \texttt{else}\ B$" for "$(\texttt{let}\ z = \zeta\ \texttt{in}\ A) + \neg(\texttt{let}\ z = \zeta\ \texttt{in}\ \not{t})B$."

The implementation of P is the process

$$I_P \stackrel{\text{def}}{=} (\nu\, k_c)\big(!\overline{p}\{d\}_{k_c} \mid R\big), \qquad \text{where}$$
$$R \Longleftarrow p(x).\texttt{let}\ z = \texttt{dec}_{k_c}(x)\ \texttt{in}\ \overline{b}z\ \texttt{else}\ (\overline{p}x \mid R).$$

Component $!\overline{p}\{d\}_{k_c}$ constantly emits d encrypted under key k_c on p, while R repeatedly tries to decrypt a ciphertext x received on p using k_c. When the decryption succeeds, the cleartext is sent on b.

Let $V = fn(I_P, \overline{b}d, B, N)$; we want to prove that

$$(\epsilon_V, \epsilon_V) \vdash (I_P \mid B \mid N) \approx (\overline{b}d \mid B \mid N).$$

In order to see this, define $\sigma_1 \stackrel{\text{def}}{=} \epsilon_V[\{d\}_{k_c}/y]$ and $\sigma_2 \stackrel{\text{def}}{=} \epsilon_V[\{k\}_k/y]$ (y fresh). We first show that

$$(\sigma_1, \sigma_2) \vdash T \stackrel{\text{def}}{=} (!\overline{p}\{d\}_{k_c} \mid R \mid B \mid N) \approx (\overline{b}d \mid !\overline{p}\{k\}_k \mid B \mid N) \stackrel{\text{def}}{=} U, \tag{5}$$

from which the thesis will follow by first applying weakening to discard the y-entry, then (C-Res) (with $(\nu\, k_c)$ on the left-hand side and $(\nu\, k)$ on the right-hand side), and then the structural laws for restriction and the structural law for replication ($N \mid (\nu\, k)!\overline{p}\{k\}_k \equiv N$) on the right-hand side. To prove (5), we consider a relation $\mathcal{R}$ consisting of *three* pairs (w is fresh):

$$\mathcal{R} = \Big\{ \big(\sigma_1 \rhd T,\, \sigma_2 \rhd U\big),\ \big(\sigma_1 \rhd \overline{b}d,\, \sigma_2 \rhd \overline{b}d\big),\ \big(\sigma_1[d/w] \rhd \mathbf{0},\, \sigma_2[d/w] \rhd \mathbf{0}\big) \Big\}$$

and show that $\mathcal{R}$ is a weak bisimulation up to parallel composition and contraction. The proof consists of analyzing every transition of $\sigma_1 \rhd T$ and $\sigma_2 \rhd U$ and of showing that a matching transition exists in the other configuration in each case. In particular, note the following:

- transitions of $\sigma_1 \rhd T$ originating from $!\overline{p}\{d\}_{k_c}$ are matched up to contraction via transitions from $!\overline{p}\{k\}_k$ in $\sigma_2 \rhd U$, and vice-versa;
- the transition of $\sigma_1 \rhd T$ originating from $R \xrightarrow{p\,\{d\}_{k_c}} \equiv \overline{b}d$ is matched up to parallel composition via $B \xrightarrow{p\,\{k\}_k} \overline{p}\{k\}_k \mid B$ in $\sigma_2 \rhd U$—to see this, first note that $\overline{p}\{k\}_k \mid !\overline{p}\{k\}_k \equiv !\overline{p}\{k\}_k$, and then cut away the parallel contexts $!\overline{p}\{d\}_{k_c} \mid B \mid N$ (from the left-hand side) and $!\overline{p}\{k\}_k \mid B \mid N$ (from the right-hand side);

- a transition of $\sigma_1 \rhd T$ originating from $R \xrightarrow{p\,M} \equiv \overline{p}M \mid R$, with $M = \widehat{\zeta\sigma_1} \neq \{d\}_{k_c}$, is matched up to parallel composition and contraction via a transition $B \xrightarrow{p\,M'} \equiv \overline{p}M' \mid B$ in $\sigma_2 \rhd U$, where $M' = \widehat{\zeta\sigma_2}$ (contraction may be used to discard any new name introduced by ζ);
- the transition of $\sigma_2 \rhd U$ originating from $\overline{b}d \xrightarrow{\overline{b}\langle d\rangle} \mathbf{0}$ is matched up to parallel composition via a communication between R and $!\overline{p}\{d\}_{k_c}$, followed by a $\overline{b}\langle d\rangle$-transition in $\sigma_1 \rhd T$.

Communications between R and $!\overline{p}\{d\}_{k_c}$ or R and N are treated as in the second and in the third item above, respectively. It should be now obvious how the other transitions match with each other.

5.3. Verification of a small protocol. Consider a system where two agents A and B share two secret keys, k_{AS} and k_{BS}, respectively, with a server S. The purpose of the protocol is to establish a new secret key k between A and B, which A may use to pass some confidential information d to B. This is achieved with a version of the Wide Mouthed Frog Protocol (see, e.g., [7]). For the sake of simplicity, we suppose that the protocol is always started by A and that all communications occur on a public channel, say p. Informally, the protocol can be described as follows:

$$\begin{array}{lll} \text{Message 1} & A \longrightarrow S: & \{k\}_{k_{AS}} \\ \text{Message 2} & S \longrightarrow B: & \{k\}_{k_{BS}} \\ \text{Message 3} & A \longrightarrow B: & \{d\}_k. \end{array}$$

Our intent here is to verify one run of the protocol (that is, we do not consider the case of multiple agents simultaneously executing the protocol). In our language, the above notation translates to a process $P(d)$ defined as follows (using the notation $R(w)$ to stress that name w may occur free in R, and, for any M, abbreviating $R[M/w]$ as $R(M)$; bound names are all distinct):

$$\begin{array}{rcl} A(d) & \stackrel{\text{def}}{=} & \overline{p}\{k\}_{k_{AS}}.\overline{p}\{d\}_k.\mathbf{0}, \\ S & \stackrel{\text{def}}{=} & p(x).\mathtt{let}\ x' = \mathtt{dec}_{k_{AS}}(x)\ \mathtt{in}\ \overline{p}\{x'\}_{k_{BS}}.\mathbf{0}, \\ B & \stackrel{\text{def}}{=} & p(y).\mathtt{let}\ y' = \mathtt{dec}_{k_{BS}}(y)\ \mathtt{in}\ p(z).\mathtt{let}\ z' = \mathtt{dec}_{y'}(z)\ \mathtt{in}\ \mathbf{0}, \\ P(d) & \stackrel{\text{def}}{=} & (\nu\, k_{AS},\, k_{BS})\big(((\nu\, k)A(d)) \mid S \mid B\big). \end{array}$$

Here we assume that A, S, and B terminate after the exchange of message 3: this assumption simplifies the reasoning below and seems sensible, because the correctness of the protocol should be assessed independently of the subsequent behavior of the participants. In what follows, following Abadi and Gordon [7], we use the contextual equivalence $\cong$ to express the properties of secrecy and integrity of $P(d)$ (below, we suppose by alpha-equivalence that names k_{AS}, k_{BS}, k, x, x', y, y', z, z' do not occur in messages M and M'):

Secrecy (“$P(d)$ does not leak d”).
For any M and M', it holds that $P(M) \cong P(M')$.

Integrity (“if B accepts a message $\{N\}_k$ then $N = d$”).
For any M, it holds that $P(M) \cong P^M_{spec}$, where

$$\begin{array}{rcl} P^M_{spec} & \stackrel{\text{def}}{=} & (\nu\, k_{AS},\, k_{BS})\big(((\nu\, k)A(d)) \mid S \mid B^M_{spec}\big), \\ B^M_{spec} & \stackrel{\text{def}}{=} & p(y).\mathtt{let}\ y' = \mathtt{dec}_{k_{BS}}(y)\ \mathtt{in}\ p(z).\mathtt{let}\ z' = \mathtt{dec}_{y'}(z)\ \mathtt{in}\ [z' \neq M]\,\overline{p}\,\mathtt{err}.\mathbf{0}. \end{array}$$

Output of name `err` on p is used to signal a violation of integrity, i.e., that some message $\{N\}_k$ with $N \neq M$ has been accepted. The fact that $P(M)$ and P^M_{spec} are

equivalent means that action $\overline{p}\langle \mathtt{err} \rangle$, hence the violation, never occurs (in this point our formalization differs a little from Abadi and Gordon's). Fix M and M' and let $V \stackrel{\text{def}}{=} fn(P(M), P(M'))$. By virtue of our soundness results for secrecy it will be sufficient to show that

$$(\epsilon_V, \epsilon_V) \vdash P(M) \approx P(M'), \tag{6}$$

while, for integrity, it will be sufficient to establish that

$$(\epsilon_{V \cup \{\mathtt{err}\}}, \epsilon_{V \cup \{\mathtt{err}\}}) \vdash P(M) \approx P_{spec}(M). \tag{7}$$

We will prove the above equalities by reasoning compositionally on processes. To do this, we first show a compositionality result for $\approx$. Let us say that a process R is σ-*safe* if for each s, whenever $\sigma \rhd R \stackrel{s}{\underset{u}{\Longrightarrow}} \sigma' \rhd R' \stackrel{(\nu\, \widetilde{b})\overline{a}\langle M \rangle}{\underset{\eta(x)}{\Longrightarrow}}$, then $M \in dc(\sigma')$ (hence $\widetilde{b} = \emptyset$). Intuitively, R is σ-safe if R cannot increase the knowledge of σ. The following proposition strengthens the congruence rule for parallel composition (C-Par), under the assumption that the involved processes are safe for the appropriate environments (the proof is in Appendix B).

Proposition 5.6. *Suppose that* $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$ *and that* $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$. *Suppose that, for* $i = 1, 2$, Q_i *and* R_i *are* σ_i*-safe. Then* $(\sigma_1, \sigma_2) \vdash Q_1 \mid R_1 \approx Q_2 \mid R_2$.

Let us examine secrecy first. Define

$$\sigma \stackrel{\text{def}}{=} \epsilon_V[\{k\}_{k_{AS}}/x_1,\ \{k\}_{k_{BS}}/x_2,\ \{M\}_k/x_3] \qquad \text{and}$$
$$\sigma' \stackrel{\text{def}}{=} \epsilon_V[\{k\}_{k_{AS}}/x_1,\ \{k\}_{k_{BS}}/x_2,\ \{M'\}_k/x_3].$$

Clearly, $\sigma \sim \sigma'$. As a first step, check that

$$\begin{array}{lrcl} (\sigma, \sigma') \vdash & A(M) & \approx & A(M'), \\ (\sigma, \sigma') \vdash & S & \approx & S, \\ (\sigma, \sigma') \vdash & B & \approx & B. \end{array}$$

The first equality follows from (C-Out), while the second and the third follow from (C-Inp) and Lemma 5.4. For instance, to establish the second equality, Lemma 5.4 is first used to check that whenever names $n(\zeta) - dom(\sigma)$ are fresh and $\widehat{\zeta\sigma_i} = \{N\}_{k_{BS}}$, then $N = k$; then (C-Inp) is applied. Next, it is easy to see that $A(M)$ is σ-safe and that $A(M')$ is σ'-safe, while S and B are both σ- and σ'-safe (again, this requires the use of Lemma 5.4). Applying Proposition 5.6, we can infer that

$$(\sigma, \sigma') \vdash A(M) \mid S \mid B \approx A(M') \mid S \mid B.$$

Next, apply weakening (so as to discard entries x_1, x_2, and x_3) and then (C-Res) with $(\nu\, k)$ and a structural law $((\nu\, a)(Q \mid R) \equiv ((\nu\, a)Q) \mid R$ if $a \notin fn(R))$ and deduce that

$$(\epsilon_V, \epsilon_V) \vdash ((\nu\, k)A(M)) \mid S \mid B \approx ((\nu\, k)A(M')) \mid S \mid B.$$

Finally apply (C-Res) with $(\nu\, k_{AS},\ k_{BS})$ to get the desired (6).

As to integrity, let $\sigma \stackrel{\text{def}}{=} \epsilon_{V \cup \{\mathtt{err}\}}[\{k\}_{k_{AS}}/x_1,\ \{k\}_{k_{BS}}/x_2,\ \{M\}_k/x_3]$. First, the crucial point is that $(\sigma, \sigma) \vdash B \approx B^M_{spec}$ (use twice Lemma 5.4 and (C-Inp)). Next, note that B, B^M_{spec}, A, and S are all σ-safe (use Lemma 5.4 again). Thus we can compose these processes and proceed like in the case of secrecy to obtain (7).

6. A calculus with pairs. Our most relevant omission from the calculus of Abadi and Gordon has been *pairing*, that is, the possibility of transmitting pairs of messages of the form $\langle M_1, M_2\rangle$. It is, however, easy to extend our theory to a calculus with pairs: The necessary modifications are reported below.

The syntax of messages and of expressions is extended by introducing appropriate constructors and selectors for pairs:

$$\begin{array}{rcl} M & ::= \ldots & \mid \quad \langle M_1, M_2\rangle, \\ \zeta & ::= \ldots & \mid \quad \pi_1(\zeta) \mid \pi_2(\zeta) \mid \langle \zeta_1, \zeta_2\rangle\,. \end{array}$$

The evaluation functions for expressions and formulae are extended accordingly (for example, $\pi_1(\widehat{\{a\}_k}) = \bot$). The definition of $dc(\cdot)$ is extended with the clause: if $\langle M_1, M_2\rangle \in dc(S)$, then $M_1, M_2 \in dc(S)$. The definition of *core* needs to be revised; informally, a message M can now have several cores, which are found at different positions inside M. If we code a position inside M as a string $p \in \{l, r\}^*$ (that is, a path through the nested pairs of M), then the core of M at position p with respect to σ, written $M[\sigma, p]$, can be formally defined as follows by induction on M:

- $a[\sigma, p] = \begin{cases} a & \text{if } p = \epsilon, \\ \bot & \text{otherwise,} \end{cases}$
- $\{M\}_k[\sigma, p] = \begin{cases} \{M\}_k & \text{if } k \notin kn(\sigma) \text{ and } p = \epsilon, \\ M[\sigma, p] & \text{if } k \in kn(\sigma), \\ \bot & \text{otherwise,} \end{cases}$
- $\langle M_1, M_2\rangle[\sigma, p] = \begin{cases} M_1[\sigma, p'] & \text{if } p = lp', \\ M_2[\sigma, p'] & \text{if } p = rp', \\ \bot & \text{otherwise.} \end{cases}$

As an example, consider message $M = \langle\langle\{b\}_{hk}, \{c\}_k\rangle, k\rangle$ and substitution $\sigma = [^{M}/_{x}, ^{\{h\}_d}/_{y}]$. We have $dc(\sigma) = \{M, M', k, \{h\}_d, \{b\}_{hk}, \{b\}_h, \{c\}_k, c\}$, where $M' = \langle\{b\}_{hk}, \{c\}_k\rangle$, and $kn(\sigma) = \{k, c\}$. Moreover, we have that $M[\sigma, r] = k$, $M[\sigma, ll] = \{b\}_h$, $M[\sigma, lr] = c$, and $M[\sigma, p] = \bot$ for $p \notin \{r, ll, lr\}$. Note also that the same would hold for every substitution σ' such that $kn(\sigma) = kn(\sigma')$.

Every core in $\sigma = [^{M_i}/_{x_i}]_{i\in I}$ is now determined by an index pair (index i, position p) which we write as ip. The following notation is useful: given a function f defined over $I \times \{l, r\}^*$ and a tuple of index pairs $\widetilde{\jmath} = (i_1p_1, \ldots, i_kp_k)$, we let $f[\widetilde{\jmath}]$ denote the tuple $(f_{i_1p_1}, \ldots, f_{i_kp_k})$ (we write f_{xy} instead of $f(x, y)$ for function application). We can now give the new definition of $\sim$.

Definition 6.1 (equivalence on environments: pairing). *Consider two substitutions $\sigma_1 = [^{M_i}/_{x_i}]_{i\in I}$ and $\sigma_2 = [^{M'_i}/_{x_i}]_{i\in I}$. Let $\widetilde{N}, \widetilde{N}' : I \times \{l, r\}^* \to \mathcal{M} \cup \{\bot\}$ be the functions defined as $N_{ip} \stackrel{\text{def}}{=} M_i[\sigma_1, p]$ and $N'_{ip} \stackrel{\text{def}}{=} M'_i[\sigma_2, p]$ for each index pair ip. We write $\sigma_1 \sim' \sigma_2$ if and only if for each $i \in I$ the following three conditions hold:*

(a) $(\sigma_1, \sigma_2) \vdash M_i \sim M'_i$;

(b) *for each p, $N_{ip} \in \mathcal{N}$ if and only if $N'_{ip} \in \mathcal{N}$;*

(c) *for each p, $j \in I$, and q, $N_{ip} = N_{jq}$ if and only if $N'_{ip} = N'_{jq}$,*

where the predicate $(\sigma_1, \sigma_2) \vdash M \sim M'$ (a recursive version of condition (a) in the old definition) is defined by induction on M as follows:

> *$(\sigma_1, \sigma_2) \vdash M \sim M'$ if and only if there is a tuple $\widetilde{\jmath}$ of index pairs such that $M = \{M_0\}_{\widetilde{N}[\widetilde{\jmath}]}$ and $M' = \{M'_0\}_{\widetilde{N}'[\widetilde{\jmath}]}$ for some M_0, M'_0 such that either* (i) *$M_0 = N_{ip}$ and $M'_0 = N'_{ip}$ for some i and p, or* (ii) *$M_0 = \langle M_1, M_2\rangle$, $M'_0 = \langle M'_1, M'_2\rangle$, and $(\sigma_1, \sigma_2) \vdash M_j \sim M'_j$ for $j = 1, 2$.*

Note that the new definition of $\sim'$ is still effective, because for each $i \in I$ there are finitely many p's s.t. $N_{ip} \neq \bot$ (not more than $|M_i|$). With the new definitions, the results we obtained in the previous sections carry over smoothly, modulo a few notational changes. For instance, the crucial Lemma 5.4 now enjoys the more compact formulation that takes advantage of the predicate $(\sigma_1, \sigma_2) \vdash M \sim M'$:

> Suppose that $\sigma_1 \sim \sigma_2$ and that $n(\zeta) \subseteq dom(\sigma_1)$. Then either $\widehat{\zeta\sigma_1} = \widehat{\zeta\sigma_2} = \bot$ or $(\sigma_1, \sigma_2) \vdash \widehat{\zeta\sigma_1} \sim \widehat{\zeta\sigma_2}$,

The changes in the other statements and proofs are obvious and omitted, with the exception of the construction of ϕ_σ, which is given in Appendix C.

EXAMPLE 6.2. *The following example is used by Abadi and Gordon to discuss the incompleteness of their proof technique for cryptographic protocols,* framed bisimulation [6]. *Processes P and Q defined below are not equated by framed bisimilarity, but Abadi and Gordon conjecture that they are barbed congruent (hence barbed- and testing-equivalent). Here, we indeed prove that P and Q are barbed equivalent: this fact confirms that framed bisimulation is not complete with respect to barbed equivalence.*

Fix some name 0 *and define (writing $\{A, B\}_c$ instead of $\{\langle A, B\rangle\}_c$)*

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & (\nu\, k, k_{01})\ \ \overline{c}\{k_{01}\}_k.\, c(x).P', \\ Q & \stackrel{\text{def}}{=} & (\nu\, k, k_0, k_1)\ \ \overline{c}\{k_0, k_1\}_k.\, c(x).Q', \end{array}$$

where

$$\begin{array}{rcl} P' & \stackrel{\text{def}}{=} & [x = 0]\overline{c}k_{01} \ \mid\ [x \neq 0]\overline{c}k_{01}, \\ Q' & \stackrel{\text{def}}{=} & [x = 0]\overline{c}k_0 \ \mid\ [x \neq 0]\overline{c}k_1, \end{array}$$

for fresh and distinct k, k_0, k_1, and k_{01}. The difference between P and Q is that P discloses a single secret k_{01}, whereas Q may disclose either secret k_0 or secret k_1, but not both. The environment cannot detect this difference, because key k, under which the first message is encrypted, is never disclosed. To prove this, take $V = fn(P, Q)$ and let $\sigma_1 \stackrel{\text{def}}{=} \epsilon_V[\{k_{01}\}_k/y]$ and $\sigma_2 \stackrel{\text{def}}{=} \epsilon_V[\{k_0, k_1\}_k/y]$. Clearly $\sigma_1 \sim \sigma_2$ and, furthermore, for each ζ s.t. $\widetilde{w} \stackrel{\text{def}}{=} n(\zeta) - dom(\sigma_1)$ are fresh names and $\widehat{\zeta\sigma_1} \neq \bot$, we have that

$$(\sigma_1[\widetilde{w}/\widetilde{w}], \sigma_2[\widetilde{w}/\widetilde{w}]) \vdash P'[\widehat{\zeta\sigma_1}/x] \ \approx\ Q'[\widehat{\zeta\sigma_2}/x].$$

In fact, it holds both that $\sigma_1[\widetilde{w}/\widetilde{w}][k_{01}/z] \sim \sigma_2[\widetilde{w}/\widetilde{w}][k_0/z]$ and that $\sigma_1[\widetilde{w}/\widetilde{w}][k_{01}/z] \sim \sigma_2[\widetilde{w}/\widetilde{w}][k_1/z]$, for any fresh z. Therefore, whether $\widehat{\zeta\sigma_1}$ is equal to 0 *or not, we can infer the equivalence above. $(\epsilon_V, \epsilon_V) \vdash P \approx Q$ then follows by applying to the equivalence above first law* (C-INP)*, then law* (C-OUT)*, then weakening to discard the y-entry of the two environments, and finally law* (C-RES) *with $(\nu\, k, k_{01})$ on the left-hand side and $(\nu\, k, k_0, k_1)$ on the right-hand side. Thus we can conclude that $P \cong Q$.*

7. Final remarks and related work. We have studied contextual equivalences and relative proof techniques for a variant of the spi-calculus, an extension of the π-calculus proposed by Abadi and Gordon [7]. We have considered a few examples of verification, concerning secure channels implementation and protocol security, which demonstrate how these techniques can be used in practice.

In this paper, we have applied our techniques to small examples, as the emphasis was more on theory. However, we believe that our methodology can be used to reason on more complex systems. In this respect, a major advantage of our approach is the

possibility of compositional reasoning offered by a set of congruence laws. A further step in this direction would be the design of a sound and complete proof system for the considered equivalences. Another direction for future research is the study of algorithms for mechanical equivalence checking (especially in the case of bisimilarity). We are also considering extensions of the theory to public keys and digital signatures. A subtle point that remains to be solved is in connection with the so-called "known plaintext attack"; for instance, knowing a, b, and a public key k^+, an attacker could distinguish by comparison $\{b\}_{k^+}$ from $\{a\}_{k^+}$, even without knowing the private key k^- that "opens" these messages. As a consequence, and in contrast with the shared key case, two environments like $\sigma = [a/x, \{a\}_{k^+}/y]$ and $\sigma' = [a/x, \{b\}_{k^+}/y]$ cannot be regarded as equivalent.

The relevance of may-testing to the analysis of security properties was first pointed out by Abadi and Gordon in [7]. May-testing was originally introduced for CCS in [11], and subsequently studied for the π-calculus in [8].

Two papers closely related to our work are [10] and [6]. In [10], Boreale and Sangiorgi introduce an lts for a typed version of π-calculus, where the environment's input/output capabilities on names are explicitly described and updated. Here, we use a similar approach to model the environment's knowledge about names and keys.

Abadi and Gordon present in [6] *framed bisimulation*, a proof technique to analyze cryptographic protocols. In framed bisimulation, when comparing two processes P and Q, a *frame-theory pair* (fr, th) is used to represent the knowledge of P's and Q's environments. A judgement $(fr, th) \vdash M \leftrightarrow N$ is also introduced to express that the effect of message M on P's environment is the same as the effect of message N on Q's environment. The judgement is used to check indistinguishability of messages M and N that are exchanged by P and its environment and by Q and its environment. In our case, the indistinguishability of M and N is guaranteed by requiring matching transitions to exhibit the same environment action and to take equivalent environments into equivalent environments. This results in a relevant difference between the work in [6] and ours when considering output transitions. In our case, given an output transition, it is sufficient to check (like in standard bisimilarity) whether the other configuration can perform a matching output transition. Output transitions are, at least for finite-control processes, finitely many. In the case of [6], one must also build a new frame-theory pair that relates N to M and consistently extends the old one; this might be not completely trivial [12]. Moreover, in [6] there seem to be very few tools for compositional reasoning (congruence laws) and no obvious way of tailoring the "up to" techniques to their setting. Finally, as shown in Example 6.2, framed bisimulation is not complete for barbed equivalence.

The process algebraic approach to cryptographic protocols has also been followed by Roscoe [20], Lowe [14], and Schneider [23], all of whom consider model-checking of security protocols in a CSP-based framework. This approach requires explicitly designing a specific (powerful enough) attacker and carrying out the analysis with it. Of course, there is always a certain amount of arbitrariness in determining the attacker; any modification of the attacker would require a new analysis. In our paper, like in [7], a more radical approach is taken: the attacker may be *any* process that can be defined in spi-calculus.

In [2], Abadi presents an approach to secrecy that combines the spi-calculus and the use of type systems: the idea is that a process $P(d)$ that type-checks guarantees secrecy of d (in a sense made precise via testing equivalence).

All the approaches mentioned so far, including ours, work under a perfect en-

cryption hypothesis: this prevents the attacker from, for example, randomly guessing some bits of a secret key, or performing statistical analysis of messages. A first step towards relaxing this hypothesis has been made in [13], where probabilistic versions of the spi-calculus and of testing equivalence are introduced. Further research is required for a fuller understanding of these notions and for devising techniques to reason over them.

Appendix A. Results on environment equivalence. In this appendix we keep the definitions of $\sim$ (Definition 3.1) and of $\sim'$ (Definition 3.3) separate and introduce the notion of characteristic formula, ϕ_σ. The main steps taken here are

1. ϕ_σ characterizes all and only those environments $\sim'$-equivalent to σ (Theorem A.11);
2. using 1, one proves that $\sim$ and $\sim'$ coincide (Theorem 3.4/Theorem A.13);
3. hence, ϕ_σ characterizes all and only those environments $\sim$-equivalent to σ (Theorem 4.3/Theorem A.14).

Along the way, a few properties of $\sim$ are established which are of independent interest and are useful for the examples of section 5 (Lemma 5.3 and Lemma 5.4).

First we show that evaluation $\widehat{\cdot}$ commutes with substitution.

LEMMA A.1. *Let ζ and η be two expressions and σ a substitution. If $x \notin dom(\sigma)$, $n(\zeta) \subseteq dom(\sigma) \cup \{x\}$, and $n(\eta) \subseteq dom(\sigma)$, then $\zeta\sigma\widehat{[\widehat{\eta\sigma}/x]} = (\zeta\widehat{[\eta/x]})\sigma$.*

Proof. The proof follows from a straightforward induction on ζ. □

The following lemma establishes a first relationship between ζ-expressions and the decryption closure $dc(\cdot)$, and between the knowledge $kn(\cdot)$ and the set of *cores* of σ.

LEMMA A.2. *Let $\sigma = [M_i/x_i]_{i\in I}$.*

(1) *If $M \in dc(\sigma)$, then there is ζ s.t. $n(\zeta) \subseteq dom(\sigma)$ and $\widehat{\zeta\sigma} = M$.*

(2) *If $a \in kn(\sigma)$, then $a = core(\sigma, x_i)$ for some $i \in I$.*

Proof. Part (1) is easily proven by induction on the definition of $dc(\cdot)$. Part (2) follows from the following statement, easily proven by induction on the definition of $dc(\cdot)$: If $M \in dc(\sigma)$, then there are $i \in I$ and $\widetilde{k}$ s.t. $M = \{core(\sigma, x_i)\}_{\widetilde{k}}$. □

We proceed by showing that the knowledge $kn(\sigma)$ of σ is precisely the set of those names that can be obtained by arbitrary combinations of encryption and decryption operations (represented by ζ-expressions), starting from messages stored in σ.

LEMMA A.3 (see Lemma 5.3). *Let σ be a substitution. Then $kn(\sigma) = \{\widehat{\zeta\sigma} \in \mathcal{N} : n(\zeta) \subseteq dom(\sigma)\}$.*

Proof. That $kn(\sigma) \subseteq \{\widehat{\zeta\sigma} \in \mathcal{N} : n(\zeta) \subseteq dom(\sigma)\}$ is a consequence of Lemma A.2(1). We now prove the opposite inclusion. Consider the set $S \stackrel{\text{def}}{=} \{\widehat{\zeta\sigma} \neq \bot : n(\zeta) \subseteq dom(\sigma)\}$ and the set $T \stackrel{\text{def}}{=} \{\{M\}_{\widetilde{k}} : M \in dc(\sigma) \text{ and } \widetilde{k} \subseteq kn(\sigma)\}$. We prove that $S \subseteq T$, from which the thesis follows, because $S \cap \mathcal{N} = \{\widehat{\zeta\sigma} \in \mathcal{N} \mid n(\zeta) \subseteq dom(\sigma)\}$ and $T \cap \mathcal{N} = kn(\sigma)$. The proof proceeds by induction on the structure of ζ. We explicitly show only the case $\zeta = \mathtt{dec}_{\zeta_1}(\zeta_2)$, as the other cases are similar or easier. Let $\widehat{\zeta\sigma} = M$. Since $\widehat{\zeta\sigma} \neq \bot$, there is a name k such that $\widehat{\zeta_2\sigma} = \{M\}_k$ and $\widehat{\zeta_1\sigma} = k$. By induction, we can assume that $\{M\}_k \in T$, which, by definition of T, implies that there are $M' \in dc(\sigma)$ and $\widetilde{h} \subseteq kn(\sigma)$ such that $\{M\}_k = \{M'\}_{\widetilde{h}}$. Hence $M = \{M'\}_{\widetilde{h}'}$, for $\widetilde{h}'$ such that $\widetilde{h} = \widetilde{h}'k$, and the thesis follows. □

The next lemma is about the effect of applying two $\sim'$-equivalent substitutions σ and σ' onto the same expression ζ.

LEMMA A.4 (see Lemma 5.4). *Let $\sigma = [M_i/x_i]_{i\in I}$ and $\sigma' = [M'_i/x_i]_{i\in I}$ be two*

substitutions such that $\sigma \sim' \sigma'$. *Define* $\widetilde{N} = core(\sigma, x_i)_{i\in I}$ *and* $\widetilde{N}' = core(\sigma', x_i)_{i\in I}$. *For each* ζ *s.t.* $n(\zeta) \subseteq \widetilde{x}$, *either*

(a) $\widehat{\zeta\sigma} = \widehat{\zeta\sigma'} = \bot$, *or*

(b) *there are* $i \in I$ *and a tuple* $\widetilde{\jmath} \subseteq I$ *such that* $\widehat{\zeta\sigma} = \{N_i\}_{\widetilde{N}[\widetilde{\jmath}]}$ *and* $\widehat{\zeta\sigma'} = \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath}]}$.

Proof. The proof proceeds by induction on ζ. We explicitly consider only the case $\zeta = \mathtt{dec}_{\zeta_2}(\zeta_1)$; the other cases are similar or easier. If $\widehat{\zeta_1\sigma} = \bot$ or $\widehat{\zeta_2\sigma} = \bot$, then by induction hypothesis it easily follows that $\widehat{\zeta\sigma} = \widehat{\zeta\sigma'} = \bot$. Otherwise, by induction hypothesis we have that for some tuples $\widetilde{\jmath}, \widetilde{\ell} \subseteq I$ and indices $i, j \in I$

$$\begin{array}{rclcrcl} \widehat{\zeta_1\sigma}, & = & \{N_i\}_{\widetilde{N}[\widetilde{\jmath}]}, & & \widehat{\zeta_1\sigma'}, & = & \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath}]}, \\ \widehat{\zeta_2\sigma} & = & \{N_j\}_{\widetilde{N}[\widetilde{\ell}]}, & & \widehat{\zeta_2\sigma'} & = & \{N'_j\}_{\widetilde{N}'[\widetilde{\ell}]}. \end{array}$$

There are two cases:

- $\widetilde{\ell} \neq \emptyset$ or $N_j \notin \mathcal{N}$. Then by definition $\widehat{\zeta\sigma} = \widehat{\zeta\sigma'} = \bot$ (note that $N_j \notin \mathcal{N}$ implies $N'_j \notin \mathcal{N}$, because $\sigma \sim' \sigma'$).
- $\widetilde{\ell} = \emptyset$ and $N_j = a \in \mathcal{N}$. Hence $N'_j = a' \in \mathcal{N}$, because $\sigma \sim' \sigma'$. Now, if the last component of $\widetilde{\jmath}$ is some j' s.t. $N_{j'} = a$, say $\widetilde{\jmath} = (\widetilde{\jmath}', j')$, then it is also $N'_{j'} = a'$, because $\sigma \sim' \sigma'$; thus $\widehat{\zeta\sigma} = \{N_i\}_{\widetilde{N}[\widetilde{\jmath}']}$ and $\widehat{\zeta\sigma'} = \{N'_i\}_{\widetilde{N}'[\widetilde{\jmath}']}$, which is the desired claim for this case. Otherwise, $\widehat{\zeta\sigma} = \widehat{\zeta\sigma'} = \bot$. ∎

The intuition underlying the following lemma is that $\sim'$-equivalence is preserved when uniformly adding entries to two equivalent environments.

LEMMA A.5. *If* $\sigma_1 \sim' \sigma_2$, *then* $\sigma_1[\widehat{\zeta\sigma_1}/y] \sim' \sigma_2[\widehat{\zeta\sigma_2}/y]$, *provided that* $\widehat{\zeta\sigma_1} \neq \bot$ *and that* $n(\zeta) \subseteq dom(\sigma_1)$.

Proof. Apply Lemma 5.4 to σ_1, σ_2, and ζ. The thesis then follows by definition of $\sim'$. ∎

We can now prove that $\sim'$ implies $\sim$.

LEMMA A.6. *Let* σ *and* σ' *be two substitutions. If* $\sigma \sim' \sigma'$, *then* $\sigma \sim \sigma'$.

Proof. Suppose that $\sigma \sim' \sigma'$. We must show that for each ϕ with $fn(\phi) \subseteq dom(\sigma)$ it holds that $\sigma \models \phi$ if and only if $\sigma' \models \phi$. The proof proceeds by induction on ϕ. Here, we explicitly consider only the case where ϕ is $\mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi'$, as the other cases are similar or easier. By definition, $\sigma \models \mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi'$ means that $\widehat{\zeta\sigma} \neq \bot$ and that $\sigma[\widehat{\zeta\sigma}/z] \models \phi'$. Now, we have that $\widehat{\zeta\sigma'} \neq \bot$ (Lemma A.4) and $\sigma[\widehat{\zeta\sigma}/z] \sim \sigma'[\widehat{\zeta\sigma'}/z]$ (Lemma A.5). By induction, $\sigma'[\widehat{\zeta\sigma'}/z] \models \phi'$ and we can conclude that $\sigma' \models \mathtt{let}\ z = \zeta\ \mathtt{in}\ \phi'$. ∎

We proceed now to showing the converse implication, that $\sim$ implies $\sim'$. Two crucial ingredients for the proof will be the notion of the *characteristic formula* of an environment σ, ϕ_σ, and Theorem A.11, which will give a logical characterization of $\sim'$. We need some notational shorthand.

NOTATION A.7.

- "$\mathtt{let}_{j\in 1..k}\ z_j = \zeta_j\ \mathtt{in}\ \phi$" *stands for* "$\mathtt{let}\ z_1 = \zeta_1\ \mathtt{in}\ (\cdots(\mathtt{let}\ z_k = \zeta_k\ \mathtt{in}\ \phi)\cdots)$."
- "$\zeta \neq \bot$" *stands for* "$\mathtt{let}\ z = \zeta\ \mathtt{in}\ tt$", *for any* z, *and* "$\zeta = \bot$" *stands for* "$\neg(\zeta \neq \bot)$."
- "$w \in \{M_1, \ldots, M_m\}$" *stands for* "$\bigvee_{j=1}^{m}[w = M_j]$."

DEFINITION A.8 (characteristic formula). *Let* $\sigma = [{}^{M_i}/_{x_i}]_{i\in I}$ *be a substitution. For each* $i \in I$, *let* $N_i = core(\sigma, x_i)$ *and let* ζ_i *be the least*[4] *expression such that*

[4]With respect to some fixed total ordering of expressions.

$$\texttt{let}_{j\in I}\ z_j = \zeta_j\ \texttt{in}\ \Big[$$

$$(*)\quad \bigwedge_{i,j\in I}[\texttt{dec}_{z_j}(z_i) = \bot] \wedge$$

$$(a)\quad \bigwedge_{i\in I}[x_i = \{z_i\}_{\widetilde{z[\mathfrak{j}_i]}}] \wedge$$

$$(b)\quad \bigwedge_{N_i\in\mathcal{N}} name(z_i) \ \wedge\ \bigwedge_{N_i\notin\mathcal{N}} \neg name(z_i) \wedge$$

$$(c)\quad \bigwedge_{i,j\in I \text{ s.t. } N_i=N_j}[z_i = z_j] \ \wedge\ \bigwedge_{i,j\in I \text{ s.t. } N_i\neq N_j}[z_i \neq z_j]$$

$$\Big]$$

FIG. 5. *The formula* ϕ_σ.

$n(\zeta_i) \subseteq \widetilde{x}$ *and* $\widehat{\zeta_i\sigma} = N_i$. *Let* $\widetilde{N} = N_{i\in I}$ *and* $\widetilde{\zeta} = \zeta_{i\in I}$. *Finally, for each* $i \in I$, *let* $\widetilde{\mathfrak{j}_i} \subseteq I$ *be a tuple such that* $M_i = \{N_i\}_{\widetilde{N[\mathfrak{j}_i]}}$. *The formula* ϕ_σ *is then defined as in Figure* 5.

About Definition A.8 we have the following.

REMARK A.9. *Note that the expressions* $\zeta_{i\in I}$ *mentioned in the definition above do exist by virtue of Lemma* A.2(1). *Furthermore, we are allowed to assume that* $M_i = \{N_i\}_{\widetilde{N[\mathfrak{j}_i]}}$, *for some tuple* $\widetilde{\mathfrak{j}_i}$, *because, by virtue of Lemma* A.2(2), $kn(\sigma) \subseteq \widetilde{N}$. *Finally, note that* $fn(\phi_\sigma) \subseteq dom(\sigma)$ *and that* $\sigma \models \phi_\sigma$.

The next lemma gives conditions under which a tuple of messages $\widetilde{Z}$ can be identified as the tuple of "cores" of a given σ.

LEMMA A.10. *Consider* $\sigma = [^{M_i}/_{x_i}]_{i\in I}$ *and a tuple* $\widetilde{Z} = Z_{i\in I}$ *s.t.* $\widetilde{Z} \subseteq dc(\sigma)$. *Suppose that the following two conditions hold:*

(a) *for each* $i, j \in I$, $\widehat{\texttt{dec}_{Z_j}(Z_i)} = \bot$;
(b) *for each* $i \in I$, *there is a tuple* $\widetilde{\mathfrak{j}} \subseteq I$ *s.t.* $M_i = \{Z_i\}_{\widetilde{Z[\mathfrak{j}]}}$.

Then $\widetilde{Z} = core(\sigma, x_i)_{i\in I}$.

Proof. The proof consists of two steps.

- We first show that $kn(\sigma) \subseteq \widetilde{Z}$. To this end, consider the set of messages $T \stackrel{\text{def}}{=} \{N : \{N\}_{\widetilde{k}} = M_i \text{ for some } i \in I \text{ and } \widetilde{k} \subseteq \widetilde{Z}\}$. First, one proves by induction on the definition of $dc(\cdot)$ that $dc(\sigma) \subseteq T$. Then, it is easy to see that $T \cap \mathcal{N} \subseteq \widetilde{Z}$: indeed, if $\{a\}_{\widetilde{k}} = M_i$ for some $\widetilde{k} \subseteq \widetilde{Z}$, then it also holds that $\{a\}_{\widetilde{k}} = \{Z_i\}_{\widetilde{Z[\mathfrak{j}]}}$ (condition (b)), which, by virtue of condition (a), implies $a = Z_i$. Therefore we can conclude $\widetilde{Z} \supseteq kn(\sigma)$.
- Let $\widetilde{N} = core(\sigma, x_i)_{i\in I}$ and take any $i \in I$; we show that $Z_i = N_i$. By definition of *core* and by condition (b), we have that $M_i = \{N_i\}_{\widetilde{h}} = \{Z_i\}_{\widetilde{Z[\mathfrak{j}]}}$ for some tuple $\widetilde{\mathfrak{j}} \subseteq I$ and $\widetilde{h} \subseteq kn(\sigma)$. There are two cases:
  - $Z_i = \{N_i\}_{\widetilde{k}}$ with $(\widetilde{k}, \widetilde{Z}[\widetilde{\mathfrak{j}}]) = \widetilde{h}$. Since $\widetilde{k} \subseteq kn(\sigma)$ (as $\widetilde{h} \subseteq kn(\sigma)$), due to $kn(\sigma) \subseteq \widetilde{Z}$ and condition (a), we deduce that $\widetilde{k} = \emptyset$; hence $Z_i = N_i$.
  - $N_i = \{Z_i\}_{\widetilde{k}}$ with $(\widetilde{k}, \widetilde{h}) = \widetilde{Z}[\widetilde{\mathfrak{j}}]$. Again, we have that $\widetilde{k} \subseteq kn(\sigma)$ (as $\widetilde{k} \subseteq \widetilde{Z} \cap \mathcal{N}$ and $\widetilde{Z} \subseteq dc(\sigma)$); hence, by definition of *core*, we get $\widetilde{k} = \emptyset$, and hence $Z_i = N_i$. □

We arrive at the crucial result on $\sim'$ and characteristic formulae.

THEOREM A.11. *Let σ and σ' be two substitutions such that $dom(\sigma) = dom(\sigma')$. We have that $\sigma \sim' \sigma'$ if and only if $\sigma' \models \phi_\sigma$.*

Proof. The "only if" part is a consequence of Lemma A.6 (as $\sigma \models \phi_\sigma$). Let us see the "if" part. Suppose that $\sigma = [M_i/x_i]_{i\in I}$, and take any $\sigma' = [M'_i/x_i]_{i\in I}$ s.t. $\sigma' \models \phi_\sigma$: we show that $\sigma' \sim \sigma$. With the notation of Figure 5, let $Z_i = \widehat{\zeta_i\sigma'}$ and $\widetilde{Z} = Z_{i\in I}$. From $\sigma' \models \phi_\sigma$, we deduce that $\bot \notin \widetilde{Z}$ and that $\sigma'[\widetilde{Z}/\tilde{z}] \models (*) \wedge$ (a) $\wedge$ (b) $\wedge$ (c). Now, we have that

- $\sigma'[\widetilde{Z}/\tilde{z}] \models (*)$ means that for each $i, j \in I$, $\widehat{\mathtt{dec}_{Z_j}(Z_i)} = \bot$;
- $\sigma'[\widetilde{Z}/\tilde{z}] \models$ (a) means that for each $i \in I$ there is a tuple $\tilde{\jmath} \subseteq I$ s.t. $M'_i = \{Z_i\}_{\widetilde{Z[\tilde{\jmath}]}}$.

The above two facts and Lemma A.10 imply that $\widetilde{Z} = core(\sigma', x_i)_{i\in I}$. Thus $\sigma'[\widetilde{Z}/\tilde{z}] \models$ (a) $\wedge$ (b) $\wedge$ (c) precisely says that $\sigma \sim \sigma'$. □

We can prove that $\sim$ implies $\sim'$.

LEMMA A.12. *Let σ and σ' be substitutions. If $\sigma \sim \sigma'$, then $\sigma \sim' \sigma'$.*

Proof. By definition, $fn(\phi_\sigma) \subseteq dom(\sigma)$ and $\sigma \models \phi_\sigma$. Hence, by hypothesis, $\sigma' \models \phi_\sigma$, and the thesis immediately follows from Theorem A.11. □

The coincidence of $\sim$ and $\sim'$ is now an immediate consequence of Lemma A.6 and of Lemma A.12.

THEOREM A.13 (see Theorem 3.4). *Let σ and σ' be substitutions. Then $\sigma \sim \sigma'$ if and only if $\sigma \sim' \sigma'$.*

The following important property of characteristic formulae is an immediate consequence of Theorem A.11 and of Theorem A.13.

THEOREM A.14 (see Theorem 4.3). *Let σ and σ' be substitutions such that $dom(\sigma) = dom(\sigma')$. We have that $\sigma \sim \sigma'$ if and only if $\sigma' \models \phi_\sigma$.*

We end the section with two technical lemmas. The first lemma is useful for manipulating environments: it says that equivalence is preserved when uniformly removing entries from two equivalent environments. Its proof is an easy consequence of the definition of $\sim$.

LEMMA A.15. *If $\sigma_1[M/y] \sim \sigma_2[M'/y]$, then $\sigma_1 \sim \sigma_2$.*

The second lemma is about the size of characteristic formulae (recall that $rk(\sigma) = |\phi_\sigma|$).

LEMMA A.16 (see Lemma 4.22). *If $\sigma \sim \sigma'$, then $rk(\sigma) = rk(\sigma')$.*

Proof. Referring to the notation of Definition A.8, note that the size of ϕ_σ and $\phi_{\sigma'}$ may only differ due a different choice of some ζ_i, $i \in I$. But Lemma A.4(a) (which ensures that the same ζ_i's can be used in both ϕ_σ and $\phi_{\sigma'}$) and the requirement of minimality imply that the ζ_i's in ϕ_σ and those in $\phi_{\sigma'}$ are actually the same. □

Appendix B. Results on trace and bisimulation semantics. We begin the section with a crucial result on operational semantics.

PROPOSITION B.1 (see Proposition 4.2). *Consider two equivalent substitutions σ_1 and σ_2. Let R be any process or observer s.t. $fn(R) \subseteq dom(\sigma_1)$.*

1. *Suppose that $R\sigma_1 \xrightarrow{(\nu\, \widetilde{b})\overline{a}\langle M\rangle} R_1$. Then* (i) *there is η with $n(\eta) \subseteq dom(\sigma_1)$ s.t. $\widehat{\eta\sigma_1} = a$;* (ii) *there are ζ and R' with $fn(\zeta, R') \subseteq dom(\sigma_1) \cup \widetilde{b}$ s.t. $M = \widehat{\zeta\sigma_1}$, and $R_1 \equiv R'\sigma_1$;* (iii) *it holds that $R\sigma_2 \xrightarrow{(\nu\, \widetilde{b})\overline{a'}\langle M'\rangle} R_2$, where $a' = \widehat{\eta\sigma_2}$, $M' = \widehat{\zeta\sigma_2}$, and $R_2 \equiv R'\sigma_2$.*
2. *Suppose that $R\sigma_1 \xrightarrow{a\, M} R_1$. Then* (i) *there is η with $n(\eta) \subseteq dom(\sigma_1)$ s.t.*

$\widehat{\eta\sigma_1} = a$; (ii) *given any fresh* y *and* $\sigma_1' \stackrel{\text{def}}{=} \sigma_1[M/y]$, *there is* R' *with* $fn(R') \subseteq dom(\sigma_1')$ *s.t.* $R_1 \equiv R'\sigma_1'$; (iii) *for any* M', *it holds that* $R\sigma_2 \xrightarrow{a'\,M'} R_2$, *where* $a' = \widehat{\eta\sigma_2}$ *and* $R_2 \equiv R'\sigma_2'$ *with* $\sigma_2' \stackrel{\text{def}}{=} \sigma_2[M'/y]$.

3. *Suppose that* $R\sigma_1 \xrightarrow{\mu} R_1$ *with* $\mu = \tau$ *or* $\mu = \omega$. *Then* (i) *there is* R' *with* $fn(R') \subseteq dom(\sigma_1)$ *such that* $R_1 \equiv R'\sigma_1$; (ii) *we have* $R\sigma_2 \xrightarrow{\mu} R_2$, *where* $R_2 \equiv R'\sigma_2$.

Proof. The proof follows from an induction on the derivation of $R\sigma \xrightarrow{\mu} R_1$. Here, we explicitly consider only the case where the last rule applied is (LET), which is the most delicate.

Suppose that R is $\texttt{let}\ z = \xi\ \texttt{in}\ P$ and that $R \xrightarrow{\mu} R_1$ is inferred via (LET) from the premise $P\rho_1 \xrightarrow{\mu} R_1$, where $\rho_1 \stackrel{\text{def}}{=} \sigma_1[\widehat{\xi\sigma_1}/z]$ $(\widehat{\xi\sigma_1} \neq \bot)$. Depending on the form of μ, we have three cases; here we consider only the case when μ is an output, say $\mu = (\nu\,\widetilde{b})\overline{a}\langle M\rangle$. Since $\widehat{\xi\sigma_2} \neq \bot$ (Lemma A.4), we can define $\rho_2 \stackrel{\text{def}}{=} \sigma_2[\widehat{\xi\sigma_2}/z]$. Since $\rho_1 \sim \rho_2$ (Lemma A.5), by induction hypothesis applied to the premise $P\rho_1 \xrightarrow{\mu} R_1$ we have that

(j) there is η' with $dom(\eta') \subseteq dom(\rho_1)$ and $\widehat{\eta'\rho_1} = a$;

(jj) there are ζ' and R'' with $fn(\zeta', R'') \subseteq dom(\rho_1) \cup \widetilde{b}$ s.t. $M = \widehat{\zeta'\rho_1}$ and $R_1 \equiv R''\rho_1$;

(jjj) $P\rho_2 \xrightarrow{(\nu\,\widetilde{b})\overline{a'}\langle M'\rangle} R_2$, where $a' = \widehat{\eta'\rho_2}$, $M' = \widehat{\zeta'\rho_2}$, and $R_2 \equiv R''\rho_2$.

Define $\eta \stackrel{\text{def}}{=} \eta'[\xi/z]$, $\zeta \stackrel{\text{def}}{=} \zeta'[\xi/z]$, and $R' \stackrel{\text{def}}{=} \texttt{let}\ z = \xi\ \texttt{in}\ R''$. Then we have that $\widehat{\eta\sigma_1} = a$, $\widehat{\zeta\sigma_1} = M$ (Lemma A.1) and that $R'\sigma_1 \equiv R_1$: this proves parts (i) and (ii) of the claim. Similarly, it holds that $\widehat{\eta\sigma_2} = a'$ and $\widehat{\zeta\sigma_2} = M'$ and that $R'\sigma_2 \equiv R_2$. Part (iii) follows from these equalities and applying (LET) to transition $P\rho_2 \xrightarrow{(\nu\,\widetilde{b})\overline{a'}\langle M'\rangle} R_2$. □

B.1. Trace semantics.

LEMMA B.2 (see Lemma 4.7). *Suppose* $\sigma_1 \sim \sigma_2$ *and let* O *be any observer such that* $fn(O) \subseteq dom(\sigma_1)$. *Suppose that* $O\sigma_1 \stackrel{r}{\Longrightarrow} O_1$, *where, for some* u *and* σ_1' *extending* σ_1, *we have* $r = \widehat{u\sigma_1'}$. *Then, there is* O' *with* $fn(O') \subseteq dom(\sigma_1')$ *s.t.* $O_1 \equiv O'\sigma_1'$. *Furthermore, for any* σ_2' *extending* σ_2 *and s.t.* $\sigma_2' \sim \sigma_1'$, *it holds that* $O\sigma_2 \stackrel{r'}{\Longrightarrow} \equiv O'\sigma_2'$, *with* $r' = \widehat{u\sigma_2'}$.

Proof. The proof consists in iterating the statement of Proposition 4.2. Formally one proceeds by induction on trace u. We only examine the case when $u = u' \cdot \eta(x)$. For any substitution σ, let us write σ^{-x} for the substitution that behaves like σ but is undefined on x. Then we can write $r = \widehat{u'\sigma_1'^{-x} \cdot \eta(x)\sigma_1'} \stackrel{\text{def}}{=} s \cdot aM$ for $\widehat{\eta\sigma_1^{-x}} = a$ and $M = x\sigma_1'$, and similarly $r' = \widehat{u'\sigma_2'^{-x} \cdot \eta(x)\sigma_2'} \stackrel{\text{def}}{=} s' \cdot a'M'$ for $\widehat{\eta\sigma_2^{-x}} = a'$ and $M' = x\sigma_2'$ (recall that, by our convention on bound names, name x does not occur in u'). Applying induction and Proposition 4.2(2) and (3), the sequence of transitions $O\sigma_1 \stackrel{r}{\Longrightarrow} O_1$ can be decomposed as follows, for suitable O'' with $fn(O'') \subseteq dom(\sigma_1'^{-x})$, and O''' and O' with $fn(O''', O') \subseteq dom(\sigma_1')$:

$$O\sigma_1 \stackrel{s}{\Longrightarrow} \equiv O''\sigma_1'^{-x} \xrightarrow{aM} \equiv O'''\sigma_1' \Longrightarrow \equiv O'\sigma_1' \equiv O_1.$$

Similarly,

$$O\sigma_2 \stackrel{s'}{\Longrightarrow} \equiv O''\sigma_2'^{-x} \xrightarrow{a'M'} \equiv O'''\sigma_2' \Longrightarrow \equiv O'\sigma_2'$$

is inferred, again using induction and Proposition 4.2(2) and (3) (note that $\sigma_1^{'-x} \sim \sigma_2^{'-x}$ by Lemma A.15). This implies $O\sigma_2 \stackrel{r'}{\Longrightarrow} \equiv O'\sigma_2'$, which concludes the proof for this case. □

LEMMA B.3 (see Lemma 4.8). *Consider σ, P and any observer O with $fn(O) \subseteq dom(\sigma)$. Suppose that $P \stackrel{s}{\Longrightarrow} P'$ and that $O\sigma \stackrel{r}{\Longrightarrow}$ with $s \underline{compl}\, r$. Then there are u and σ' extending σ such that $r = \widehat{u\sigma'}$ and $\sigma \rhd P \stackrel{s}{\underset{u}{\Longmapsto}} \sigma' \rhd P'$.*

Proof. The proof consists of iterating the following two statements (the first one is an easy consequence of Proposition 4.2):

1. Suppose that $P \stackrel{\mu}{\longrightarrow} P'$ ($\mu \neq \tau$). Suppose that, for some observer O with $fn(O) \subseteq dom(\sigma)$, it holds that $O\sigma \stackrel{\lambda}{\longrightarrow} O_1$, with $\mu \,\underline{compl}\, \lambda$. Then there are σ' extending σ, δ, and O' with $fn(O') \subseteq dom(\sigma')$ s.t. $\sigma \rhd P \stackrel{\mu}{\underset{\delta}{\longmapsto}} \sigma' \rhd P'$ with $\lambda = \widehat{\delta\sigma'}$ and $O_1 \equiv O'\sigma'$.
2. Suppose that $P \stackrel{\tau}{\longrightarrow} P'$. Then $\sigma \rhd P \stackrel{\tau}{\underset{-}{\longmapsto}} \sigma \rhd P'$. □

LEMMA B.4 (see Lemma 4.9). *Suppose that $\sigma \rhd P \stackrel{s}{\underset{u}{\Longmapsto}} \sigma' \rhd P'$. Then it holds that $P \stackrel{s}{\Longrightarrow} P'$, that σ' extends σ, and that $s \,\underline{compl}\, r$, where $r = \widehat{u\sigma'}$.*

Proof. The proof follows by an easy induction on u. □

B.2. Bisimilarity.

PROPOSITION B.5 (see Proposition 4.17). *Let $\mathcal{R}$ be a weak bisimulation up to structural equivalence (resp., weakening, contraction, restriction, parallel composition). Then $\mathcal{R} \subseteq \mathcal{R}_s \subseteq \approx$ (resp., $\mathcal{R} \subseteq \mathcal{R}_t \subseteq \approx$, for $t = w, c, r, p$).*

Proof. It is obvious from the definition that $\mathcal{R} \subseteq \mathcal{R}_t$ for any t. Let us examine the other inclusion. For $t \in \{s, w, c\}$ the proof simply consists in showing that $\mathcal{R}_t$ is a weak bisimulation. This is straightforward by relying on the fact that $(\mathcal{R}_t)_t = \mathcal{R}_t$; in the case $t = c$, we also use the fact that, when $\bot \notin \widehat{\widetilde{\zeta}\sigma}$ and $n(\widetilde{\zeta}) \subseteq dom(\sigma)$, it holds $\sigma[\widehat{\widetilde{\zeta}\sigma}/\widetilde{y}] \rhd P \stackrel{\mu}{\underset{\delta}{\longmapsto}} \sigma'[\widehat{\widetilde{\zeta}\sigma}/\widetilde{y}] \rhd P'$ if and only if $\sigma \rhd P \stackrel{\mu}{\underset{\delta'}{\longmapsto}} \sigma' \rhd P'$, where if δ is an input, then $\delta' = \delta[\widetilde{\zeta}/\widetilde{y}]$; if δ is an output, then $\delta' = (\nu\, \widetilde{b})\delta[\widetilde{\zeta}/\widetilde{y}]$, with $\widetilde{b} = n(\zeta) - dom(\sigma)$, and $\delta' = \delta$ otherwise (the proof of this fact uses Lemma A.1). Next, we examine in more detail the two most interesting cases, up to restriction and parallel composition.

- *Up to restriction.* The proof is a variation on the proof for π-calculus found, e.g., in [22]. In order to cope with the fact that the output of a name which is bound by a restriction (like k in $(\nu\, k)\overline{a}k.P$) gives rise to infinitely many transitions, all of which differ by some injective renaming of this bound name, it is convenient to introduce bisimulation *up to injective renaming*.
  A substitution of names $\rho : \mathcal{N} \to \mathcal{N}$ is *injective on* $V \subseteq \mathcal{N}$ if for each $x, y \in V$, $x\rho = y\rho$ implies $x = y$; given any $\sigma = [M_i/x_i]_{i \in I}$, we denote by $\sigma \circ \rho$ the substitution $[M_i\rho/x_i]_{i \in I}$. We adapt the definition in [22] to our setting and define up to injective renaming as the technique that corresponds to the functional $(\cdot)_{in}$ defined by the rule

$$\frac{(\sigma_1, \sigma_2) \vdash P_1 \,\mathcal{R}\, P_2 \qquad \rho_i \text{ injective on } \cup_{i=1,2}\, n(\sigma_i) \cup fn(P_i)}{(\sigma_1 \circ \rho_1, \sigma_2 \circ \rho_2) \vdash P_1\rho_1 \,\mathcal{R}_{in}\, P_2\rho_2}\,.$$

  It is straightforward to show that if $\mathcal{R}$ is a bisimulation up to injective renaming then $\mathcal{R}_{in} \subseteq \approx$ (this relies on the fact that the both the transition

relation $\xrightarrow{\mu}$ and the relation $\sim$ on environments are preserved by injective renaming). Next, it is easy to show that if $\mathcal{R}$ is a bisimulation up to restriction, then $\mathcal{R}_r$ is a bisimulation up to injective renaming (the proof also uses the fact that $(\mathcal{R}_r)_r = \mathcal{R}_r$); thus we have that $\mathcal{R}_r \subseteq \approx$.

- *Up to parallel composition.* We will show that $\mathcal{R}_p$ is a weak bisimulation up to weakening, restriction, and structural equivalence. Since these techniques have already been proven sound, this implies $\mathcal{R}_p \subseteq \approx$. Suppose that $(\sigma_1, \sigma_2) \vdash A \,\mathcal{R}_p\, B$, with $A \equiv P \mid R\sigma_1$ and $B \equiv Q \mid R\sigma_2$ as given by the definition of $\mathcal{R}_p$. One analyzes the possible transitions from $\sigma_1 \rhd A$ and in each case finds a matching (weak) transition from $\sigma_2 \rhd B$.
  We examine only the most delicate case, which is when P and $R\sigma_1$ interact with each other:

$$\sigma_1 \rhd A \overset{\tau}{\underset{-}{\longmapsto}} \sigma_1 \rhd A',$$

  where $A \equiv P \,|\, R\sigma_1 \xrightarrow{\tau} (\nu\, \widetilde{h})(P' \,|\, R_1) \equiv A'$, $P \xrightarrow{(\nu\, \widetilde{h})\overline{a}\langle M\rangle} P'$, and $R\sigma_1 \xrightarrow{a\, M} R_1$. By virtue of Proposition 4.2(2)(i) applied to $R\sigma_1 \xrightarrow{a\, M} R_1$, there must be some η with $n(\eta) \subseteq dom(\sigma_1)$ s.t. $\widehat{\eta\sigma_1} = a$. Since $(\sigma_1, \sigma_2) \vdash P \,\mathcal{R}\, Q$ and $\mathcal{R}$ is a bisimulation up to parallel composition, it holds that $Q \xRightarrow{(\nu\, \widetilde{k})\overline{a'}\langle M'\rangle} Q'$ with $a' = \widehat{\eta\sigma_2}$ and $(\sigma_1[M/x], \sigma_2[M'/x]) \vdash P' \,\mathcal{R}_p\, Q'$ (x fresh). Moreover, by virtue of Proposition 4.2(2)(iii) applied to $R\sigma_1 \xrightarrow{a\, M} R_1$ and M', there is R' with $fn(R') \subseteq dom(\sigma_1) \cup \{x\}$ s.t. $R_1 \equiv R'\sigma_1[M/x]$ and $R\sigma_2 \xrightarrow{a'\, M'} \equiv R'\sigma_2[M'/x]$. Hence

$$Q \mid R\sigma_2 \equiv B \Longrightarrow B' \equiv (\nu\, \widetilde{k})(Q' \mid R'\sigma_2[M'/x]).$$

  By definition, $(\sigma_1[M/x], \sigma_2[M'/x]) \vdash (P' \mid R'\sigma_1[M/x]) \,\mathcal{R}_p\, (Q' \mid R'\sigma_2[M'/x])$ (here we have used the fact that $(\mathcal{R}_p)_p = \mathcal{R}_p$ for any $\mathcal{R}$); hence, by weakening (to discard the x-entries) and restriction, we get that

$$(\sigma_1, \sigma_2) \vdash (\nu\, \widetilde{h})(P' \mid R'\sigma_1[M/x]) \;((\mathcal{R}_p)_w)_r\; (\nu\, \widetilde{k})(Q' \mid R'\sigma_2[M'/x]).$$

  Finally, by structural equivalence, $(\sigma_1, \sigma_2) \vdash A'(((\mathcal{R}_p)_w)_r)_s B'$, which is the desired claim for this case. □

The following proposition strengthens the congruence rule for parallel composition (C-Par), under the assumption that the involved processes are safe for the appropriate environments. Recall that a process R is *σ-safe* if for each s, whenever $\sigma \rhd R \overset{s}{\underset{u}{\Longmapsto}} \sigma' \rhd R' \overset{(\nu\, \widetilde{b})\overline{a}\langle M\rangle}{\underset{\eta(x)}{\Longmapsto}}$, then $M \in dc(\sigma')$ (hence $\widetilde{b} = \emptyset$).

Proposition B.6 (see Proposition 5.6). *Suppose that $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$ and that $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$. Suppose that, for $i = 1, 2$, Q_i and R_i are σ_i-safe. Then $(\sigma_1, \sigma_2) \vdash Q_1 \mid R_1 \approx Q_2 \mid R_2$.*

Proof. Consider the relation $\mathcal{R}$ defined by

$$(\sigma_1, \sigma_2) \vdash (Q_1 \mid R_1)\ \mathcal{R}\ (Q_2 \mid R_2) \quad \text{if and only if}$$
$$\text{(a) } (\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2 \text{ and } (\sigma_1, \sigma_2) \vdash R_1 \approx R_2,$$
$$\text{(b) for } i = 1, 2,\ Q_i \text{ and } R_i \text{ are } \sigma_i\text{-safe.}$$

One shows that $\mathcal{R}$ is a weak bisimulation. Suppose that $(\sigma_1, \sigma_2) \vdash Q_1 \mid R_1 \,\mathcal{R}\, Q_2 \mid R_2$. We only examine the most delicate case, that is, when Q_1 and R_1 interact with one

another. Thus, suppose that

$$\sigma_1 \rhd Q_1 \mid R_1 \stackrel{\tau}{\longmapsto} \sigma_1 \rhd (\nu\, \widetilde{h})(Q_1' \mid R_1'), \tag{8}$$

where $Q_1 \xrightarrow{(\nu\, \widetilde{h})\overline{a}\langle M\rangle} Q_1'$ and $R_1 \xrightarrow{a\, M} R_1'$ (the other cases follow by symmetry). Note that it must be $\widetilde{h} = \emptyset$, because $M \in dc(\sigma_1)$. Since $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$, we get that $Q_2 \xRightarrow{(\nu\, k)\overline{a'}\langle M'\rangle} Q_2'$, for some a', M', and Q_2', s.t. $(\sigma_1', \sigma_2') \vdash Q_1' \approx Q_2'$ (here $\sigma_1' \stackrel{\text{def}}{=} \sigma_1[M/x]$ and $\sigma_2' \stackrel{\text{def}}{=} \sigma_2[M'/x]$, for a fresh x). Also in this case, $\widetilde{k} = \emptyset$, because $M' \in dc(\sigma_2)$.

Now, since $M \in dc(\sigma_1)$, there is ζ s.t. $n(\zeta) \subseteq dom(\sigma_1)$ and $M = \widehat{\zeta\sigma_1}$ (Lemma A.2(1)). From Lemma A.4, it follows that $\widehat{\zeta\sigma_1} = \{N_i\}_{\widetilde{N}[\widetilde{\jmath}]}$ and that $\widehat{\zeta\sigma_2} = \{N_i'\}_{\widetilde{N}'[\widetilde{\jmath}]}$, for appropriate indices $i, \widetilde{\jmath}$ and cores $\widetilde{N}$ and $\widetilde{N}'$. Hence, from $\sigma_1' \sim \sigma_2'$, we get that $M' = \{N_i'\}_{\widetilde{N}'[\widetilde{\jmath}]}$; hence $M' = \widehat{\zeta\sigma_2}$. Similarly, one finds an η s.t. $n(\eta) \subseteq dom(\sigma_1)$ and $a = \widehat{\eta\sigma_1}$ and $a' = \widehat{\eta\sigma_2}$.

Now, using the facts on M and M' established above, from $R_1 \xrightarrow{a\, M} R_1'$ we deduce $\sigma_1 \rhd R_1 \stackrel{a\, M}{\longmapsto_{\overline{\eta}\zeta}} \sigma_1 \rhd R_1'$; hence, from $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$, we get $\sigma_2 \rhd R_2 \stackrel{a'\, M'}{\Longmapsto_{\overline{\eta}\zeta}} \sigma_2 \rhd R_2'$, where $(\sigma_1, \sigma_2) \vdash R_1' \approx R_2'$. Combining $Q_2 \xRightarrow{\overline{a'}\langle M'\rangle} Q_2'$ and $R_2 \xRightarrow{a'\, M'} R_2'$, we get $Q_2 \mid R_2 \Longrightarrow Q_2' \mid R_2'$; hence

$$\sigma_2 \rhd Q_2 \mid R_2 \stackrel{\tau}{\Longmapsto} \sigma_2 \rhd Q_2' \mid R_2'\,.$$

To see that the above weak transition matches (8), let us check conditions (a) and (b) of the definition of $\mathcal{R}$. Indeed, $(\sigma_1, \sigma_2) \vdash Q_1' \approx Q_2'$ (by $(\sigma_1', \sigma_2') \vdash Q_1' \approx Q_2'$ and weakening) and $(\sigma_1, \sigma_2) \vdash R_1' \approx R_2'$ imply condition (a); moreover, for $i = 1, 2$, R_i' and Q_i' are σ_i-safe, because R_i and Q_i are; thus condition (b) is true. ∎

Appendix C. Characteristic formula: Calculus with pairing.

DEFINITION C.1. *Let $\sigma = [M_i/x_i]_{i\in I}$ be a substitution. For each $i \in I$ and $p \in \{l, r\}^*$, let*

- *N_{ip} be $M_i[\sigma, p]$;*
- *ζ_{ip} be the least expression s.t. $n(\zeta_{ip}) \subseteq \widetilde{x}$ and $\widehat{\zeta_{ip}\sigma} = N_{ip}$;*
- *z_{ip} be some fixed fresh name.*

Let $\widetilde{N}$ be the function that maps each ip to N_{ip}, $\widetilde{\zeta}$ be the function that maps each ip to ζ_{ip}, and $\widetilde{z}$ be the function that maps each ip to z_{ip} ($\widetilde{z}$ will also denote the set of all z_{ip}'s). Let ρ_σ be the substitution that maps each z_{ip} to N_{ip} if $N_{ip} \neq \perp$.

Finally, for any $i \in I$, let M_i^σ be a message s.t. $n(M_i^\sigma) \subseteq \{z_{ip} : p \in \{l, r\}^*\}$ and $(M_i^\sigma)\rho_\sigma = M_i$. The formula ϕ_σ is then defined as in Figure 6.

The existence of expressions ζ_{ip} can be easily proven by relying on (the analogue of) Lemma A.2, while in the case of the M_i^σ, it is sufficient to prove that for each $M \in dc(\sigma)$ there is M' with $n(M') \subseteq \widetilde{z}_{ip}$ s.t. $M'\rho_\sigma = M$ (induction on M). The interested reader can easily supply the details. Note that $fn(\phi_\sigma) \subseteq dom(\sigma)$ and that $\sigma \models \phi_\sigma$.

The proof of Theorem A.11 is easily extended via the following lemma (the analogue of Lemma A.10), where we refer to the notation introduced in Definition C.1.

LEMMA C.2. *Consider $\sigma = [M_i/x_i]_{i\in I}$ and $\sigma' = [M_i'/x_i]_{i\in I}$, and let ρ' be a substitution s.t. $dom(\rho') = dom(\rho_\sigma)$ and $\widetilde{Z} \stackrel{\text{def}}{=} range(\rho') \subseteq dc(\sigma')$. Suppose that the following two conditions hold:*

$$
\begin{array}{ll}
 & \mathtt{let}_{N_{jq}\neq\perp}\ z_{jq} = \zeta_{jq}\ \mathtt{in}\ \Big[\\
(*) & \bigwedge_{N_{ip},N_{jq}\neq\perp} [\mathtt{dec}_{z_{jq}}(z_{ip}) = \perp]\ \wedge [\pi_1(z_{ip}) = \perp]\ \wedge\ [\pi_2(z_{ip}) = \perp] \wedge \\
(a) & \bigwedge_{i\in I} [x_i = M_i^\sigma] \wedge \\
(b) & \bigwedge_{N_{ip}\in\mathcal{N}} name(z_{ip})\ \wedge \bigwedge_{N_{ip}\neq\perp \text{ and } N_{ip}\notin\mathcal{N}} \neg name(z_{ip}) \wedge \\
(c) & \bigwedge_{N_{ip},N_{jq}\neq\perp \text{ and } N_{ip}=N_{jq}} [z_{ip} = z_{jq}]\ \wedge \bigwedge_{N_{ip},N_{jq}\neq\perp \text{ and } N_{ip}\neq N_{jq}} [z_{ip} \neq z_{jq}] \\
 & \qquad\qquad \Big]
\end{array}
$$

FIG. 6. *The formula ϕ_σ (calculus with pairing).*

(a) *for each Z_1, $Z_2 \in \widetilde{Z}$, $\widehat{\mathtt{dec}_{Z_2}(Z_1)} = \perp$ and Z_1 is not a pair;*
(b) *for each $i \in I$, it holds that $M_i' = (M_i^\sigma)\rho'$.*

Then for each i and p s.t. $M_i'[\sigma', p] \neq \perp$, it holds that $M_i'[\sigma', p] = \rho'(z_{ip})$. Furthermore, for each $i \in I$ it holds that $(\sigma, \sigma') \vdash M_i \sim M_i'$.

Proof. We just outline the proof. There are three steps:

- $kn(\sigma') \subseteq \widetilde{Z}$. To prove this, consider the set $T \stackrel{\text{def}}{=} \{M\rho' : n(M) \subseteq dom(\rho')\}$ and show that $dc(\sigma') \subseteq T$ (by induction on the definition of $dc(\cdot)$). The thesis then follows because $T \cap \mathcal{N} \subseteq \widetilde{Z}$.
- For any M and context $C[\cdot]$ (a message with a "hole"), if $C[M] = M_i^\sigma$ and $(M\rho')[\sigma', p] \neq \perp$, then $(M\rho')[\sigma', p] = \rho'(z_{i(qp)})$, where q is the position of the hole $[\cdot]$ inside $C[\cdot]$ (this implies $M_i'[\sigma', p] = \rho'(z_{ip})$ when $C[\cdot]$ is the empty context $[\cdot]$). The proof of this fact proceeds by induction on M and relies, for the base case, on the fact that $kn(\sigma') \subseteq \widetilde{Z}$.
- For any M that is a submessage of M_i^σ, it holds that $(\sigma, \sigma') \vdash M\rho_\sigma \sim M\rho'$ (this implies $(\sigma, \sigma') \vdash M_i \sim M_i'$ when M is M_i^σ, by virtue of hypothesis (b)). The proof of this fact proceeds by induction on M, and relies on the fact that $M_i'[\sigma', p] = \rho'(z_{ip})$, which has been proven above. ∎

Acknowledgments. We would like to thank the anonymous referees for helpful comments. Discussions with Martin Abadi, Cedric Fournet, and Andrew Gordon have helped us to improve the paper.

REFERENCES

[1] M. ABADI, *Protection in programming-language translations*, in Proceedings ICALP'98, Lecture Notes in Comput. Sci. 1443, K.G. Larsen, S. Skyum, and G. Winskel, eds., Springer-Verlag, New York, 1998, pp. 868–883.

[2] M. ABADI, *Secrecy by typing in security protocols*, in Proceedings STACS'97, Lecture Notes in Comput. Sci. 1281, Springer-Verlag, New York, 1997, pp. 611–638.

[3] R. AMADIO, I. CASTELLANI, AND D. SANGIORGI, *On bisimulations for the asynchronous π–calculus*, Theoret. Comput. Sci., 195 (1998), pp. 291–324.

[4] M. ABADI, C. FOURNET, AND G. GONTHIER, *Secure implementation of channel abstractions*, in Proceedings of the 13th IEEE Symposium Logic Computer Science (LICS'98), IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 105–116.

[5] M. ABADI AND A.D. GORDON, *Reasoning about cryptographic protocols in the spi calculus*,

in Proceedings CONCUR'97, A. Mazurkiewicz and J. Winkowsky, eds., Lecture Notes in Comput. Sci. 1243, Springer-Verlag, New York, 1997, pp. 59–73.

[6] M. Abadi and A.D. Gordon, *A bisimulation method for cryptographic protocols*, Nordic J. Comput., 5 (1998), pp. 267–303.

[7] M. Abadi and A.D. Gordon, *A calculus for cryptographic protocols: The spi calculus*, Inform. and Comput., 148 (1999), pp. 1–70.

[8] M. Boreale and R. De Nicola, *Testing equivalence for mobile processes*, Inform. and Comput., 120 (1995), pp. 279–303.

[9] M. Boreale, R. De Nicola, and R. Pugliese, *Proof techniques for cryptographic processes*, in Proceedings of the 14th IEEE Symposium Logic in Computer Science (LICS'99), IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 157–166.

[10] M. Boreale and D. Sangiorgi, *Bisimulation in name-passing calculi without matching*, in Proceedings of the 13th IEEE Symposium Logic in Computer Science (LICS'98), IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 165–175.

[11] R. De Nicola and M.C.B. Hennessy, *Testing equivalence for processes*, Theoret. Comput. Sci., 34 (1984), pp. 83–133.

[12] A.S. Elkjaer, M. Höhle, H. Hüttel, and K.O. Nielsen, *Towards automatic bisimilarity checking in the spi calculus*, in Proceedings of DMTCS'99+CATS'99, Springer-Verlag, Singapore, 1999.

[13] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov, *A probabilistic poly-time framework for protocol analysis*, in Fifth ACM Conference on Computer and Communication Security (CCS-5), San Francisco, CA, 1998, pp. 112–121.

[14] G. Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using fdr*, in Proceedings TACAS'96, T. Margaria and B. Steffen, eds., Lecture Notes in Comput. Sci. 1055, Springer-Verlag, New York, 1996, pp. 147–166.

[15] R. Milner, *Communication and Concurrency*, Prentice–Hall, New York, 1989.

[16] R. Milner, *The polyadic π-calculus: A tutorial*, in Logic and Algebra of Specification, F.L. Hamer, W. Brauer, and H. Schwichtenberg, eds., Springer-Verlag, Berlin, 1993, pp. 203–246.

[17] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes*, I, Inform. and Comput., 100 (1992), pp. 1–41.

[18] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes*, II, Inform. and Comput., 100 (1992), pp. 41–77.

[19] R. Milner and D. Sangiorgi, *Barbed bisimulation*, in Proceedings ICALP'92, W. Kuich, ed., Lecture Notes in Comput. Sci. 623, Springer-Verlag, New York, 1992, pp. 685–695.

[20] A.W. Roscoe, *Modelling and verifying key-exchange using CSP and FDR*, in 8th Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[21] M.C. Sanderson, *Proof Techniques for CCS*, Internal report CST-19-82, Department of Computer Science, University of Edinburgh, UK, 1982.

[22] D. Sangiorgi, *On the bisimulation proof method*, Math. Structures Comput. Sci., 8 (1998), pp. 447–479.

[23] S. Schneider, *Verifying authentication protocols in CSP*, IEEE Transactions on Software Engineering, 24 (1998), pp. 743–758.

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 987–987

CORRIGENDUM: PROXIMITY IN ARRANGEMENTS OF ALGEBRAIC SETS*

J. H. RIEGER†

Abstract. This erratum corrects an error found in [J. H. Rieger, *SIAM J. Comput.*, 29 (1999), pp. 433–458].

PII. S0097539700380201

At present, it is not known whether the singular stratification of a real algebraic hypersurface $\hat{\mathcal{B}} \subset \mathbb{R}^d$ has $O((\deg \hat{\mathcal{B}})^d)$ strata, contrary to the statement in parentheses on the bottom of [1, p. 440].

As a consequence, the upper bounds for the number of connected regions of $\mathbb{R}^d \setminus \mathcal{B}$ and for the size of $\mathcal{A}(\mathcal{B})$ stated in Propositions 2.2, 3.1(iii), and 3.2(iii) are only valid for the number of regions of $\mathbb{R}^d \setminus \mathcal{B}$ but not for the size of $\mathcal{A}(\mathcal{B})$.

REFERENCES

[1] J. H. RIEGER, *Proximity in arrangements of algebraic sets*, SIAM J. Comput., 29 (1999), pp. 433–458.

*Received by the editors October 30, 2000; accepted for publication July 17, 2001; published electronically February 20, 2002.
http://www.siam.org/journals/sicomp/31-3/38020.html
†Institut für Algebra und Geometrie, Martin-Luther-Universität Halle, 06099 Halle (Saale), Germany (rieger@mathematik.uni-halle.de).

SIAM J. COMPUT.
Vol. 31, No. 3, pp. 988–988

CORRIGENDUM: EDGE-DISJOINT PATHS IN EXPANDER GRAPHS*

ALAN FRIEZE†

PII. S0097539701395097

We would like to correct a small error involving some conditioning in the argument of [1]. It arose from trying to oversimplify the algorithm and proof. We are given an expander graph $G = (V, E)$ and κ pairs of vertices a_i, b_i, $i = 1, 2, \ldots, \kappa$, which must be joined by edge-disjoint paths.

As part of the algorithm we partition E into a number of sets E_i, $i = 1, 2, \ldots, k$, such that each $G_i = (V, E_i)$ has some guaranteed expansion properties. The next step is to generate two random sets $\tilde{A}, \tilde{B}$ of κ vertices in V. We then join $A = \{a_1, a_2, \ldots, a_\kappa\}$ to $\tilde{A}$ using a network flow algorithm on the graph G_1 and $B = \{b_1, b_2, \ldots, b_\kappa\}$ to $\tilde{B}$ using a network flow algorithm on the graph G_2. The endpoints of the paths produced are then relabelled $\tilde{a}_i, \tilde{b}_i, i = 1, 2, \ldots, \kappa$. This constitutes Phase 1. The task now is to join the random pairs $\tilde{a}_i, \tilde{b}_i$, $i = 1, 2, \ldots, \kappa$, by edge-disjoint paths using graphs G_i, $i \geq 3$.

The paper then claims it is sufficient to (try to) join $\tilde{a}_i$ to $\tilde{b}_i$ for $i = 1, 2, \ldots, \kappa$ by a shortest path in G_3. If we fail to find a path we put the pair into a set L which will be dealt with later. The pairs are considered in this order, and it is crucial for the analysis that when we come to consider $\tilde{a}_i$ or $\tilde{b}_i$ this vertex can be considered to be uniformly chosen from some large set, i.e., of size order n, *independent of the previous history.* This is true looking at each sequence $\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_\kappa$ and $\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_\kappa$ individually, and we tried to argue that this was sufficient:

"It is important to note here that regardless of the history of Phase 1, $\tilde{A}$ is a random set and we can assume that $\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_\kappa$ is a random ordering of its elements. (Phase 1 may cause some correlation between the orderings of $\tilde{A}, \tilde{B}$, but we compute the contribution of each set separately.)"

This is not actually true. The problem is that knowing $\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_i, \tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_{i-1}$ we do learn something about the distribution of $\tilde{b}_i$.

Fortunately, there is a simple way of dealing with this problem. We can generate $x_1, x_2, \ldots, x_\kappa$ uniformly at random from V and then (try to) join $\tilde{a}_i$ to x_i, $i = 1, 2, \ldots, \kappa$, using the graph G_3 and then (try to) join $\tilde{b}_i$ to x_i, $i = 1, 2, \ldots, \kappa$, using the graph G_4. We can then analyze the joining of the $\tilde{a}_i$'s to the x_i's separately from the joining of the $\tilde{b}_i$'s to the x_i's and add up the separate contributions to the set L. It is now the case that when considering, say, the pairs $\tilde{a}_i, x_i$ in sequence, they are independently chosen from large sets.

REFERENCE

[1] A. M. FRIEZE, *Edge-disjoint paths in expander graphs,* SIAM J. Comput., 30 (2001), pp. 1790–1801.

*Received by the editors September 10, 2001; accepted for publication (in revised form) September 12, 2001; published electronically February 20, 2002.
http://www.siam.org/journals/sicomp/31-3/39509.html
†Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA 15213 (alan@random.math.cmu.edu).

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 989–1021

A LAYERED ANALYSIS OF CONSENSUS*

YORAM MOSES† AND SERGIO RAJSBAUM‡

Abstract. This paper introduces a simple notion of *layering* as a tool for analyzing well-behaved runs of a given model of distributed computation. Using layering, a model-independent analysis of the consensus problem is performed and then applied to proving lower bounds and impossibility results for consensus in a number of familiar and less familiar models. The proofs are simpler and more direct than existing ones, and they expose a unified structure to the difficulty of reaching consensus. In particular, the proofs for the classical synchronous and asynchronous models now follow the same outline. A new notion of connectivity among states in runs of a consensus protocol, called *potence connectivity*, is introduced. This notion is more general than previous notions of connectivity used for this purpose and plays a key role in the uniform analysis of consensus.

Key words. distributed systems, shaved-memory systems, topology, consensus, impossibility results, lower bounds

AMS subject classifications. 68Q10, 68Q22, 68Q25, 68R05, 68R10

PII. S0097539799364006

1. Introduction.

For almost two decades now, the consensus problem has played a central role in the study of fault-tolerant distributed computing, e.g., [33, 18, 19, 15, 20, 29, 22, 9, 13]. It has clearly received the greatest amount of attention of any problem treated in the theoretical literature on distributed computing and has been studied in a large variety of models and under many types of failure assumptions. The structure of the consensus problem in different settings is frequently based on closely related notions. However, proofs for different models are often based on distinct and somewhat ad hoc techniques. In particular, there have been considerable differences between the study of consensus in asynchronous models and its study in synchronous and partially synchronous ones.

In order to cope with the proliferation of distributed computing models, researchers have proposed a variety of simulations between models. The aim is to establish a relation (often of equivalence) between the possibility of solving problems of certain types in different models. This is used to establish impossibility results in particular models or to provide a systematic way to transform protocols written in one model into protocols for another model. Various such simulations have been given, e.g., between shared memory and message passing [3]; between snapshot shared memory and read/write shared memory [1]; between immediate snapshot shared memory and read/write shared memory [10, 11]; between synchronous and asynchronous message passing [6]; and between two shared-memory models of different resilience [12].

This paper attempts to present a uniform approach to the study of solvability of consensus in various models of computation in which, intuitively, crash failure

*Received by the editors November 8, 1999; accepted for publication (in revised form) July 6, 2001; published electronically March 13, 2002. This paper is the first of two parts. A preliminary version reporting results of these papers appeared in the *Proceedings of the* 17*th ACM Symposium on Principles of Distributed Computing (PODC)*, 1998, pp. 123–132.

http://www.siam.org/journals/sicomp/31-4/36400.html

†Dept. of Electrical Engineering, Technion, Israel (yoram@ee.technion.ac.il). This author was supported in part by an ATS grant, and by a grant from the fund for advancement of research.

‡Instituto de Matemáticas, UNAM Ciudad Universitaria, D.F. 04510, México (rajsbaum@math.unam.mx). On leave at Compaq Cambridge Research Laboratory, One Cambridge Center, Cambridge, MA 02142 (Sergio.Rajsbaum@compaq.com). This author was partially supported by Conacyt and DGAPA-UNAM projects.

behavior can occur. That is, where a process can be silenced from some point on in an execution and thus appear as if it has crashed. The standard forms of malicious (Byzantine) faults [33], in which faulty processes may behave in an arbitrary manner, satisfy the crashlike behavior condition, as do models of omission failures and, of course, the usual models of crash failures. Our results apply even to situations in which there are only link failures [35], provided there are sufficiently many of them to enable the environment to silence a process.

We start by presenting an alternative proof of the impossibility of consensus in the asynchronous shared-memory model treated by Loui and Abu-Amara [29]. This proof is based on a new notion of connectivity which we call *potence connectivity* and on an analysis based on a nicely structured subset of the runs. This subset consists of runs that are obtained by imposing a round-by-round *layering* structure on the model. Roughly, we use layerings in the following sense. Given a model of distributed computation, we identify particular legal sequences of actions for the scheduler, each of which we think of as generating a "layer." We require that any way of performing such layers repeatedly starting from a legal initial state will result in a legal run in the model. Thus, in a precise sense, such a layering can be viewed as defining a submodel of the original model.[1] Any protocol for the original model translates directly to one in the submodel. Moreover, the model and submodel generally share many features. In particular, lower bounds and impossibility results proven for the submodel translate directly into the original model. The use of layerings facilitates performing round-by-round analysis: Almost all of our results regarding consensus will follow from analyzing a single layer of computation.

The benefit of working in a submodel or a set of runs with a simpler structure than that of the original model is well known; some recent examples are [4, 10, 11, 9, 27, 34].

In this paper we concentrate on proving lower bounds and impossibility results for the consensus problem; in a sequel paper (as briefly described in [32]) we show how the ideas of this paper extend naturally to other decision tasks and are useful also to prove solvability results. First, we perform an abstract and model-independent analysis of consensus using layering. Based on this analysis, implications for specific models are obtained by demonstrating that appropriate layerings can be defined in the model. We exemplify this approach by applying it to the following models: Shared-memory asynchronous model with one crash failure, message-passing asynchronous model with one crash failure, message-passing synchronous model with one mobile failure, and message-passing synchronous model with t crash failures.

For the asynchronous models, we describe two styles of layerings: the *synchronic* and the *permutation* layering. The synchronic layerings we consider define submodels of the asynchronous models that are very close in structure to being synchronous, thereby defining "almost synchronous" submodels of the asynchronous models. Indeed, we show that synchronic layerings can be applied to the synchronous message-passing model too. The permutation layering is inspired by the immediate snapshot wait-free model of [10, 34], although we define it both for the message-passing and for the shared-memory models, both 1-resilient. This appears to be the first variant of the immediate snapshot model suggested for a message-passing model.

Regarding consensus, we provide the following:

- New, simple and uniform lower bound and impossibility proofs for the standard synchronous and asynchronous models. In particular, we show a simple

[1] Layering saves us the trouble of explicitly defining the submodel as a model of computation with a new transition function, new actions, etc.

bivalence-style proof for the synchronous case, and a direct round-by-round construction of a bivalent run (not employing a critical state argument) in asynchronous cases.

- Stronger impossibility results with respect to submodels of the full asynchronous models, in which there is only a small degree of asynchrony. These demonstrate how little asynchrony is needed to make consensus unsolvable.

In a sequel paper we show that the asynchronous models are equivalent in terms of the 1-resilient solvability of decision tasks. In particular, the slightly asynchronous submodels are no more powerful than the fully asynchronous ones. Moreover, in a precise sense, these models are strictly stronger than what can be done t-resiliently in t rounds of the standard synchronous model.

We consider the layering technique to be useful in a number of ways:

- It provides a tool for performing model-independent round-by-round analysis of decision tasks and related problems.
- Results are obtained directly and not by means of specially tailored reductions.
- Popular topological treatments (e.g., [25, 10, 26, 34]) focus on the local *final* states of processes. We consider states at intermediate stages of the computation as well. Moreover, the state of the environment is represented as an explicit component in the global state, which facilitates the treatment of message-passing as well as shared-memory models.
- We make use of a novel notion of *potence connectivity* of a set of states, whose definition depends on the decisions taken by the protocol in the possible futures of these states. In addition we use the more traditional notion of connectivity based on indistinguishability of states. The combined use of the two notions proved very useful for unifying the analysis of consensus in the synchronous and asynchronous models.

Our analysis in this paper concentrates on the consensus problem. By focusing on this "basic" case we obtain a direct and uniform analysis in simple and elementary terms. The proofs of all of our results are short and rather straightforward, which suggests to us that the notions we use are fairly natural. We believe that, with small modifications, the same type of analysis and style of reasoning can be applied to study more general problems involving topological connectivity of higher dimensionality (e.g., [25]).

There are two other papers that attempt to unify synchronous and asynchronous models in a round-by-round style. The research project of [21], concurrent and independent to ours, is based on a notion of *fault detectors*. Then there is the work of [24], which uses topological techniques in synchronous, asynchronous, and partially synchronous message-passing systems, with applications to set-consensus. There are two other papers that independently discovered bivalence arguments for the synchronous consensus lower bound: in the randomized setting there is [7], while in the deterministic setting (same as our application in section 7.2) there is [2]. A different abstract model based on the runs of a distributed system is proposed in [30]. Recently our layering technique was used to prove a synchronous lower bound for uniform consensus [28].

This paper is organized as follows. In section 2 we define the consensus problem and the basic elements of a distributed computing model, such as processes, environment (which can be used to model different communication mechanisms), states, actions, runs, and failures. In section 3 we illustrate the layering approach in the

concrete setting of an asynchronous shared-memory system by proving the impossibility of consensus. The notion of potence of a state is introduced here. This section provides a complete example of the use of our ideas; a reader uninterested in the full generality of the approach developed in the remaining of the paper could stop here. In section 4 we present a generic framework for defining models of a distributed system. As an example, we show how the synchronous mobile failures model is captured in our abstract framework. In section 5 we develop the generic setting for using potence connectivity to study consensus in the presence of crashlike failure behavior. In addition, the connection between potence connectivity and earlier notions of connectivity is formalized and proven. In section 6 we introduce layering in the generic setting, and its basic properties. In section 7.1 we illustrate the ideas with an impossibility result in the mobile failures (synchronous) model, demonstrating that asynchrony is not necessary in order to make consensus impossible. In section 7.2 we apply our framework to provide a new proof for the classic synchronous message-passing model. More applications are presented in section 8 for asynchronous systems, where various particular layerings are described. The conclusions are in section 9.

2. Preliminaries.

2.1. Consensus. In the consensus problem, we start out in an initial state in which the local state of a process i consists of a binary *initial value* variable $\mathtt{v}_i \in \{0,1\}$ and an undefined write-once *decision* variable $\mathtt{d}_i = \perp$. All communication channels (if any exist) are empty, and shared variables (if any exist) all have an undefined value of $\perp$. We denote the set of (all 2^n) initial states of consensus by Con_0. A protocol for consensus is a deterministic protocol D all of whose runs satisfy the following three properties:

Agreement: All nonfaulty processes reach the same decision.

Decision: Every nonfaulty process irrevocably *decides* on a value.

Validity: In runs in which all processes start with the same initial value w, the value that the nonfaulty processes decide on is w.

As presented, the consensus problem is well defined only once we have provided a model of computation. In this model, we must, in particular, define the structure of local and global states, as well as what protocols are and how they generate runs (computations).

In addition, we need to define when a process is faulty in a run, since the consensus problem distinguishes between the behavior of faulty processes and that of nonfaulty ones. In section 2.2 we introduce the most basic elements of a distributed system: states, actions, and runs.[2] Then in section 2.3, we consider the notion of a system, which is simply a set of runs of the model with an associated definition of who is faulty in each of the runs.

2.2. States, actions, and runs. Throughout the paper we will assume there is a fixed finite set of $n \geq 2$ processes, which we shall denote by $1, 2, \ldots, n$, and an *environment,* denoted by e, which is used to model aspects of the system that are not modeled as being part of the activity or state of the processes. For example, we will model the communication channels or shared variables as being part of the environment's state. In addition, we will assume that various nondeterministic choices such as various delays and failures are actions performed by the environment. (What

[2]Our modeling here and in section 4.1 is based on the work of [36], which in turn extends the modeling style of [17].

are typically thought of as actions of the "scheduler" or the "adversary" we model as actions of the environment.)

For every $i \in \{e, 1, 2, \ldots, n\}$, we assume there is a set L_i consisting of all possible *local states* for i. The set of *global states,* which we will simply call *states*, will consist of

$$\mathcal{G} \;=\; L_e \times L_1 \times \cdots \times L_n.$$

We denote by x_i the local state of i in the state x.

In a given setting, every $i \in \{e, 1, 2, \ldots, n\}$ is associated with a nonempty set ACT_i of possible actions. Intuitively, these may include shared-memory operations that the process can perform, messages the process sends, and any internal bookkeeping operations or computations the process may perform. In principle, a single action can cause a number of operations to take place. However, the important point is that the decision to perform this action is taken atomically. We find it convenient to model the environment as performing actions as well. Depending on the model, the environment's actions may involve the delivery of messages, the loss of messages, determining what failures happen and when they occur, resolving race conditions, etc. We also think of the environment as being in charge of *scheduling* the processes, determining which processes are to move in each round of the computation. A *scheduling action* is a set $\mathsf{Sched} \subseteq \{1, \ldots, n\}$ of the processes that are scheduled to move next. We assume the existence of a set $\mathsf{act_e}$ describing the aspects of the environment's actions that handle everything other than the scheduling of processes. Without loss of generality we will assume that an environment's action (an element in ACT_e) is a pair (Sched, a), where Sched is a scheduling action and $a \in \mathsf{act_e}$. A *joint action* is a pair $\bar{\mathsf{a}} = (\epsilon, \mathbf{a})$, where $\epsilon = (\mathsf{Sched}, a)$ is in ACT_e, and $\mathbf{a}$ is a function with domain Sched such that $\mathbf{a}(i) \in \mathsf{ACT}_i$ for each $i \in \mathsf{Sched}$. Thus, $\bar{\mathsf{a}}$ specifies an action for the environment (via $\mathbf{a}$) and an action for every process that is scheduled to move. We define the set of joint actions by $\overline{\mathsf{ACT}}$. Clearly, $\overline{\mathsf{ACT}}$ is determined by a collection of action sets $\{\mathsf{ACT}_i\}_{i=1,\ldots,n}$ and a set $\mathsf{act_e}$. Roughly speaking, joint actions are the events that cause the global state to change into a new state. Thus, for example, a joint action in which process i sends j a message m, and the environment delivers the message m' to process i' will typically cause the local state of i' to change, as well as modifying the state of the communication channel between i and j (this state will be part of the environment's local state). This is formally captured by the notion of a *transition function*, which is a function $\tau : \mathcal{G} \times \overline{\mathsf{ACT}} \to \mathcal{G}$ from global states and joint actions to global states, describing how a joint action transforms the global state.

A *deterministic protocol* for i is a function $P_i : L_i \to \mathsf{ACT}_i$ specifying the action that i is ready to perform in every state of L_i. A *nondeterministic protocol* for i is a function $P_i : L_i \to 2^{\mathsf{ACT}_i} \setminus \emptyset$ specifying for every state of L_i a nonempty set of actions, one of which i must perform in that state. In this paper, we will focus our attention on the case in which the processes follow deterministic protocols, while the environment may follow a nondeterministic protocol.[3]

Intuitively, we think of a run as consisting of an infinite sequence of global states and the joint actions that cause the transitions among them. Notice that once we fix a

[3]The assumption that the environment's protocol may be nondeterministic is necessary for a faithful description of many models of interest. The assumption that processes follow deterministic protocols is without loss of generality in this paper, in which we are interested in worst-case lower bounds and impossibility results. It is well known and straightforward to show that any result of this type for a protocol for consensus in a given model that is proved for deterministic protocols applies to the more general class of nondeterministic protocols as well.

deterministic protocol $D = (D_1, \ldots, D_n)$ for the processes, an action $\epsilon = (\mathsf{Sched}, a)$ of the environment uniquely determines the joint action $\bar{\mathsf{a}} = (\epsilon, \mathbf{a})$ that will be performed in a given (global) state x: the set Sched determines the processes that participate in the joint action, and $\mathbf{a}(i) = D_i(x_i)$ for each $i \in \mathsf{Sched}$. Proving lower bound and impossibility results can often be thought of as showing that the adversary, which here is the environment, always has a strategy that can guarantee the desired bad behavior. With this end in mind, we will represent a joint action by specifying the environment action that determines it. Formally, we model a *run* over $\mathcal{G}$ and ACT_e as a pair $R = (r, \alpha)$, where $r : N \to \mathcal{G}$ is a function from the natural numbers to $\mathcal{G}$ defining an infinite sequence of states of $\mathcal{G}$, and $\alpha : N \to \mathsf{ACT}_e$ defines a corresponding sequence of environment actions. The intuition will be that the joint action caused by $\alpha(k)$ and the underlying protocol leads us from a state $r(k)$ to a state $r(k+1)$. As we will see later on, once we fix a model of computation and a protocol D for the processes to follow, there will be additional conditions relating the sequences r and α. These conditions guarantee, for example, that the actions recorded by α do indeed cause the transitions among the corresponding states recorded by r. The state $r(0)$ is called the *initial state* of the run R. We denote by $r(k)_i$ (resp., $r(k)_e$) the local state of process i (resp., of the environment) in $r(k)$.

An *execution* is a finite or infinite subinterval of a run, starting and ending in a state, as described next. For a run $R = (r, \alpha)$ and a pair $m \leq m'$ where m is finite and m' is finite or infinite, we denote by $R[m, m']$ the execution starting at the state $r(m)$ and ending in $r(m')$ and behaving as R does between them. Formally, $R[m, m'] = (\sigma, \beta)$ where σ has domain $[0, m' - m]$ and β has domain $[0, m' - m - 1]$, and they satisfy $\sigma(k) = r(m + k)$ and $\beta(k) = \alpha(m + k)$ for every k in their respective domains. Notice that, in principle, the same execution can occur in different runs, and for that matter even at different times. A *suffix* of a run R is an execution of the form $R[m, \infty]$ for some finite m; similarly, a *prefix* of R is an execution of the form $R[0, m]$.

Given an execution R (possibly consisting of just one state), let us denote by $R \odot \epsilon$ the execution that results from extending R by having the environment perform the action ϵ. In models in which performing a joint action at a state results in a unique next state (which will invariably be the case in this paper), every run of a deterministic protocol D can be represented in the form $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots$ where x is an initial state and ϵ_i is an environment action, for every integer $i \geq 0$.

2.3. Systems and failures. We define a *system* to be a pair $(\mathcal{R}, \mathsf{Faulty})$ where $\mathcal{R}$ is a set of runs and Faulty is a predicate on processes and runs of $\mathcal{R}$. In what follows, $\mathsf{Faulty}(i, R)$ will be taken to mean that i is faulty in the run R. We often focus on the runs of a system $\mathcal{S} = (\mathcal{R}, \mathsf{Faulty})$ and write $R \in \mathcal{S}$ as shorthand for $R \in \mathcal{R}$. Notice that Faulty determines who is faulty in a run as a function of the *whole* infinite run. Obviously, in some models of distributed computation it is possible to determine that a process is faulty by considering only a prefix of the run, sometimes even a single state. In other cases, however, the full history of the run is needed in order to determine who is faulty (this is the case, for example, in the asynchronous models of [20, 29]).

With respect to a system $\mathcal{S}$, a state y is said to *extend* the state x if there is a finite execution in some run of $\mathcal{S}$ that starts in x and ends in y. A run R is said to *contain* a state x if x is one of the global states in R. For conciseness, we will use terminology such as *a state x of $\mathcal{S}$*, when we mean a state x appearing in a run of $\mathcal{S}$, or *an initial state of $\mathcal{S}$*, when we mean a state appearing as an initial state in a run

of $\mathcal{S}$. By convention, x extends x for every state x of $\mathcal{S}$.

3. Proving impossibility using layering. Our purpose in this section is to give a simple and elegant proof of the Fischer, Lynch, and Patterson impossibility result for consensus [20], based on the notion of layering. This concrete example serves to introduce the more general ideas developed in the rest of the paper. We will present the proof for the asynchronous shared-memory model [29]. Later on in the paper we will describe the properties that play a role in the proof. The standard asynchronous shared-memory model is well known, and detailed formal descriptions can be found in textbooks such as [5, 31]. We now briefly review the features of the model that are relevant for the analysis presented in this section.

We assume the standard asynchronous shared-memory model where n processes, $n \geq 2$, communicate by reading and writing to single-writer/multi-reader, shared variables, and one process can crash. A (global) *state* x of the system is a tuple specifying a local state x_i for every process i, and the state of the environment, which in this case consists of the assignment of values to the shared variables, as well as the set of pending shared-memory operations, and the set of pending reports for read operations that have been recorded (the value has been read) but not yet reported to the reading process. The pending operations are the read and write operations that have been issued for these variables and have not yet taken effect.

The sets ACT_i and ACT_e of the actions of the processes and the environment are defined as follows. A process performs an action only when it is scheduled to move. This action is either a local operation, a read of a shared variable (belonging to it or to some other process), or a write to one of its own variables. An action of the environment can have one of three forms: (a) scheduling a process to move—resulting in the process performing an action, (b) performing a pending shared-memory operation, or (c) reporting the value read in a recorded read operation to the reading process.

A process that is scheduled to move only a finite number of times in a given run R is said to have *crashed*. We define $\mathsf{Faulty}(i, R)$ to hold, and we consider i to be *faulty in* R exactly if i crashes in R. (Notice that in this model, there is no way to determine at a finite state x that a given process is faulty in the run; a process can always "recover" in the future.) As mentioned earlier, we consider deterministic protocols without loss of generality, because any nondeterministic protocol that solves consensus in this model can, by fixing the nondeterministic choices in a fixed arbitrary way, be turned into a deterministic protocol solving it.

A run of a given deterministic protocol D in this model is a run $R = (r, \alpha)$ satisfying the following:

(i) $r(0) \in \mathsf{Con}_0$,

(ii) each process follows its protocol, and every pair of consecutive states are related according to the operations that take place as scheduled by the environment.

If in addition

(iii) every read and write action issued is eventually serviced appropriately by the environment, and

(iv) at most one process fails in R,

then we say that the run is *admissible*. Let $\mathcal{S}(D)$ be the system consisting of the set of all admissible runs of D and the predicate Faulty described above. Now, the consensus problem is well defined: D solves consensus if all runs of $\mathcal{S}(D)$ satisfy agreement, decision, and validity with failures as defined by Faulty.

3.1. Layers in the shared-memory model. We define a *layer* to consist of a finite sequence of environment actions. Our intention is to focus on runs of a protocol that are generated by a particular set of layers. If the set of layers is chosen appropriately, these runs can have structural properties that will simplify their analysis.

We define the set L^{rw} of layers in the asynchronous shared-memory model to consists of all layers of the forms

- $[p_1, \ldots, p_n]$ and
- $[p_1, \ldots, p_{n-1}]$,

where the p_i's are process names (elements of $\{1, \ldots, n\}$), and the names appearing in a given layer are pairwise distinct. We think of layers of the first type as *full* layers, since in such a layer every process moves. Layers of the second type enable the environment to "silence" a process from any point in the computation; this will play an important role in determining the "topological" properties of the layered runs (e.g., in Lemma 3.4(b)). A layer specifies a linear ordering in which the environment schedules the processes to move. In a given layer, whenever a process p_i is scheduled to move, it performs a single action, and this action (internal, a read or a write) is serviced (i.e., the read or write action is performed and, in the case of a read, the value that was read is reported to the reader) before the next process moves. Since in our model every process can be scheduled to move at any state (although in some cases the pending operation may just be a "skip" internal operation), the sequence of environment actions described in a layer of L^{rw} can be applied at every state. The intuition behind the definition of L^{rw} is that a layer consists of a "round" in which at least $n-1$ processes get to move (sequentially). Notice that in an infinite sequence of layers, at least $n-1$ process names appear infinitely often. The layers just defined are designed to ensure that each layer contributes towards the fairness conditions (iii) and (iv). As a result, every run generated by an infinite sequence of such layers is admissible. Notice that a layer of the first type is specified by a permutation on $\{1, \ldots, n\}$. We will find it useful to identify a layer of the first type with the corresponding permutation. Because of this connection we call this layering a *permutation layering*.

Once we have fixed the protocol D, we can think of a layer $L \in \mathsf{L}^{rw}$ as an individual "higher-level" action by the environment. An individual environment step in a run would then consist of performing a whole layer. We define *an* L^{rw}*-run of* D to be a run of the form

$$x^0 \odot L_1 \odot L_2 \odot \cdots,$$

where $x^0 \in \mathsf{Con}_0$ and $L_i \in \mathsf{L}^{rw}$ for every $i \geq 1$. More formally, let us denote by $x \cdot L$ the state that is reached at the end of an execution that starts in x, where the processes follow D and the environment performs actions according to L. The L^{rw}-run depicted above is a run $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha)$ with $\alpha = (L_1, L_2, \ldots)$ and $r_{\mathsf{L}} = (x^0, x^1, \ldots)$ where for $i \geq 0$ we inductively define $x^{i+i} = x^i \cdot L_{i+1}$. Thus, in an L^{rw}-run, we view the environment as performing actions consisting of whole layers, and we ignore the intermediate states that arise "in the middle" of a layer.

Notice that every L^{rw}-run R_{L} of D corresponds to a unique run $R \in \mathcal{S}(D)$ that is obtained from R_{L} by "expanding" the layers into the detailed sequence of environment actions they describe, and adding the intermediate states. We call a process *faulty* in R_{L} if it is faulty in the corresponding run R. We denote by $\mathcal{S}_{\mathsf{L}}(D)$ the system consisting of the L^{rw}-runs of D, with this definition of failures. The argument above that executing an infinite number of L^{rw} layers yields an admissible run can now be formalized as follows.

LEMMA 3.1. *For every run $R_{\mathsf{L}} \in \mathcal{S}_{\mathsf{L}}(D)$, the run $R \in \mathcal{S}(D)$ corresponding to R_{L} is admissible.*

Proof. By definition of R_{L}, the corresponding run R is clearly a run of D in the model. It is admissible because (a) by definition of the layers, each read and write action that is performed by a process is serviced immediately, and (b) at most one process can fail in R because in each of the infinite sequence of layers that generate R at least $n-1$ of the n processes is scheduled to perform an action; hence, at most one process can move only a finite number of times in R. □

3.2. Potence connectivity. Recall that decisions made by the processes appear in their local states from the point of decision on. Therefore, any infinite subsequence of the states of a run will contain the information about the decisions that the different processes perform in the run. Thus, we do not lose information about the decision values by considering the states of an L^{rw}-run instead of looking at the full detail of the run corresponding to it. If a protocol D solves consensus in the asynchronous model just described, it must do so in *every* admissible run. It follows that in every L^{rw}-run of D the nonfaulty processes must decide on a value $v \in \{0,1\}$. A crucial role in the proof of impossibility of consensus in the asynchronous shared-memory model is played by the decision values that are possible in the future of a given global state. This is captured by the notion of the *potential valence* (or *potence* for short)[4] of a state with respect to a set of runs.

DEFINITION 3.2. *A state x is* w-potent *with respect to a system $\mathcal{S}$ if x is a state of a run $R \in \mathcal{S}$ in which at least one nonfaulty process decides w. The state x is* bipotent *if it is both* 0*-potent and* 1*-potent.*

When discussing potence, we sometimes omit the system $\mathcal{S}$ when it is clear from context. Notice that if a state x is w-potent (resp., bipotent) with respect to $\mathcal{S}_{\mathsf{L}}(D)$, then it is guaranteed to be w-potent (resp., bipotent) with respect to $\mathcal{S}(D)$: If R_{L} is the witness proving that x is w-potent in $\mathcal{S}_{\mathsf{L}}(D)$, then x appears in the run $R \in \mathcal{S}(D)$ corresponding to R_{L}, and R is a witness to w-potence of x with respect to $\mathcal{S}(D)$. Clearly, the converse need not hold. Bipotent states play an important role in delaying (or precluding) consensus, as we shall see in the next section. Our goal will be to show that every consensus protocol must have a run where agreement is not reached. We will do this in the next section by demonstrating that every protocol for consensus must have a run whose states are all bipotent. Such a run we call a *bipotent run.* The notion of *potence connectivity* is a powerful tool for proving this and other impossibility results.

DEFINITION 3.3. *With respect to a system $\mathcal{S}$,*

(i) *two states x and y have* shared potence*, denoted by $x \sim_p y$, if both are w-potent for some $w \in \{0,1\}$;*

(ii) *a set of states X is* potence connected *if the graph $(X, \sim_p)$ induced by $\sim_p$ on X is connected.*

Potence connectedness is not a very strong condition. Indeed, it is easy to check that a set X of states is potence connected exactly if either (i) for some value w, all states of X are w-potent, or (ii) there exists at least one bipotent state in X. Equivalently, X is not potence connected exactly if X contains both 0-potent and 1-potent states but does not contain a bipotent state.

[4] In [32] we used the term *valence* instead of potence. This is changed here because the term valence is used slightly differently in the literature, starting with [20]. Briefly, a state is said to be w-valent if *all* extensions decide w. (However, bivalent is equivalent to bipotent.) We thank Gerard Tel for suggesting the term.

The following criterion will serve us to prove that two states have shared potence.

LEMMA 3.4. *Assume the protocol D satisfies the decision property, and let x and x' be states of $\mathcal{S}_{\mathsf{L}}(D)$. If there are states $y, y' \in \mathcal{S}_{\mathsf{L}}(D)$, where y extends x and y' extends x', such that y and y' differ at most in the local state of a single process, then $x \sim_p x'$ with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. Let R_x be an execution in $\mathcal{S}_{\mathsf{L}}(D)$ that ends in state x, and let $R_{x'}$ be one that ends in x'. Since y extends x, there is a finite sequence of layers σ such that $R_x \odot \sigma$ ends in y, and a corresponding sequence σ' such that $R_{x'} \odot \sigma'$ ends in y'. By assumption, we have that y and y' differ at most in the local state of process p. Choose a layer $L \in \mathsf{L}^{rw}$ in which p is not scheduled to move. We extend each state by applying L repeatedly. There are two runs $R, R' \in \mathcal{S}_{\mathsf{L}}(D)$ such that $R = R_x \odot \sigma \odot L^\infty$ and $R' = R_{x'} \odot \sigma' \odot L^\infty$. Both are clearly runs of $\mathcal{S}_{\mathsf{L}}(D)$, and in both runs process p is faulty and the rest are not faulty. A straightforward induction on the number of actions performed by the environment shows that all processes other than p have the same local state history in the suffix of R starting at y as they do in corresponding points in the suffix of R' starting at y'. It follows that the nonfaulty processes reach the same decision in both. Since D satisfies the decision property, decisions are reached by the nonfaulty processes in these runs, and we have that $x \sim_p x'$. □

Lemma 3.4 directly captures as particular instances many useful cases. Thus, for example, it implies that if x extends y, then they have a shared potence. Similarly, if a state z extends both x and y, then $x \sim_p y$. A third useful instance is that if x and y differ only in the local state of one process, then $x \sim_p y$.

The crux of the impossibility proof is captured in the following lemma, which shows that the set of successors of a state in $\mathcal{S}_{\mathsf{L}}(D)$ is guaranteed to be potence connected.

LEMMA 3.5. *Assume D satisfies the decision property. For every state x of $\mathcal{S}_{\mathsf{L}}(D)$, the set of states $\mathsf{L}^{rw}(x) = \{y \mid y = x \cdot L,$ for some $L \in \mathsf{L}^{rw}\}$ is potence connected with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. We start by showing two cases in which states of $\mathsf{L}^{rw}(x)$ have a shared potence, from which the result will follow. For every permutation $p_1, p_2, \ldots, p_n$ of $\{1, \ldots, n\}$ we claim that the following holds:

(i) $x \cdot [p_1, \ldots, p_{n-1}] \ \sim_p \ x \cdot [p_1, \ldots, p_n]$.

This is true by Lemma 3.4 because there is a state of $\mathcal{S}_{\mathsf{L}}(D)$ that extends both states. This state results from extending the first state by applying the layer $L = [p_n, p_1, \ldots, p_{n-1}]$, and it also results from extending the second state by $L' = [p_1, \ldots, p_{n-1}]$. Clearly,

$$x \cdot [p_1, \ldots, p_{n-1}] \cdot [p_n, p_1, \ldots, p_{n-1}] \quad = \quad x \cdot [p_1, \ldots, p_n] \cdot [p_1, \ldots, p_{n-1}],$$

since in both cases exactly the same actions take place in the same order in the two layers following x.

For a permutation $\pi = [p_1, \ldots, p_n]$, we denote the permutation $[p_1, \ldots, p_{k-1}, p_{k+1}, p_k, p_{k+2}, \ldots, p_n]$ that is obtained by transposing the kth and $(k+1)$st elements of π by $\mathsf{Tr}(k, \pi)$. We claim the following.

(ii) Let π be a full layer and $k \in \{1, \ldots, n-1\}$. Then $x \cdot \pi \quad \sim_p \quad x \cdot \mathsf{Tr}(k, \pi)$.

Let $y = x \cdot \pi$ and $y' = x \cdot \mathsf{Tr}(k, \pi)$. In moving from x to both y and y', every individual process performs the same action. Recall that in our model each shared variable can be written by a single process. Thus, every process that executes a write writes the same value in both cases, and the state of the shared memory is the same in y and in y'. In addition, every process except possibly for p_k and p_{k+1} end

up in the same local state in y and in y'. Now, if both processes perform a read or both perform a write, each of them will end up in the same local state in y and in y'. Finally, if one of them executes a read and the other a write, only the reading process may end up in a different state in y and in y'. It follows that y and y' differ at most in the local state of one process, so $y \sim_p y'$ by Lemma 3.4.

The potence connectivity of $\mathsf{L}^{rw}(x)$ follows: A well-known property of the group of permutations is that we can transform any permutation π of $\{1, \ldots, n\}$ to any other permutation π' using a finite number of single transpositions $\mathsf{Tr}(k, \pi)$ as in (ii). Thus, by transitivity of connectivity we have that all states $x \cdot \pi$ obtained from x by a full layer are potence connected. Each of the remaining states of $\mathsf{L}^{rw}(x)$ is obtained by a layer $[p_1, \ldots, p_{n-1}]$ and is connected to the rest by part (i). □

The last connectivity property is implicit in [20].

LEMMA 3.6. *If D satisfies the decision property, then Con_0 is potence connected with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. In this proof, given a state z we denote by z_j the local state of process j in the state z. Let $x, y \in \mathsf{Con}_0$, and for every $0 \leq m \leq n$ define x^m by setting

$$x_j^m = \begin{cases} y_j & \forall j \leq m, \text{ and} \\ x_j & \forall j > m. \end{cases}$$

Clearly, $x^m \in \mathsf{Con}_0$, and it is easy to check that $x^0 = x$ and $x^n = y$. Moreover, for every $0 < l \leq n$ we have that x^{m-1} and x^m differ exactly in the local state of process m. Con_0 is a subset of the states of $\mathcal{S}_{\mathsf{L}}(D)$. Hence, by Lemma 3.4 we have that $x^{m-1} \sim_p x^m$. It follows that x and y are potence connected and we are done. □

3.3. Using bipotence to prove impossibility. In the previous section we used the decision requirement of consensus, together with the possibility of having one process crash, to establish connectivity properties. In this section we show how these properties, together with the agreement and validity requirements of consensus, imply the impossibility result. We start by demonstrating that bipotent states can play a role in delaying, or precluding, consensus.

LEMMA 3.7. *Assume that the protocol D satisfies the agreement and decision properties. If a state x of $\mathcal{S}_{\mathsf{L}}(D)$ is bipotent with respect to $\mathcal{S}_{\mathsf{L}}(D)$, then no process has decided in x.*

Proof. Assume by way of contradiction that i has decided on value w in x. Since x is bipotent, there is a run $R \in \mathcal{S}_{\mathsf{L}}(D)$ containing x in which some process, say j, decides $1 - w$. Let y be a state of this run extending x in which j has already decided $1 - w$. Thus, at y processes i and j have (irrevocably) decided on different values. Let P be a prefix of this run that ends in y, and let $L \in \mathsf{L}^{rw}$ be any full layer. The run $R' = P \odot L^\infty$ is a run of $\mathcal{S}_{\mathsf{L}}(D)$ in which i decides w, process j decides $1 - w$, and both are nonfaulty (they move infinitely often). Processes i and j similarly decide w and $1 - w$, respectively, in the run $\hat{R} \in \mathcal{S}(D)$ corresponding to R', contradicting the assumption that D satisfies the agreement property. □

The following lemma is the basis for the impossibility proof.

LEMMA 3.8. *Let X be a potence connected (with respect to $\mathcal{S}_{\mathsf{L}}(D)$) set of states of $\mathcal{S}_{\mathsf{L}}(D)$. If X contains both 0-potent and 1-potent states, then there is a bipotent state in X.*

Proof. Let X^0 be the subset of X consisting of 0-potent states, while X^1 is the subset of 1-potent states. By assumption, both subsets are nonempty. It thus follows that there must be an edge $x^0 \sim_p x^1$ with $x^0 \in X^0$ and $x^1 \in X^1$ (otherwise X is not potence connected).

Since $x^0 \sim_p x^1$, the states x^0 and x^1 have a shared potence. If their shared potence is 0, then x^1 is bipotent, while if the shared potence is 1, then x^0 is bipotent. In either case there is a bipotent state in X, as desired. □

We can use Lemma 3.6 to obtain a bipotent initial state x^0, a well-known result of [20].

LEMMA 3.9. *If D satisfies the decision and validity properties, then the set Con_0 contains a bipotent state with respect to $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. Let $x^0 \in \mathsf{Con}_0$ be the initial state in which all initial values are 0, and let x^1 be the corresponding state with all values 1. By the validity condition in every admissible run of D starting in the state x^0 (resp., x^1) the nonfaulty processes decide 0 (resp., 1). Since L^{rw}-runs correspond to admissible runs, and there is an L^{rw}-run starting in x^0 and an L^{rw}-run starting in x^1, x^0 is 0-potent and x^1 is 1-potent. By Lemma 3.6 Con_0 is potence connected, and thus by Lemma 3.8 we obtain that there is a bipotent state in Con_0. □

LEMMA 3.10. *Every protocol D that satisfies decision and validity has a bipotent run in $\mathcal{S}_{\mathsf{L}}(D)$.*

Proof. By Lemma 3.9, there is a bipotent state, say x^0, in Con_0. We will construct a sequence of bipotent states $x^1, x^2, \ldots, x^k, \ldots$ and a corresponding sequence of layers $L_1, L_2, \ldots, L_k, \ldots$ such that $x^{i+1} = x^i \cdot L_{i+1}$ for all $i \geq 0$. The desired run will be

$$R_{\mathsf{L}} = x^0 \odot L_1 \odot L_2 \odot \cdots \odot L_k \odot \cdots.$$

It remains to define the two sequences. We will define the layers L_i and the states x^i by induction on i. Notice that x^0 is bipotent. Let $k \geq 0$, and assume that we have constructed sequences $L_1, L_2, \ldots, L_k$ and $x^1, x^2, \ldots, x^k$ with the desired properties. In particular, x^k is a bipotent state. Since D satisfies decision, we have by Lemma 3.5 that $\mathsf{L}^{rw}(x^k)$ is potence connected. Since x^k is bipotent, $\mathsf{L}^{rw}(x^k)$ contains both 0-potent and 1-potent states. It follows from Lemma 3.8 that there is a bipotent state $y \in \mathsf{L}^{rw}(x^k)$. Since $y \in \mathsf{L}^{rw}(x^k)$ we have by the definition of $\mathsf{L}^{rw}(x^k)$ that $y = x^k \cdot L$ for some layer L. Set $L_{k+1} = L$ and $x^{k+1} = y$. □

We conclude the following.

THEOREM 3.11. *There is no 1-resilient consensus protocol in the asynchronous shared-memory model.*

Proof. We need to show that no protocol D can satisfy all three properties of consensus: decision, validity, and agreement. Assume that D satisfies decision and validity. By Lemma 3.10 there is a bipotent run $R \in \mathcal{S}_{\mathsf{L}}(D)$. This run has an infinite number of bipotent states. If D were to satisfy agreement as well, we have by Lemma 3.7 that no process could have decided in a bipotent state. But then no process would ever decide in the run R, and hence no process would ever decide in the corresponding run of $\mathcal{S}(D)$, contradicting the assumption that D satisfies the decision property. □

3.4. Discussion. In this section we have used layering to provide an alternative proof of the impossibility of consensus in the asynchronous shared-memory model. Our proof differs from those of [20, 29] in a number of ways. First, it does not depend on a "critical state" argument; rather, it constructs a bipotent run inductively one state (or rather one layer) at a time. A subtle aspect of the standard proofs is proving that prefixes of a run can be extended into full runs in a manner that satisfies the fairness (or admissibility) conditions. In our case this is simplified by having each layer contribute "enough" to the fairness conditions to guarantee that any run constructed as a sequence of layers is admissible. We have attempted to provide a proof that

makes fairly limited and local use of the particular properties of the model. As a result, as we shall see in the following, the same proof outline is applicable to the analysis of consensus in other models. Thus, we believe that the notions of potence connectivity and layering capture some of the topological structure underlying the consensus problem (and more generally 1-resilient solvability, as described in [32]). For example, the layering approach can be used to prove global connectivity properties of a model, as opposed to the approach described in this section, which shows connectivity of successors of a state; more about this is in section 9.

We next provide a more general framework and show how the notions of potence and layering and the proof outline just given can be applied more broadly. We start by describing how to model different types of distributed systems in a general fashion.

4. Layering consensus in general models. We now consider how layering can be used to analyze consensus for a variety of models. We start by defining models of distributed systems in a more general manner, and we show how layering can be applied to consensus in such generic models.

4.1. Models of distributed computation. Using the notions of section 2.2, we define a generic model of computation. A *model of distributed computation* is determined by sets L_i, $i \in \{e, 1, 2, \ldots, n\}$, of local states for the processes and the environment, and corresponding sets of actions ACT_i, for every $i \in \{e, 1, 2, \ldots, n\}$, and by a tuple $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$, where the following hold:

- $\mathcal{G}_0 \subseteq \mathcal{G}$ is called the set of *initial states*. The identity of $\mathcal{G}_0$ will depend on the type of analysis for which the model is introduced. When we focus on a particular problem such as consensus, $\mathcal{G}_0$ is the set Con_0 of initial states for consensus.
- P_e is a (nondeterministic) protocol for the environment.
- τ is a transition function.
- Ψ is a set of runs over $\mathcal{G}$ and ACT_e, such that for every pair of runs R and R' that have a suffix in common, $R \in \Psi$ if and only if $R' \in \Psi$. The set Ψ is called the set of *admissible* runs in the model. This is a tool for specifying fairness properties of the model. For example, properties such as "every message sent is eventually delivered" or "every process moves infinitely often" are enforced by allowing as admissible only runs in which these properties hold. The condition we have on Ψ being determined by the suffixes of its runs ensures that admissibility depends only on the infinitary behavior of the run.
- FGen is a function that, for each protocol D gives a predicate Faulty_D defined on the runs of D in M (defined below). We remark that the dependence of the Faulty_D on the protocol is useful when we want to capture the idea that a process is faulty if it deviates from the protocol it is supposed to follow. This is relevant for handling malicious failures, for example.

We say that a run $R = (r, \alpha)$ is a *run of the protocol* $D = (D_1, \ldots, D_n)$ *in* $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$ when

(i) $r(0) \in \mathcal{G}_0$, so that R begins in a legal initial state according to M,
(ii) $\alpha(k) \in P_e(r(k)_e)$ for all k,
(iii) $r(k+1) = \tau(r(k), (\alpha(k), \mathbf{a}^k))$ holds for all $k \geq 0$, where the domain of $\mathbf{a}^k$ is the set Sched in $\alpha(k)$, and $\mathbf{a}^k_i = D_i(r(k)_i)$ for every $i \in \mathsf{Sched}$,[5] and

[5]Given this choice, any deviations of a process from the protocol, as may happen in a model with malicious failures, will need to be modeled as resulting from the environment's actions. The behavior of faulty processes in such a case will be controlled by the environment.

(iv) $R \in \Psi$, so that R is admissible.

Condition (ii) implies that the environment's action at every state of R is legal according to its protocol P_e, and condition (iii) states that the state transitions in R are according to the transition function τ, assuming that the joint action is the one determined by the environment's action and the actions that the protocol D specifies for the processes that are scheduled to move. A run satisfying properties (i)–(iii) but not necessarily the admissibility condition (iv) is called *a run of D consistent with M*. It is a run in which the initial state and local transitions are according to D and M, but the admissibility conditions imposed by Ψ are not necessarily satisfied. We will find it useful to consider such runs in section 6.

The notions of models and protocols give us a way of focusing on a special class of systems, resulting from the execution of a given protocol in a particular model. We denote by $\mathcal{S}(D, M, I)$ the system $(\mathcal{R}, \mathsf{Faulty})$, where $\mathcal{R}$ is the set of all runs of protocol D in the model M that start in initial states from a set I, where $I \subseteq \mathcal{G}_0$, and $\mathsf{Faulty} = \mathsf{FGen}^M(D)$.

We say that a system $\mathcal{S}$ satisfies the *pasting property* if, for every pair $R = (r, \alpha)$ and $R' = (r', \alpha')$ of runs of $\mathcal{S}$ such that $r(m) = r'(m')$ for some integers m, m', there is a run R'' of $\mathcal{S}$ such that $R''[0, m] = R[0, m]$ and $R''[m, \infty] = R'[m', \infty]$. Intuitively, the pasting property says that we can "paste" any prefix ending in a state x with a suffix starting in x, and obtain a run of $\mathcal{S}$. In a sense, this means that all of the information that is relevant to determining the future of a state is included in the state.

It is straightforward to show the following.

LEMMA 4.1. *Every system of the form $\mathcal{S} = \mathcal{S}(D, M, I)$ has the pasting property.*

Proof. Assume that $R = (r, \alpha)$ and $R' = (r', \alpha)$ are runs of $\mathcal{S}$, and that $r(m) = r'(m')$, and let R'' be defined as in the definition of the pasting property. We need to show that R'' satisfies conditions (i)–(iv) above. For condition (i), $r''(0) = r(0)$ and $r(0) \in \mathcal{G}_0$ since R satisfies condition (i). For $k < m$, we have that $r''(k) = r(k)$, $\alpha''(k) = \alpha(k)$, and $r''(k+1) = r(k+1)$. Therefore properties (ii) and (iii) follow for $k < m$ from the fact that they hold for R. Similarly, for $k \geq m$, $r''(k) = r'(m'+k-m)$, $\alpha''(k) = \alpha'(m'+k-m)$, and $r''(k+1) = r'(m'+k+1-m)$ so that (ii) and (iii) for these values of k follow from the fact that they hold for the run R' (at time $m'+k-m$). Finally, $R''[m, \infty] = R'[m', \infty]$, so that R'' and R' have a suffix in common. Since $R' \in \Psi$, it thus follows that $R'' \in \Psi$ so R'' satisfies (iv), and we are done. □

Finally, recall that the definition of consensus depends in an essential way on the behavior of the nonfaulty processes. Since we have a very general notion of a Faulty predicate that may depend on the model, we will restrict attention to cases in which the notion of failures is not completely out of hand. A predicate Faulty defined for the runs of a system $\mathcal{S}$ induces a notion of a process being *failed at a state* (with respect to $\mathcal{S}$). We say that a process i is *failed at state x* if i is faulty in all runs of $\mathcal{S}$ containing x. Otherwise i is *nonfailed* at x.

DEFINITION 4.2. *A system $\mathcal{S} = (\mathcal{R}, \mathsf{Faulty})$ satisfies* fault independence *if the following hold:*

(i) *For every state x of $\mathcal{S}$ there is a run $R^x \in \mathcal{S}$ in which x appears, such that the only processes that fail in R^x are those that are already failed at x.*
(ii) *No process is failed at an initial state of $\mathcal{S}$.*
(iii) *If R and R' have a common suffix, then the same processes fail in both runs.*
(iv) *At most $n-1$ processes fail in any run $R \in \mathcal{S}$.*

Part (i) is formally captured by the condition

$$(\forall x \text{ of } \mathcal{S})(\exists R^x \in \mathcal{S})\forall i\,[\mathsf{Faulty}(i, R^x) \text{ iff } i \text{ is failed at } x].$$

The intuition here is that every instance of faulty behavior should be the result of the failure of some component in the system. If there is an extension of a state in which one component fails and another does not, and there is a different execution where the second component fails but the first one does not, then there should be a third execution where neither one fails. Failures are thus independent in this sense. Part (ii) implies that for any initial state x and every process i, there is a run R^x containing x where i is nonfaulty. (Of course there may be another such run where i is faulty.) Part (iii) is included because the failures we are interested in (e.g., crashes) are determined by the infinite part (i.e., suffix) of a run. Notice that there are failure models in which a failure can be committed at a given point in time and not be reflected in the processes' states at a later time. This could cause two runs to have the same suffix while a particular process behaves in a faulty manner in one of the runs but not in the other. However, by appropriately modeling the environment's state to keep track of such failures, part (iii) can be guaranteed in such models as well. Part (iv) allows us to concentrate on runs where at least one process did not fail.

All systems we will consider are assumed to satisfy fault independence.

4.2. Example: The mobile failures model. We illustrate the use of the abstract framework just described by describing a synchronous model with a *single mobile failure* [35]. The model is the standard synchronous message-passing model in which communication proceeds in lockstep rounds; in every round each process can send a message to each of the other processes, and messages are delivered in the round they are sent with no corruptions. The failure assumption is that in every round m there can be at most one process i_m some of whose messages are lost. The set of messages lost by this process in the round in question is arbitrary. We use the term *mobile failure* because the identity of the process whose messages may be lost can change from one round to the next.

We now sketch how this model, which we denote by $M^{\mathsf{m}} = (\mathcal{G}_0^{\mathsf{m}}, P_e^{\mathsf{m}}, \tau^{\mathsf{m}}, \Psi^{\mathsf{m}}, \mathsf{FGen}^{\mathsf{m}})$, can be represented in terms of our formalism. The environment's state is assumed to be the same in all states of $\mathcal{G}_0^{\mathsf{m}}$. The environment's protocol P_e^{m} is uniform at all states: nondeterministically choose a process $i \in \{1, \ldots, n\}$ and a set $T \subseteq \{1, \ldots, n\}$ and perform the action $(\{1, \ldots, n\}, (i, T))$. Notice that in all cases $\mathsf{Sched} = \{1, \ldots, n\}$ so that, intuitively, all processes are scheduled to move in every round. The action $a = (i, T)$ specifies that any message sent from i to members of T will be lost. The actions a process can perform in this model specify a list of at most one message to be sent to each process in the current round. Thus, given a protocol $D = (D_1, \ldots, D_n)$ for the processes, for every state x the protocol D_i for i defines an action $D_i(x_i)$ that specifies what is its next state and the messages that i sends in that state.

Given a state x and a joint action $\bar{\mathsf{a}} = (a_e, \mathbf{a})$, $a_e = (\{1, \ldots, n\}, (j_0, T_0))$, the transition function τ^{m} updates the local state of a process i as a function of its current action, $D_i(x_i)$, and the list of messages that are sent to it in the current round that are not blocked (i.e., the list of messages sent to i by processes j according to $D_j(x_j)$ except that if $i \in T_0$, we ignore any message that may be sent by $j = j_0$).

The set Ψ^{m} makes no restrictions whatsoever: it consists of all possible runs consistent with M^{m}. Finally, $\mathsf{FGen}^{\mathsf{m}}(D)(i, R) = \mathit{false}$ for all processes i, runs R, and

protocols D; in this model, no process is ever considered faulty.[6]

Note that M^{m} satisfies fault independence. Namely, consider any system $\mathcal{S} = \mathcal{S}(D, M^{\mathsf{m}}, I)$. All properties of Definition 4.2 hold trivially for $\mathcal{S}$, because no processes are ever considered faulty in M^{m}.

5. Abstract impossibility framework for consensus. We are now ready to initiate a general model-independent analysis of the consensus problem. We will attempt to show that the general structure of the proof we presented in section 3 is widely applicable. As in section 2.1, we will assume a uniform set Con_0 of initial states for consensus. The local state of every process i in a state x of Con_0 consists of two distinct variables, $\mathtt{v}_i$ and $\mathtt{d}_i$. The first has a binary value and is considered i's *initial value* for the purpose of the consensus procedure. The second is a write-once variable that appears in all of i's local states and is initially undefined (i.e., initially $\mathtt{d}_i = \perp$). The environment's state is assumed to be the same in all states of Con_0.

In what follows, we shall focus on systems that are compatible with the consensus problem. We call a system $\mathcal{S} = \mathcal{S}(D, M, I)$ a *system for consensus* if (i) the set I of initial states in $\mathcal{S}$ is Con_0 and (ii) the local state x_i of every process i in each state x of $\mathcal{S}$ contains the variable $\mathtt{d}_i$, and in all runs the variable $\mathtt{d}_i$ is write-once. A model $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$ is a *model for consensus* if $\mathsf{Con}_0 \subseteq \mathcal{G}_0$ and every system of the form $S(D, M, \mathsf{Con}_0)$ is a system for consensus.

All models are models for consensus satisfying fault independence.

The consensus problem in section 2.1 is now well defined with respect to a general model M: a protocol D solves consensus if all the runs of the system $\mathcal{S}(D, M, \mathsf{Con}_0)$ satisfy agreement, decision, and validity, where faulty processes are defined according to the function $\mathsf{Faulty}_D = \mathsf{FGen}(D)$.

5.1. Potence and bipotence in general models. Recall the definitions of w-potence and bipotence with respect to a system $\mathcal{S}$ of section 3.2. In particular, they hold for $\mathcal{S} = \mathcal{S}(D, M, I)$. In this section we consider two useful ways to show that $x \sim_p y$ in general models. We start with some specific conditions under which analogues to Lemma 3.4 hold.

LEMMA 5.1. *Assume that $\mathcal{S}$ satisfies decision and let x, y, z be states of $\mathcal{S}$.*

(i) *For $v \in \{0, 1\}$, if z is v-potent and z extends y, then y is also v-potent.*

(ii) *If z extends both x and y, then $x \sim_p y$.*

Proof. For part (i), assume z is v-potent. Then there exists a run $R = (r, \alpha)$ of $\mathcal{S}$ and a time $m \geq 0$ such that $r(m) = z$ and some nonfaulty process i in R decides v. Since z extends y, there is a run $R' = (r', \alpha')$ and there are times $k' \geq k \geq 0$ such that $r'(k) = y$ and $r'(k') = z$, in which the only processes that fail are those that are already failed at z (fault independence (i) and (iii)). Consider the run R'' obtained by pasting the prefix $R'[0, k']$ with the suffix $R[m, \infty]$. Since the system $\mathcal{S}$ satisfies the pasting property (by Lemma 4.1), R'' is a run of $\mathcal{S}$. Process i reaches the same decisions in R and in R'' (decisions are write-once and the $\mathtt{d}_i$ variable appears in all of i's local states). Since R and R'' have a common suffix, the same processes fail in both runs (fault independence (iii)). It follows that i is a nonfaulty process deciding v in R''. The state y is therefore v-potent and we are done.

[6] This assumption is made for simplicity. Notice that the process i_m, some of whose messages may be lost in a given round m, still receives all messages sent to it. This process should therefore not be at a disadvantage when it comes to being able to decide on the consensus value. The same analysis and conclusions as we present here would hold if, for example, we would assume that a process that is silenced from some point on in a given run is considered faulty in that run.

We now show part (ii). Since $\mathcal{S}$ satisfies the decision property, every state of $\mathcal{S}$, including z in particular, is either 0-potent or 1-potent (or both). Assume that z is v-potent. By part (i) y is v-potent, and by the same argument x is v-potent as well. Hence $x \sim_p y$ and we are done. □

Recall that a run is bipotent if all of its states are bipotent. An important consequence of the agreement property is that a consensus protocol cannot terminate while in a bipotent state. As a result, if a protocol has a bipotent run, then it cannot solve consensus. This is an important feature underlying impossibility proofs for consensus. We now capture these claims as they apply to general models more formally. One feature of many popular models, including the common asynchronous ones, is that the failure of a process is not determined in finite time. We say that a system $\mathcal{S}$ displays *no finite failure* if, for all states x of $\mathcal{S}$, no process is failed at x. Namely, for every process i, there is a run containing x in which i is nonfaulty. For such systems we have the following lemma, which is a slight generalization of Lemma 3.7 and whose proof has the same structure.

LEMMA 5.2. *Let $\mathcal{S}$ satisfy the agreement requirement and assume there is a bipotent run $R^b \in \mathcal{S}$. If $\mathcal{S}$ displays no finite failure then $\mathcal{S}$ does not satisfy the decision property.*

Proof. We will show that if a state x of $\mathcal{S}$ is bipotent, then no process has decided by x. The claim follows, since in a bipotent run no process will ever decide, and we have by the fault independence assumption part (iv) that there must be at least one nonfaulty process in R^b.

Assume by way of contradiction that x is bipotent and i is decided at x. Let its decision at x be $\mathtt{d}_i = w \neq \perp$. Since x is bipotent, there is a run R containing x in which some process, say j, decides $1 - w$. It follows that there is in R a state y extending x in which $\mathtt{d}_j = 1 - w$. Since $\mathcal{S}$ displays no finite failure, both i and j are nonfailed at y. By the fault independence assumption part (i), there is a run R' containing y in which both i and j are nonfaulty. Since $\mathtt{d}_i$ and $\mathtt{d}_j$ are write-once, however, in R' process i decides w while j decides $1-w$, contradicting the assumption that $\mathcal{S}$ satisfies the agreement property. □

Lemma 5.2 clearly demonstrates that consensus cannot be attained at a bipotent state of a system that displays no finite failure. The following lemma shows that a consensus protocol is still unable to terminate in a bipotent state even in systems in which failures can be observed in finite time.

LEMMA 5.3. *Let $\mathcal{S}$ be a system satisfying the agreement requirement and assume that no more than $t < n$ processes fail in any run of $\mathcal{S}$. If x is a bipotent state of $\mathcal{S}$ then at least $n - t$ nonfailed processes at x have not decided by x.*

Proof. Since x is bipotent, there is a run R^0 containing x in which at least one nonfaulty process decides 0. The set P_0 of nonfaulty processes in R^0 consists of at least $n-t$ processes. They are all nonfailed at x, and by the agreement property none of them has decided 1 by x. By symmetry of 0 and 1, we obtain the existence of a set P_1 of at least $n - t$ nonfailed processes at x that have not decided 0 by x. By the fault independence property part (i), there is a run $\hat{R}$ containing x in which the only processes that fail are the ones that are already failed at x. All processes in $P_0 \cup P_1$ are nonfaulty in $\hat{R}$. By the agreement property, there is at most one value $w \in \{0, 1\}$ on which nonfaulty processes decide in $\hat{R}$. If nonfaulty processes do not decide 0 in $\hat{R}$, then no process in P_0 has decided by x, and if they do not decide 1 in $\hat{R}$, then no process in P_1 has decided by x. In either case, the claim holds. □

5.2. Potence connectivity revisited. The central role played by connectivity in the analysis of consensus and decision problems in general has been observed by many authors starting with [18]. The traditional notion of connectivity in the literature [18, 31, 35] is based on comparing the local states of processes in the *current* (global) state: Two states are similar if they share enough structure (e.g., equal environment and process states), and the transitive closure of this binary relation provides a corresponding notion of (similarity-based) connectivity. In contrast, the shared potence of states depends on their possible future extensions, and hence so is potence connectivity. Clearly, a notion of the first kind is independent of the protocol used, while potence connectivity is protocol dependent. It is our view that potence connectivity plays a crucial role in the structure of consensus. In addition to generalizing our treatment of potence connectivity slightly, in this section we will draw a formal connection between similarity connectivity and potence connectivity. Intuitively, we will show that in models in which the environment can always silence an arbitrary process, similarity connectivity yields potence connectivity. Similarity-based connectivity will thus prove to be a useful tool for showing potence connectivity.

Similarity connectivity + crashlike behavior $\Rightarrow$ potence connectivity. Many failure models considered in the study of fault tolerance allow faulty behavior in which the state of a process is "hidden" from some point on. Usually this happens as a result of a process crash, but it can also be the result of the process's memory being erased, for example. When such hiding can occur, states that differ only in the local state of one process will often have a shared potence. It follows that there is a connection between similarity of states and potence connectivity. We now formalize this connection.

The state of the environment is often best described as a tuple of distinct components, each accounting for a separate aspect of the system. In some models, part of the components of the state of the environment can affect only the state of a single process. For example, in a shared-memory model, if there is a variable that can be read *only* by process j, then it can affect only j. A similar situation occurs in the message-passing model, when a channel contains messages that were all sent to j. We call such components *j-components*. Clearly, in some models the environment state has no j-components. When they do exist, the following notions depend on a clear definition of these components. Two environment states are said to *agree modulo j* if they are the same except, possibly, for their j-components. We say that two states x and y *agree modulo j* if $x_i = y_i$ for all $i \neq j$ and their environment states x_e and y_e agree modulo j. The intuition is that if process j can somehow be "silenced" or if its local state can be "hidden" and not observed by others in the future, runs resulting from both states will be the same from the point of view of the other processes. In models in which there are no j-components to the environment's state, states x and y that agree modulo j differ at most in the identity of process j's local state.

DEFINITION 5.4 (similarity). *With respect to a system $\mathcal{S}$,*

(i) *states x and y are* similar*, denoted by $x \sim_s y$, if there is a process j such that* (a) *the states x and y agree modulo j and* (b) *there exists $i \neq j$ that is nonfailed in both x and y,*

(ii) *a set of states X is* similarity connected *if the graph $(X, \sim_s)$ induced by $\sim_s$ on X is connected.*

A well-known and useful property of the initial states for consensus, Con_0, is given by the following lemma.

LEMMA 5.5. *The set Con_0 is similarity connected.*

Proof. To prove that Con_0 is similarity connected we will show that every two initial states $x, y \in \mathsf{Con}_0$ are connected by a sequence of states in which each pair of neighbors are similar. Choose $x, y \in \mathsf{Con}_0$. For $0 \leq l \leq n$, define x^l by setting

$$x_j^l = \begin{cases} y_j & \forall j \leq l, \text{ and} \\ x_j & \forall j > l. \end{cases}$$

(Recall that $x_e = y_e$ by definition of Con_0.) Clearly, $x^l \in \mathsf{Con}_0$, and it is easy to check that $x^0 = x$ and $x^n = y$. Moreover, for every $0 < l \leq n$ we have that x^{l-1} and x^l agree modulo l, since the local states of the environment and of all processes, except possibly that of l, are equal. The assumption that no process is failed in an initial state (fault independence (ii)), and the fact that $n \geq 2$, imply that there is a process $i \neq l$ that is nonfailed in both x^{l-1} and x^l, and hence these two states are similar. ∎

We now formalize the intuition that similarity connectedness yields potence connectedness when processes may crash.

DEFINITION 5.6 (crashlike behavior). *Let X be a set of states of the system $\mathcal{S}$. We say that $\mathcal{S}$* displays crashlike behavior *with respect to X if the following condition holds. For every $x, y \in X$ and process j, if x and y agree modulo j, then there exist in $\mathcal{S}$ runs R^x and R^y and times $m_x, m_y \geq 0$ such that*

(i) *$r^x(m_x) = x$ and $r^y(m_y) = y$,*
(ii) *$r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and*
(iii) *every process $i \neq j$ that is not failed in x and in y is nonfaulty in R^x and in R^y.*

It is worth noting that the abstract definition of crashlike behavior is not restricted to crash failures. What the definition requires is that the state of j may be "hidden" from the rest of the processes indefinitely from some point. This can happen in models of process failures such as crash, omission, or Byzantine failures. It can also happen in models of link failures [35] or even in some cases when a failure may simply change the local state of a process, thereby effectively corrupting or erasing its memory.

A very useful relation between potence connectedness and similarity connectedness is given by the following lemma.

LEMMA 5.7. *Let $\mathcal{S}$ be a system satisfying the decision property, and let X be a similarity connected set of states of $\mathcal{S}$. If $\mathcal{S}$ displays crashlike behavior with respect to X, then X is potence connected.*

Proof. Since similarity connectedness is the transitive closure of $\sim_s$ and potence connectedness is the transitive closure of $\sim_p$, it suffices to show that for all $x, y \in X$, if $x \sim_s y$, then $x \sim_p y$. Assume $x \sim_s y$ and $\mathcal{S}$ displays crashlike behavior with respect to $X \supseteq \{x, y\}$. Then there exists a process j such that (i) the states x and y agree modulo j and (ii) there exists a process $i \neq j$ that is nonfailed in both x and y. Since $\mathcal{S}$ displays crashlike behavior with respect to $\{x, y\}$, part (i) implies that there exist in $\mathcal{S}$ runs R^x and R^y and times $m_x, m_y \geq 0$ such that (a) $r^x(m_x) = x$ and $r^y(m_y) = y$, (b) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and (c) every process $i \neq j$ that is nonfaulty in x and in y is nonfaulty in R^x and in R^y. Moreover, by (ii), there is at least one such nonfaulty process i. Since $\mathcal{S}$ satisfies the decision requirement, process i eventually decides in both runs. Let w be the value that i decides in R^x. Because $\mathtt{d}_i$ is write-once and appears in all local states of i, and since $r^x(m_x + k)_i = r^y(m_y + k)_i$ for all $k \geq 0$, we conclude that process i decides on w in R^y as well. It follows that both x and y are w-potent, and hence $x \sim_p y$. ∎

Notice that Lemma 3.8 holds also for a general model M. Also, recall from the

proof of Lemma 3.9 that when decision and validity hold, the initial state with all initial values 0 is 0-potent, while the one with all values 1 is 1-potent; this holds in the generic setting due to fault independence. Thus, an immediate consequence of Lemmas 3.8, 5.5, and 5.7 is a generalization of the well-known fact from [20] that when even a single process can crash, there must be a bipotent initial state for consensus.

THEOREM 5.8. *Let $\mathcal{S}$ be a system for consensus satisfying the decision and validity conditions. If $\mathcal{S}$ displays crashlike behavior with respect to Con_0, then there is a bipotent initial state in $\mathcal{S}$.*

6. Layering. In section 3.1 we defined a set of layers to facilitate the analysis of particular well-behaved runs of the shared-memory model. We now consider a similar operation for general models. Recall that given a finite execution R we denote by $R \odot \epsilon$ the execution that results from extending R by having the environment perform $\epsilon = (\mathsf{Sched}, a)$ in its final state. A run of $\mathcal{S} = \mathcal{S}(D, M, I)$ can thus be represented in the form $x \odot \epsilon_0 \odot \epsilon_1 \odot \cdots$, where $x \in I$ and $\epsilon_i \in \mathsf{ACT}_e$ for $i \geq 0$.

In some cases we are interested in single actions of the environment, while in others we may be interested in thinking of sequences of such actions as constituting a "round" or "layer" of the computation. In such cases, we will be interested only in those states that appear at the end of layers, and we want to ignore the intermediate states. Given a model $M = (\mathcal{G}_0, P_e, \tau, \Psi, \mathsf{FGen})$, we define a *layer* over ACT_e to be a nonempty finite sequence $\ell = \epsilon_1, \epsilon_2, \ldots, \epsilon_k$ of actions of the environment, $\epsilon_i \in \mathsf{ACT}_e$. If a protocol D is specified, given a state x, the round ℓ would lead from x to the state at the end of $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots \odot \epsilon_k = x \odot \ell$, provided each ϵ_i is an action that can be executed according to the environment protocol in the state at the end of $x \odot \epsilon_1 \odot \epsilon_2 \odot \cdots \odot \epsilon_{i-1}$. In this case we say that ℓ is *executable* at x. We denote the state at the end of the execution $x \odot \ell$ by $x \cdot \ell$.

One of the principles behind the use of layers and layerings is that we would like to ignore small steps and intermediate states and center our attention on interesting landmarks in the computation. Given a set of layers L, we define an L-*run* to be a pair $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha_{\mathsf{L}})$, where $r_{\mathsf{L}} : N \to \mathcal{G}$ defines an infinite sequence of states of $\mathcal{G}$, and $\alpha_{\mathsf{L}} : N \to \mathsf{L}$ is a sequence of layers. We will be interested in L-runs that describe runs of a system $\mathcal{S} = \mathcal{S}(D, M, I)$. We say that a run $R = (r, \alpha)$ of $\mathcal{S}$ *corresponds to* an L-run $R_{\mathsf{L}} = (r_{\mathsf{L}}, \alpha_{\mathsf{L}})$ if there is an infinite sequence $i_0 < i_1 < \cdots < i_k < \cdots$, such that $i_0 = 0$ and for all $k \geq 0$ we have both (a) $r(i_k) = r_{\mathsf{L}}(k)$ and (b) the sequence $\alpha(i_k), \alpha(i_k + 1), \ldots, \alpha(i_{k+1} - 1)$ is exactly the layer $\alpha_{\mathsf{L}}(k)$.

Intuitively, an L-run is a run that is obtained by starting at some initial state and repeatedly performing layers of L. The run R_{L} keeps track of the specific sequence of layers used, and of the states at the end of layers. Thus, the view presented in R_{L} is that the actions that the environment performs are in the form of whole layers. It is easy to check that there can be at most one run R corresponding to a given R_{L} in the system $\mathcal{S}(D, M, I)$. Suppose that $r_{\mathsf{L}}(0) \in I$ and for every $m \geq 0$ there is an execution $x^m \odot \ell_m$ in $\mathcal{S} = \mathcal{S}(D, M, I)$, where $x^m = r_{\mathsf{L}}(m)$ and $\ell_m = \alpha_{\mathsf{L}}(m)$. In this case, there is a unique run R of D consistent with M that corresponds to R_{L}. Notice, however, that R might still not be a run of D in M, because it may fail to satisfy the fairness condition Ψ of M. We will be interested in sets L of layers in which this cannot happen.

DEFINITION 6.1. *A nonempty set L of layers is called a* layering *of a model M if, for every protocol D, the following condition holds. Every run R of D consistent with M that corresponds to an L-run R_{L} is a run of D in M.*

Notice that for every set of layers, L, we can consider the runs of a protocol D in

which the environment performs actions according to the layers in this set. A set of layers is a layering if it satisfies the property that *every* run obtained by performing an infinite sequence of layers of L must satisfy the fairness conditions imposed by M, as specified by its admissibility condition Ψ. Roughly speaking, then, each layer must carry out a sufficient amount of work to guarantee that the fairness requirements imposed by Ψ are ultimately satisfied.

Given a system $\mathcal{S} = \mathcal{S}(D, M, I)$ and a layering L of M, we define the corresponding layered system $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, I)$ by

$$\mathcal{S}_{\mathsf{L}} = \{R_{\mathsf{L}} \mid \text{there is a run } R \text{ of } \mathcal{S} \text{ that corresponds to } R_{\mathsf{L}}\},$$

with Faulty predicate as follows. Since every run R_{L} of $\mathcal{S}_{\mathsf{L}}$ has associated a unique run R of the original system $\mathcal{S}$, the Faulty_D predicate from M can be extended to the runs of $\mathcal{S}_{\mathsf{L}}$ by defining $\mathsf{Faulty}_D(R_{\mathsf{L}}) = \mathsf{Faulty}_D(R)$ for all runs $R_{\mathsf{L}} \in \mathcal{S}_{\mathsf{L}}$. Thus, to figure out who the faulty processes in a run of $\mathcal{S}_{\mathsf{L}}$ are, we check the corresponding run of $\mathcal{S}$. Notice, however, that fault independence of $\mathcal{S}$ does not necessarily imply fault independence of $\mathcal{S}_{\mathsf{L}}$.

DEFINITION 6.2. *A* layering L *of a model* M satisfies fault independence *if every system of the form* $\mathcal{S}_{\mathsf{L}}(D, M, I)$ *satisfies fault independence.*

There are several obvious close correspondences between $\mathcal{S}_{\mathsf{L}}$ and $\mathcal{S}$ that follow directly from the definition of $\mathcal{S}_{\mathsf{L}}$ in terms of the runs of $\mathcal{S}$. First of all, $\mathcal{S}_{\mathsf{L}}$ inherits a number of "universal" properties from $\mathcal{S}$: Each of the properties of decision, agreement, and validity is satisfied by $\mathcal{S}_{\mathsf{L}}$ if it is satisfied by $\mathcal{S}$. In the other direction, "existential" properties of $\mathcal{S}_{\mathsf{L}}$ pass on to $\mathcal{S}$: If a state x is bipotent with respect to $\mathcal{S}_{\mathsf{L}}$, then it is bipotent with respect to $\mathcal{S}$. Moreover, if R_{L} is a bipotent run of $\mathcal{S}_{\mathsf{L}}$, then the run R of $\mathcal{S}$ that corresponds to R_{L} is a bipotent run with respect to $\mathcal{S}$.

Our use of layering to prove lower bound and impossibility results for consensus will be based on the close correspondence between $\mathcal{S}_{\mathsf{L}}$ and $\mathcal{S}$. The general idea is to assume there is a protocol D solving consensus in some particular model M. An appropriate layering L for M is then defined, for which it can be shown that there is a bipotent run (or in the case of a lower bound a bipotent prefix of a run) in $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, \mathsf{Con}_0)$.

For every state x of $\mathcal{S}_{\mathsf{L}}$ we define $\mathsf{L}(x)$ to be the set of the successors of x in $\mathcal{S}_{\mathsf{L}}$. More formally,

$$\mathsf{L}(x) = \{x \cdot \ell \mid \ell \in \mathsf{L} \text{ and } x \odot \ell \text{ is an execution of } \mathcal{S}_{\mathsf{L}}\}.$$

Provided the layering L guarantees that $\mathsf{L}(x)$ is potence connected, we can use the following theorem to prove the existence of a bipotent run in $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$. We can then apply Lemma 5.2 to show that the protocol D does not solve consensus.

THEOREM 6.3. *Let* L *be a layering of* M *satisfying fault independence, and let* $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$ *be a system for consensus satisfying the decision condition. Consider the layered system* $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M, \mathsf{Con}_0)$*, and assume that there is a bipotent initial state in* $\mathcal{S}_{\mathsf{L}}$*. If, for every state* x *of* $\mathcal{S}_{\mathsf{L}}$ *the set* $\mathsf{L}(x)$ *is potence connected in* $\mathcal{S}_{\mathsf{L}}$*, then there is a bipotent run in the original system* $\mathcal{S}$*.*

Proof. We will construct a bipotent L-run R^b_{L} in $\mathcal{S}_{\mathsf{L}}$, and the corresponding run R^b will be a bipotent run in $\mathcal{S}$. We obtain this run by starting from a bipotent initial state x^0 and constructing an infinite sequence of layers $\ell_0, \ell_1, \ldots$ from L, such that the state $x^m = x^0 \cdot \ell_0 \cdot \ldots \cdot \ell_{m-1}$ is bipotent with respect to $\mathcal{S}_{\mathsf{L}}$ for all $m \geq 0$. By construction, for all $m \geq 0$ we will have that $x^m \odot \ell_m$ is an execution in $\mathcal{S}_{\mathsf{L}}$, since it will be taken from $\mathsf{L}(x^m)$. This means that it is consistent with D and M to execute

the sequence of environment actions ℓ_m, and corresponding protocol actions starting in x^m, leading to x^{m+1}. Hence the run

$$R^b = x^0 \odot \ell_0 \odot \ell_1 \odot \cdots$$

will be a run of D consistent with M. The fact that L is a layering of M will then imply that R^b is a run of $\mathcal{S}(D, M, \mathsf{Con}_0)$. Recall that a bipotent state of $\mathcal{S}_\mathsf{L}$ is necessarily also a bipotent state of $\mathcal{S}$. Moreover, if a state of a run R is bipotent, then all states preceding x in R are also bipotent, by Lemma 5.1(i). Since there are infinitely many bipotent states in R^b (all the states x^m from the construction), it follows that all states of R^b are bipotent, so that R^b is a bipotent run of $\mathcal{S}$.

It remains to define x^0 and the sequence of layers ℓ_k. By assumption, there is a bipotent initial state in $\mathcal{S}_\mathsf{L}$. We shall choose this state to be x^0. Assume inductively we have chosen $\ell_0, \ldots, \ell_{m-1}$ so that $x^m = x^0 \cdot \ell_0 \cdot \cdots \cdot \ell_{m-1}$ is bipotent (with respect to $\mathcal{S}_\mathsf{L}$). Since x^m is bipotent, the set $\mathsf{L}(x^m)$ contains both 0-potent and 1-potent states. By assumption, $\mathsf{L}(x)$ is potence connected for every state x of $\mathcal{S}_\mathsf{L}$, and hence in particular $\mathsf{L}(x^m)$ is potence connected. It follows by Lemma 3.8 that there is a bipotent state $x' \in \mathsf{L}(x^m)$. By definition of $\mathsf{L}(x^m)$ there must also be a layer $\ell' \in \mathsf{L}$ such that $x' = x^m \cdot \ell'$. Set $\ell_m = \ell'$, and let $x^{m+1} = x^0 \cdot \ell_0 \cdot \cdots \cdot \ell_{m-1} \cdot \ell_m$. Clearly, $x^{m+1} = x^m \cdot \ell_m = x^m \cdot \ell' = x'$. It follows that x^{m+1} is bipotent, and we are done. □

We have essentially completed the description of our abstract framework for proving impossibility of consensus. Given a model M, the strategy would be to choose an appropriate layering L for M. With respect to an arbitrary protocol D, we use Theorem 5.8 to prove the existence of a bipotent initial state, and use Theorem 6.3 to extend this state into a bipotent run. Finally, from Lemmas 5.2 or 5.3 we conclude that this run is a counterexample to the decision property of consensus. We now apply this scheme to prove impossibility of consensus and lower bounds for models with synchronous message passing.

7. Synchronous message passing. We present two applications of the general framework to synchronous models: first for mobile failures and then for the classic crash failure model.

7.1. Impossibility for mobile failures. We illustrate the use of the abstract framework just described by proving a new impossibility result for consensus in the presence of a single mobile failure in the synchronous model. This model, denoted $M^\mathsf{m} = (\mathcal{G}_0^\mathsf{m}, P_e^\mathsf{m}, \tau^\mathsf{m}, \Psi^\mathsf{m}, \mathsf{FGen}^\mathsf{m})$, is described in section 4.2. Recall that M^m is the standard synchronous model with the failure assumption that in every round m there can be at most one process i_m some of whose messages are lost. The environment's protocol P_e^m is uniform in all states: nondeterministically choose a process $i \in \{1, \ldots, n\}$ and a set $T \subseteq \{1, \ldots, n\}$ and perform the action $(\{1, \ldots, n\}, (i, T))$, denoted simply by (i, T); all processes are scheduled to move in every round. The messages from i to members of T will be lost. Recall that Ψ^m makes no restrictions whatsoever: it consists of all possible runs consistent with M^m, and $\mathsf{FGen}^\mathsf{m}(D)(j, R) = \mathit{false}$ for all processes j, runs R, and protocols D.

To get the following impossibility result in M^m, we assume for contradiction that there is a protocol D solving consensus, and prove three basic claims:

(i) there exists a layering L for M^m that satisfies fault independence;
(ii) $\mathcal{S}_\mathsf{L} = \mathcal{S}_\mathsf{L}(D, M^\mathsf{m}, \mathsf{Con}_0)$ displays crashlike behavior with respect to every subset X of its states; and
(iii) for every state x in a layer of $\mathcal{S}_\mathsf{L}$, the set $\mathsf{L}(x)$ is potence connected.

The following proof establishes these claims in detail. The general scheme will be the same in the next sections (and we will not include as many details).

THEOREM 7.1. *No protocol solves the consensus problem in* M^{m}.

Proof. We start by choosing a layering for M^{m}. Each layer in this case will consist of a single action of the environment. We use $[k]$ to denote the set $\{1, \ldots, k\}$, with $[0]$ denoting the empty set. Define $\mathsf{L} = \{(i, [k]) : 1 \leq i \leq n \text{ and } 0 \leq k \leq n\}$. To see that L is a layering of M^{m}, observe that every layer is an action of the environment and the admissibility condition Ψ^{m} in this model makes no restrictions. Hence, for every protocol D, every run resulting from an infinite sequence of layers of L is immediately a run of M^{m} as desired. Moreover, L satisfies (trivially) fault independence because no processes are ever considered faulty in M^{m}.

Assume by way of contradiction that there is a protocol D solving consensus in M^{m}. Then the system $\mathcal{S} = \mathcal{S}(D, M^{\mathsf{m}}, \mathsf{Con}_0)$ is a system for consensus that satisfies decision, agreement, and validity. Given that no process is ever considered faulty in this model, we have by Lemma 5.2 that if there is a bipotent run in $\mathcal{S}$, then the decision property must fail in $\mathcal{S}$, contradicting the assumption that D solves consensus. We will complete the proof by demonstrating the existence of a bipotent run in $\mathcal{S}$ via Theorem 6.3. Hence, we next consider the layered system $\mathcal{S}_{\mathsf{L}} = \mathcal{S}_{\mathsf{L}}(D, M^{\mathsf{m}}, \mathsf{Con}_0)$.

Our next goal is to show that there is a bipotent initial state in $\mathcal{S}_{\mathsf{L}}$. Notice that, by definition of L, the environment is able to "silence" any given process j from a given state on, simply by performing the layer $(j, [n])$ in all subsequent rounds. The other processes will have no way to learn anything about j's state. It immediately follows that $\mathcal{S}_{\mathsf{L}}$ displays crashlike behavior with respect to every subset X of its states. More precisely, we verify that Definition 5.6 holds. Let x, y be two states in a layer of $\mathcal{S}_{\mathsf{L}}$, such that x and y agree modulo j. Let $\hat{R}^x$ and $\hat{R}^y$ be prefixes of L-runs ending in x and y, respectively. Then the L-runs $R^x = \hat{R}^x \odot (j, [n])^\infty$ and $R^y = \hat{R}^y \odot (j, [n])^\infty$ are in $\mathcal{S}_{\mathsf{L}}$. Letting $R^x = (r^x, \alpha^x)$ and $R^y = (r^y, \alpha^y)$, it is straightforward to verify that there are times $m_x, m_y \geq 0$ such that

(i) $r^x(m_x) = x$ and $r^y(m_y) = y$,
(ii) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and
(iii) every process $i \neq j$ that is not failed in x and in y is nonfaulty in R^x and in R^y.

Recall that $\mathcal{S}$ is a system for consensus satisfying decision and validity. As mentioned above, $\mathcal{S}_{\mathsf{L}}$ inherits these properties from $\mathcal{S}$. It now follows from Theorem 5.8 that there is a bipotent initial state in $\mathcal{S}_{\mathsf{L}}$.

We are finally in a position to apply Theorem 6.3 to derive the existence of a bipotent run R^b in $\mathcal{S}_{\mathsf{L}}$. To do so, we still need to show that for every state x of $\mathcal{S}_{\mathsf{L}}$, the set $\mathsf{L}(x)$ is potence connected. For every pair of processes j, j' we have that $x \cdot (j, [0]) = x \cdot (j', [0])$ because in both cases no messages are lost in the round following x. It follows that $x \cdot (j, [0]) \sim_s x \cdot (j', [0])$. Moreover, for every $k < n$ we have that $x \cdot (j, [k]) \sim_s x \cdot (j, [k+1])$, because the two states can differ only in the state of process $k + 1$. It follows that $\mathsf{L}(x)$ is similarity connected for all x. That $\mathsf{L}(x)$ is potence connected now follows from Lemma 5.7, since $\mathcal{S}$ displays crashlike behavior with respect to $\mathsf{L}(x)$. We thus obtain that there exists a bipotent run R^b in $\mathcal{S}_{\mathsf{L}}$, and the run of $\mathcal{S}$ that corresponds to R^b is bipotent with respect to $\mathcal{S}$, as desired. ∎

Theorem 7.1 illustrates the fact that it doesn't take much to make consensus impossible. The adversary in M^{m} has fairly limited powers. Furthermore, we obtained the impossibility with a layering that restricts the adversary even more. We remark that this result can be obtained by a modification of the proof of a theorem by Santoro

and Widmayer in [35]. Their analysis involved attaining consensus in the presence of communication link failures. Their proof is the only one we have found that predates this paper and that uses a bipotence-based argument in the style of Fischer et al. [20] in the synchronous context.

7.2. The synchronous lower bound. The analysis we performed for the mobile failure model M^{m} in the synchronous case should, intuitively, apply equally well to the standard t-resilient case in the synchronous model. In this model there is a bound of t on the total number of processes who may fail in the run, and a process some of whose messages are lost is considered faulty. The well-known lower bound for this case (originally due to [18, 15]) states that every consensus protocol must require at least $t + 1$ rounds in its worst-case run. Roughly speaking, any prefix consisting of t rounds of a run of M^{m} can be viewed as a prefix of a run in the standard omissions failure model with at most t failures. (Formally, the only modification required would be to have the environment's state record processes that have omitted in the past.) The impossibility of solving consensus in M^{m} therefore immediately implies that there can be no protocol solving consensus in t rounds in the omissions model. For if one existed, it would also solve consensus in M^{m}. This argument immediately implies the $(t+1)$-round lower bound for the omissions and Byzantine failure models. We cannot, however, use the impossibility for M^{m} to derive the lower bound for the crash failure model. Nevertheless, even in the crash failure model one might expect to prove that there will exist a bipotent state at the end of round t, and thus derive the $(t + 1)$-round lower bound just as in our analysis for M^{m}. A close inspection, however, shows that things are not *that* simple. There will typically not need to be a bipotent state at the end of round t. But the essence of this idea still works.

We shall now provide the lower bound analysis for the crash failure model and a number of related failure models at once. We assume the failure model satisfies the following: (i) in the first round in which a process fails, the environment can block the delivery of an arbitrary subset of its messages; (ii) the environment can silence a faulty process forever in all rounds after the first one in which it fails; and (iii) the environment's local state keeps track of the processes that have failed. It is easy to check that it satisfies fault independence. The layers we will focus on will consist of single environment actions, each corresponding to a single round of the synchronous model. Specifically, we consider two kinds of actions by the environment:

- clean This environment action, applicable at all states, involves no new process failure. Messages of failed processes are not delivered, but all messages of nonfailed processes are delivered.
- (j, k) This action is applicable to a state x only if fewer than t processes are failed at x, and process j is not failed at x. As before, messages of failed processes are not delivered, while all messages of nonfailed processes other than j are delivered. The messages of j act as with the action $(j, [k])$ in M^{m}: Those addressed to processes up to and including k are not delivered, while the others are delivered.

We denote the layering consisting of all actions of these types by L^t. Notice that the number of processes that fail in a run of $\mathcal{S}_{\mathsf{L}^t}$ is at most t, since once t processes are failed at a state, all later layers will be clean. It is now straightforward to show the following.

LEMMA 7.2. *Let M be one of the standard synchronous t-resilient models with either crash, omission, or Byzantine failures. Let D be a protocol for the processes in the model M, and let $\mathcal{S} = \mathcal{S}(D, M, \mathsf{Con}_0)$. Finally, let x be a state of $\mathcal{S}_{\mathsf{L}^t}$. Then*

(i) L^t *is a layering of* M *that satisfies fault independence;*
(ii) *for every state* x *of* $\mathcal{S}_{\mathsf{L}^t}$ *the set* $\mathsf{L}^t(x)$ *is similarity connected; and*
(iii) *if no more than* $t-2$ *processes are failed at* x, $\mathsf{L}^t(x)$ *is potence connected.*

Proof. For part (i), recall that in a run of $\mathcal{S}_{\mathsf{L}^t}$ the number of failed processes does not exceed t. Moreover, every L^t action is a legal action for the environment in the model M. It follows that L^t is a layering of $\mathcal{S}$. Also, it is easy to check fault independence.

Part (ii) follows the pattern from M^{m}: If t processes are failed at x, then the set $\mathsf{L}^t(x)$ consists of the singleton state $x \odot \mathsf{clean}$ and is hence trivially similarity connected. If fewer than t processes are failed at x, then $\mathsf{L}^t(x) = \{x \odot \mathsf{clean}\} \cup \{x \odot (j,k) | j \text{ not failed in } x\}$. It is straightforward to check that $x \odot (j,k) \sim_s x \odot (j,k+1)$ for all j and $k < n$; additionally, $x \odot \mathsf{clean} \sim_s x \odot (j,0)$. It follows that $\mathsf{L}^t(x)$ is similarity connected in either case.

Finally, for part (iii), notice that $\mathcal{S}_{\mathsf{L}^t}$ displays crashlike behavior with respect to every set X consisting of states in each of which at most $t-1$ processes are failed. Since an L^t-action fails at most one new process, if at most $t-2$ processes are failed in x, then $\mathsf{L}^t(x)$ is similarity connected by part (ii), and at most $t-1$ processes are failed in any given state of $\mathsf{L}^t(x)$. It now follows from Lemma 5.7 that $\mathsf{L}^t(x)$ is potence connected and we are done. ∎

We thus have the following lemma.

LEMMA 7.3. *Consider the system* $\mathcal{S}_{\mathsf{L}^t} = \mathcal{S}_{\mathsf{L}^t}(D, M, \mathsf{Con}_0)$. *Let* x^0 *be a bipotent state in a layer of* $\mathcal{S}_{\mathsf{L}^t}$ *in which no more than* f *processes are failed. Then there is an* L^t*-execution with states* $x^0, x^1, \ldots, x^{t-f-1}$, *such that* x^{t-f-1} *is bipotent and no more than* $t-1$ *processes are failed in* x^{t-f-1}.

Proof. We prove by induction on m, for $0 \le m \le t-f-1$, that an execution of the desired form exists, with x^m bipotent and where no more than $m+f$ processes are failed in x^m. The basis $m=0$ holds by assumption. Assume inductively that the claim holds for $m < t-f-1$. Thus, we have that $m+f < t-1$ processes are failed in x^m. By Lemma 7.2(iii) we have that $\mathsf{L}^t(x^m)$ is potence connected, and by an argument similar to the one in the proof of Theorem 6.3, there is a bipotent state $x^{m+1} \in \mathsf{L}^t(x^m)$. By definition of L^t, the number of failed processes in x^{m+1} is at most $m+f+1 \le t$. ∎

Since Theorem 5.8 guarantees the existence of a bipotent initial state with $f=0$ failed processes, Lemma 7.3 immediately implies the existence of a bipotent state x^{t-1} at the end of round $t-1$. By Lemma 5.3, this gives us a t-round lower bound for consensus. The true $(t+1)$-round lower bound is obtained by showing that two rounds are still necessary after a bipotent state.

LEMMA 7.4. *Under the conditions of Lemma* 7.2, *assume that* $t \le n-2$ *and let* D *be a protocol for consensus. If* $\hat{x}$ *is a bipotent state in a layer of* $\mathcal{S}_{\mathsf{L}^t}$, *then there is a state* $y \in \mathsf{L}^t(\hat{x})$ *in which at least one nonfailed process has not decided.*

Proof. Notice that a state x with t failed processes cannot be bipotent, since there is a unique infinite L^t extension starting at x. Hence, to be bipotent, the state $\hat{x}$ can have no more than $t-1$ failed processes. By Lemma 7.2(ii), we have that $\mathsf{L}^t(\hat{x})$ is similarity connected. Since $\hat{x}$ is bipotent, there are states $y^0, y^1 \in \mathsf{L}^t(\hat{x})$ such that y^0 is 0-potent and y^1 is 1-potent. The similarity connectivity of $\mathsf{L}^t(\hat{x})$ implies the existence of states $z^0, z^1 \in \mathsf{L}^t(\hat{x})$ (not necessarily distinct) satisfying $z^0 \sim_s z^1$ that are 0- and 1-potent, respectively. Recall that all states of $\mathcal{S}_{\mathsf{L}^t}$ have at most t faulty processes. Since $t \le n-2$ and z^0 and z^1 agree modulo j for some j, it follows that there is at least one process $i \ne j$ such that i is not failed in both states and $z_i^0 = z_i^1$. Assume by way of

contradiction that every nonfailed process is decided in both z^0 and z^1. In particular, i is decided, say with value v. Agreement implies that in both states, every nonfailed process decides v. It follows that both z^0 and z^1 are v-potent, and neither of them is $(1-v)$-potent, contradicting the assumption that one of them is 0-potent and the other is 1-potent. □

We can now put the two results together and obtain the desired lower bound.

THEOREM 7.5. *Let $t \leq n-2$. Every t-resilient protocol for consensus in synchronous models where faulty processes can either crash, omit, or behave in a Byzantine fashion has a run in which decision requires at least $t+1$ rounds. Moreover, for $t = n-1$, every such protocol has a run in which decision requires at least t rounds.*

This result was first proved for crash failures by Dolev and Strong [15], and the latest version of the proof is in [16]. Our proof here is the first one we are aware of that is in the style and spirit of the impossibility proofs for the asynchronous case.[7] Moreover, we feel that it is even simpler than the one of [16]. In addition to generalizing the lower bound for t-resilient consensus, we feel that our proof provides further insight into the structure of consensus protocols in this model. Let us briefly consider an example. It is well known [33] that there are t-resilient consensus protocols that are guaranteed to decide in precisely $t+1$ rounds. Thus, the worst-case lower bound of Dolev and Strong is tight. We call a protocol in which consensus is always reached in at most $t+1$ rounds *fast*. We can now show the following.

LEMMA 7.6. *Let D be a fast t-resilient consensus protocol. For every execution with states $x^0, x^1, \ldots, x^k, x^{k+1}$ of D, if at most k processes have failed by x^k, and the $(k+1)$st round is failure-free, then x^{k+1} cannot be bipotent.*

Proof. By assumption, only k processes have failed by x^{k+1}. If x^{k+1} is bipotent, then by Lemma 7.3 it can be extended to a run with a bipotent state x^t at the end of t rounds. By Lemma 7.4, two more rounds are necessary for agreement in the worst case, contradicting the assumption that D is fast. □

Clearly, Lemma 7.3 also partially describes the situation in runs in which potentially more than one process can crash in a given round. It matches the upper (and lower) bounds given in [16], which show roughly that if in some execution $k+w$ crashes are detected by the end of round k, then agreement can be secured by the end of round $t+1-w$. Hence, by allowing $k+w$ crashes by the end of round k, the environment has essentially "wasted" w faults in its quest to delay agreement. Lemma 7.3 guarantees that the environment has not lost more than w rounds in this case.

8. Asynchronous message passing. In section 3 we illustrated the layering technique by proving impossibility of consensus in the asynchronous shared memory model. The proof was based on a "permutation layering," in which processes move one at a time and there is very little concurrency. Our proof of impossibility for M^{m}, on the other hand, considered every synchronous round to be a layer, and such a layer contains a great deal of concurrency. In this section we apply our framework to prove impossibility for the asynchronous message-passing model. We will give two proofs: one using a permutation layering based on the proof of section 3 and the other using a "synchronic layering," which is very close to the layering used in the synchronous model M^{m}. We do this to illustrate the closeness of the different models and to show the choice and flexibility that are often provided by the layering technique.

[7] We have recently been informed that Aguilera and Toueg [2] have independently and slightly later given a proof for this result using a bipotence argument. The structure of their proof is similar to ours. Bar-Joseph and Ben-Or also reported having found some of the arguments in [7].

As we have already seen in the shared-memory case, in asynchronous models "slow" behavior of processes can be used to imitate the omitting behavior in M^{m}. The small but crucial difference now will be that, in the asynchronous model, delayed messages will nevertheless eventually be delivered or, similarly, a slow process that is about to write a variable will ultimately write the value. In the synchronous model M^{m}, the lost messages are gone forever. Hence, to perform a careful analysis of the round by round evolution, we need to consider as part of the state (i.e., in the environment's local state) the status of the messages in transit or, similarly, the current values of shared variables. In this sense, our treatment goes slightly beyond the scope of most of the recent work on topological approaches, in which the state of the environment does not play a role, and asynchronous message-passing models are often somewhat subtle to deal with.

Consider the standard (see, for example, [20, 31]) asynchronous message-passing model M^{mp} in which processes communicate by message passing and both processes and communication are asynchronous. A process is faulty if it is scheduled to move only a finite number of times in an infinite run. The celebrated result of Fischer, Lynch, and Paterson [20] proving the impossibility of solving consensus in the presence of a single crash failure was carried out in this model. In the asynchronous message-passing model M^{mp}, an action for i is a pair: it contains a local action on the variables of the local state, and a communication action, which is either skip_i that does nothing, or is a $\mathsf{send}_i(j, m)$ action, specifying the sending of the message m to process j. The effect of a $\mathsf{send}_i(m, j)$ action is to change the environment's state by recording the new message as being in the channel between i and j. For the definition of similarity among states in this model, we consider the j-component of the environment's state to consist of the set of channels whose destinations are process j. Since channels are point-to-point in this model, components corresponding to different processes are disjoint. The environment's actions consist of either scheduling a process i to move, or delivering a set of one or more messages that have been sent and not yet delivered to their destination. The fairness conditions in this model are that (a) at most one process can fail in any given run and (b) every message that is sent to a process that does not fail in the run must eventually be delivered.

In this model we consider a layering L^{mp} consisting of all layers of the following three types:

- $[p_1, \ldots, p_n]$,
- $[p_1, \ldots, p_{n-1}]$, and
- $[p_1, \ldots, p_{k-1}, \{p_k, p_{k+1}\}, p_{k+2}, \ldots, p_n]$ with $k < n$.

The first two layers are of the same form as in the layering L^{rw} for the shared-memory model. In a layer of L^{mp}, when a process i is scheduled to *move* it first performs its action, and then it receives all the messages that have been sent to i by that point and have not yet been delivered (if any exist). The third type of layer is considered a *full* layer, since it involves all processes as the first type does. In this case, the processes take steps in the linear order given by the sequence, except that p_k and p_{k+1} move *concurrently*. Thus, a message sent by one of these two processes to the other will not be delivered in the current layer. The reason we use the new third type of layer is that with this particular layering, a transposition in a layer of the first two types can change the local state of both p_k and p_{k+1}. Whereas in the case of L^{rw} a move by a process involved either a read or a write, in L^{mp} when a process moves it may both send a message and receive messages. Adding a layer of the third type enables us to take smaller steps, in which at most one process at a

time will change its state. We identify a layer of the first type with the corresponding permutation on $\{1,\ldots,n\}$. For a permutation $\pi = [p_1,\ldots,p_n]$, we denote the layer $[p_1,\ldots,p_{k-1},\{p_k,p_{k+1}\},p_{k+2},\ldots,p_n]$ of the third type by $\pi_{\{k,k+1\}}$. Notice that the set notation in layers of the third type is justified, since the exact same sequence of operations occurs in $\pi_{\{k,k+1\}}$ and $\pi_{\{k+1,k\}}$. Hence, for every state x of $\mathcal{S}_{\mathsf{L}^{mp}}(D)$ and permutation π we have that $x \cdot \pi_{\{k,k+1\}} = x \cdot \pi_{\{k+1,k\}}$.

We can show the following.

LEMMA 8.1. *Let D be a protocol for the processes in the model M^{mp}.*

(i) *L^{mp} is a layering of M^{mp} that satisfies fault independence;*

(ii) *$\mathcal{S}_{\mathsf{L}^{mp}} = \mathcal{S}_{\mathsf{L}^{mp}}(D, M^{mp}, \mathsf{Con}_0)$ displays crashlike behavior with respect to every subset X of its states; and*

(iii) *for every state x in a layer of $\mathcal{S}_{\mathsf{L}^{mp}}$, the set $\mathsf{L}^{mp}(x)$ is potence connected.*

Proof. (i) Let R be an L^{mp}-run. By definition, R consists of an infinite sequence of layers from L^{mp}. Since in each layer L of L^{mp} at least $n-1$ processes move, in an infinite number sequence there can be at most one process that fails to take an infinite number of steps. It follows that at most one process can be faulty in R. Since a process that moves receives all of the pending messages sent to it, all messages sent to nonfaulty processes in R are delivered. It follows that the run R' that corresponds to R is admissible. It is, in addition, easy to check that L^{mp} satisfies fault independence, since M^{mp} satisfies no finite failure.

Part (ii) follows from the fact that the environment can halt the operation of an arbitrary process j at any given state by repeatedly performing layers of the form $[1,2,\ldots,j-1,j+1,\ldots,n]$ from that state on.

It remains to show part (iii). First notice that just as in the case of L^{rw} we have that

$$x \cdot [p_1,\ldots,p_{n-1}] \cdot [p_n,p_1,\ldots,p_{n-1}] \quad = \quad x \cdot [p_1,\ldots,p_n] \cdot [p_1,\ldots,p_{n-1}],$$

since in both cases exactly the same actions take place in the same order in the two layers following x. Lemma 5.1(ii) implies that $x \cdot [p_1,\ldots,p_n] \sim_p x \cdot [p_1,\ldots,p_{n-1}]$, and it follows that every state of $\mathsf{L}^{mp}(x)$ obtained from x by a layer of the second type is potence connected to a state obtained by a (full) layer of the first type. Let X_f be the subset of $\mathsf{L}^{mp}(x)$ consisting of states of the form $x \odot \mathsf{L}$, where L is a full layer. It remains to show that X_f is potence connected. Since, by part (ii), $\mathcal{S}_{\mathsf{L}^{mp}}(D)$ displays crashlike behavior with respect to X_f, Lemma 5.7 implies that it suffices to show that the set X_f is similarity connected.

We first claim that, for a full layer $L = \pi$ of the first type and an index $k < n$, we have that

$$x \cdot \pi \quad \sim_s \quad x \cdot \pi_{\{k,k+1\}}.$$

To see why, recall that according to the environment's actions in a layer of the types we are considering, a process that is scheduled first performs its action and then receives the messages that were sent to it and are still in transit. Thus, the actions of a process p_j in the layer $x \odot \pi$ depend only on p_j's local state in x. It now follows that every process p_i with $i \neq k+1$ performs the same actions and receives the same messages in the layer following x in both $x \odot \pi$ and $x \odot \pi_{\{k,k+1\}}$. The only process whose state at the end of the layer may differ in the two cases is p_{k+1}. If p_k sends p_{k+1} a message, it will be delivered in the first case and will remain in the channel from p_k to p_{k+1} in the second case. Since the channel from p_k to p_{k+1} is part of the

p_{k+1}-component of the environment's state,[8] the two states agree modulo p_{k+1} and are thus similar. The claim follows.

Finally, we use this last claim to argue as in the shared-memory case. Recall (proof of Lemma 3.5) that $\mathsf{Tr}(k,\pi)$ is the permutation obtained by transposing the kth and $(k+1)$st elements in π. Since the order of the concurrent processes in a layer of the third type is immaterial and since $\pi_{\{k+1,k\}} = \mathsf{Tr}(k,\pi)_{\{k,k+1\}}$ we obtain

$$x \cdot \pi \quad \sim_s \quad x \cdot \pi_{\{k,k+1\}} \quad = \quad x \cdot \pi_{\{k+1,k\}} \quad = \quad x \cdot \mathsf{Tr}(k,\pi)_{\{k,k+1\}} \quad \sim_s \quad x \cdot \mathsf{Tr}(k,\pi),$$

and we are done, since any layer of the first type can be transformed into any other by a sequence of transpositions. □

Given Lemma 8.1, we can use Theorem 6.3 which, together with Theorem 5.8 and Lemma 5.2, now yields the following.

THEOREM 8.2. *No protocol solves the consensus problem in M^{mp}.*

Observe that the layers in a given layering need not be of the same length. Indeed, we saw both full and nonfull layers being used in the permutation layerings above. In addition, in the permutation layerings we discussed above each process performed at most one action in a layer. This was chosen for simplicity. In [32], other variants of permutation layerings are given in which a process may perform many actions in a layer.

8.1. A synchronic layering. We now sketch a second type of layering for the asynchronous models we have considered. This one, which we call the *synchronic* layering, imitates the synchronous model and yields layered systems of M^{mp} whose runs are very close in structure to those of M^{m}. We shall describe it for the message-passing model, and a completely analogous treatment works for the asynchronous shared memory model sketched in section 3 as well.

The synchronic layering, denoted by L^s, is defined as follows. The layers are denoted by (j,A) or (j,k) with $1 \le j \le n$ and $0 \le k \le n$. In the sequence (j,A), all processes except j move, and then they all receive whatever outstanding messages are addressed to them (including messages that have just been sent). The "A" in (j,A) stands for process j being *absent* from the layer. In an action of the form (j,k), all n processes move simultaneously, and then all outstanding messages to the processes are delivered as before, with one exception: If j's current action is $\mathsf{send}_j(\ell,m)$ and $\ell \le k$, then this message remains outstanding and is not delivered. Notice that the actions $(j,0)$ are all identical and independent of j; they are written this way for ease of exposition. We think of j as being the "slow" process in the layer defined by (j,n) or (j,A).

As with the permutation layering, a layer in the synchronic case contributes sufficiently to the fairness requirements made by the model: At least $n-1$ processes get to move in every layer, and all messages that have been sent by the previous layer to a process that moves are guaranteed to be delivered by the end of the current layer. It follows that L^s is a layering of M^{mp}, and it is easy to check that it satisfies fault independence. Crashlike behavior is immediate as well. To complete the impossibility argument, one needs to show that $\mathsf{L}^s(x)$ is potence connected. The argument for this uses elements from the proof for M^{m} and from the proof in the case of permutation layerings. For every j and every $k < n$, the states $x \odot (j,k)$ and $x \odot (j,k+1)$ agree

[8] This is the place where we make use of the notion of j-component in the definition of similarity. Intuitively, we expect this will be needed whenever message-passing models are considered, in which messages can be delayed in the channels for more than one "round."

modulo process $k+1$; they can differ at most in the state of $k+1$ and of the channel from j to $k+1$, which is part of the $k+1$-component of the state. It follows that the set of states of the form $x \odot (j,k)$ is similarity connected, and by Lemma 5.7 this set is also potence connected.

The argument that all of $\mathsf{L}^s(x)$ is potence connected uses the diamond property that

$$x \odot (j,n) \odot (j,A) \quad = \quad x \odot (j,A) \odot (j,0).$$

To see why this equality is true, notice that in both $x \odot (j,n)$ and $x \odot (j,A)$ no process receives a message from j in the last round. These messages are sent, based on j's local state in x, in the last round of $x \odot (j,n)$, but are only received in the following round. In the last round of $x \odot (j,A) \odot (j,0)$ process j sends messages to everyone, and these messages are all received in that round. Notice, however, that these messages are sent before j has received any new messages following the state x. Hence, the messages it sends are again based on j's local state in x. It follows that the same messages are sent by j in the last round of $x \odot (j,A) \odot (j,0)$ and in the last round of $x \odot (j,n)$. Moreover, in both cases these messages are received in the second round following x. It follows that $x \odot (j,n) \odot (j,A)$ and $x \odot (j,A) \odot (j,0)$ are equal.

9. Conclusions. Roughly speaking, we have seen that invariably, when the processes follow a protocol D in a given model M, the environment has a simple best strategy to delay consensus: Start with a bipotent state and, for as long as possible, find a layer that will cause a transition from the current bipotent state to a successor state that is also bipotent. If there exists a layering function L for which $L(x)$ can be shown to be potence connected for all x, then we obtain impossibility. For lower bounds the analogy is almost (but not quite) complete.

Interestingly, some of the layering functions for the different models turn out to be extremely similar. Moreover, we believe that the vast majority of lower bounds and impossibility results for consensus in the different models can be cast in terms of such layering functions, with essentially the same proof. We feel that this uniformity provides new insight into the inherent structure of the consensus problem and helps pinpoint, in a fairly model-independent way, the reason why consensus is difficult.

Our analysis emphasized and focused on the role of connectivity in the structure of consensus. Indeed, it is straightforward to convert our proofs of the existence of a bipotent state to ones that show the following, roughly: For a set X of states and a layering function L, define $L(X) = \cup_{x \in X} L(x)$. Let $X_0 = \mathsf{Con}_0$, and inductively define $X_{k+1} = L(X_k)$. The proof of Lemma 5.8 shows that X_0 is similarity connected and potence connected (in the presence of a single failure). For the layering functions we have been discussing it is not hard to modify our proofs to show that, roughly, if X_k is potence connected then so is X_{k+1}. Therefore, we have two possible styles of proofs. First is the FLP [20] style, which we followed in the paper. The strategy in this case is to show that there is a bipotent initial state and then find a sequence of successor bipotent states by proving *locally* the connectivity of the successors of a bipotent state. The other, more *global*, type of proof is closer to the topological approach of works such as [10, 26, 23, 34] and, in particular, [24]. It shows that each layer X_k is connected, and since it has at least one 0-potent and one 1-potent state, then it must have a bipotent state. The connectivity of these sets is essentially a topological property (although clearly the analysis involves no deep topology). As long as these sets are connected, Lemma 3.8 guarantees that there is a bipotent state in the set.

In this paper we dealt with consensus only, but the connectivity properties we prove hold for arbitrary decision problems as well, and we pursue this line further in a sequel paper (as described in [32]), where we show how our work generalizes the necessary conditions of [8] to other models, perhaps more remarkably to the synchronous model.

It is often useful to ask what knowledge [17] about the state is required by a process in order to be able to reach a decision. We note that once the protocol is fixed, as long as the state is bipotent, even an observer with complete information about the state cannot determine the final outcome. In a precise sense, no knowledge about the actual state can help the process decide at that point. Indeed, this has been formally captured in Lemma 5.3. Once the state ceases to be bipotent (and becomes "unipotent"), however, information about the state can be of use in determining what value a process should decide on. In fact, the proof of Lemma 7.4 shows that at the first instant along a run in which the state ceases to be bipotent in the t-resilient synchronous case, there are often many processes who have insufficient knowledge to be able to decide. An additional round is sufficient for providing them with this knowledge, in which case they can safely decide. In summary, as long as the state is bipotent, there are "structural" reasons why decisions cannot be taken. After that, the reasons merely involve disseminating the relevant information to all relevant parties. In particular, in the $(t+1)$-round lower bound (Theorem 7.5), t rounds are paid on account of topological reasons (the need to disconnect the set of global states), while an additional round is paid on account of the insufficient knowledge available after t rounds. A similar phenomena may occur in other topology-related problems, such as the k-set consensus problem in the synchronous case [14], where the tight lower bound is $\lfloor t/k \rfloor + 1$ rounds.

Acknowledgments. We would like to thank Juan Garay for bringing to our attention the mobile failure model and the related reference [35]. Special thanks to Eric Ruppert for pointing out an error in the definition of the message-passing asynchronous layering in [32]. We thank Marcos Aguilera, Bernadette Charron-Bost, Idit Keidar, and the referees for a very careful reading of an earlier version of the paper and many useful comments.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.

[2] M.K. Aguilera and S. Toueg, *A simple bivalency-based proof that t-resilient consensus requires $t+1$ rounds*, Inform. Process. Lett., 71 (1999), pp. 155–158.

[3] H. Attiya, A. Bar-Noy, and D. Dolev, *Sharing memory robustly in message-passing systems*, J. ACM, 42 (1995), pp. 124–142.

[4] H. Attiya and S. Rajsbaum, *The combinatorial structure of wait-free solvable tasks*, in Proceedings of the 10th International Workshop on Distributed Algorithms (WDAG), Lecture Notes in Comput. Sci. 1151, O. Babaoglu and K. Marzullo, eds., Springer-Verlag, New York, 1996, pp. 321–343.

[5] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw–Hill, New York, 1998.

[6] B. Awerbuch, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.

[7] Z. Bar-Joseph and M. Ben-Or, *A tight lower bound for randomized synchronous consensus*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1998, pp. 193–199.

[8] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed 1-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.

[9] O. Biran, S. Moran, and S. Zaks, *Tight bounds on the round complexity of distributed* 1*-solvable tasks*, Theoret. Comput. Sci., 145 (1995), pp. 271–290.
[10] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computations*, in Proceedings of the 1993 ACM Symposium on Theory of Computing (STOC), ACM, New York, 1993, pp. 91–100.
[11] E. Borowsky and E. Gafni, *A simple algorithmically reasoned characterization of wait-free computations*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1997, pp. 189–198.
[12] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum *The BG Distributed Simulation Algorithm*, Distrib. Comput., 14 (2001), pp. 127–146.
[13] T. Chandra and S. Toueg, *Unreliable failure detectors for asynchronous systems*, in Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1991, pp. 257–272.
[14] S. Chaudhuri, M.P. Herlihy, N. Lynch, and M.R. Tuttle, *Tight bounds for k-set agreement*, J. ACM, 47 (2000), pp. 912–943.
[15] D. Dolev and H.R. Strong, *Authenticated algorithms for Byzantine agreement*, SIAM J. Comput., 12 (1983), pp. 656–666.
[16] C. Dwork and Y. Moses, *Knowledge and common knowledge in a Byzantine environment: Crash failures*, Inform. and Comput., 8 (1990), pp. 156–186.
[17] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi, *Reasoning about Knowledge*, MIT Press, Cambridge, MA, 1995.
[18] M.J. Fischer and N.A. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process. Lett., 14 (1982), pp. 183–186.
[19] M.J. Fischer, *The Consensus Problem in Unreliable Distributed Systems (a Brief Survey)*, Research report YALE/DCS/RR-273, Dept. Comp. Sci., Yale University, New Haven, CT, 1983.
[20] M.J. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed commit with one faulty process*, J. ACM, 32 (1985), pp. 374–382.
[21] E. Gafni, *Round-by-round fault detectors: Unifying synchrony and asynchrony*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1998, pp. 143–152.
[22] M.P. Herlihy, *Wait-free synchronization*, ACM ToPLaS, 13 (1991), pp. 123–149.
[23] M.P. Herlihy and S. Rajsbaum, *A primer on algebraic topology and distributed computing*, in Computer Science Today, Lecture Notes in Comput. Sci. 1000, Jan van Leeuwen, ed., Springer-Verlag, New York, 1995, pp. 203–217.
[24] M.P. Herlihy, S. Rajsbaum, and M.R. Tuttle, *Unifying synchronous and asynchronous message-passing models*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1998, pp. 133–142.
[25] M.P. Herlihy and S. Rajsbaum, *New perspectives in distributed computing (invited lecture)*, in Proceedings of the 24th International Symposium Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 1672, M. Kutylowski, L. Pacholski, and T. Wierzbicki, eds., Springer-Verlag, New York, 1999, pp. 170–186.
[26] M.P. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J. ACM, 46 (1999), pp. 858–923.
[27] G. Hoest and N. Shavit, *Towards a topological characterization of asynchronous complexity*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1997, pp. 199–208.
[28] I. Keidar and S. Rajsbaum, *On the Cost of Fault-Tolerant Consensus When There Are No Faults—A Tutorial*, Technical report MIT-LCS-TR-821, MIT, Cambridge, MA, 2001.
[29] M.C. Loui and H.H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, in Parallel and Distributed Computing, Adv. Comp. Res. 4, F.P. Preparata, ed., JAI Press, Greenwich, CT, 1987, pp. 163–183.
[30] R. Lubitch and S. Moran, *Closed schedulers: A novel technique for analyzing asynchronous protocols*, Distrib. Comput., 8 (1995), pp. 203–210.
[31] N.A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, CA, 1996.
[32] Y. Moses and S. Rajsbaum, *The unified structure of consensus: A layered analysis approach*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), ACM, New York, 1998, pp. 123–132.
[33] M. Pease, R. Shostak, and L. Lamport, *Reaching agreement in the presence of faults*, J. ACM, 27 (1980), pp. 228–234.
[34] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proceedings of the 1993 ACM Symposium on Theory of Computing (STOC),

ACM, New York, 1993, pp. 101–110.

[35] N. Santoro and P. Widmayer, *Time is not a healer*, in Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 349, Springer-Verlag, New York, 1989, pp. 304–313.

[36] R. van der Meyden and Y. Moses, *Top-down considerations on distributed computing (invited talk)*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC), S. Kutten, ed., Springer-Verlag, New York, 1998, pp. 16–20.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1022–1047

ON BINARY SEARCHING WITH NONUNIFORM COSTS*

EDUARDO S. LABER†, RUY L. MILIDIÚ†, AND ARTUR A. PESSOA†

Abstract. Let us consider an ordered vector $A[1:n]$. If the cost of testing each position is similar, then the standard binary search is the best strategy to search the vector. This is true in both the average and worst case. However, if the costs are nonuniform, then the best strategy is not necessarily the standard binary search. The best algorithm to construct a strategy that minimizes the expected search cost runs in $O(n^3)$ time and requires $O(n^2)$ space. The same complexities hold for the best algorithm to construct a strategy that minimizes the worst case search cost.

Here, we show how to efficiently construct search strategies that are at most at a constant factor from the optimal one. These constructions take linear time and use only linear space. For both the problem of minimizing the expected search cost, under uniform access probabilities, and the problem of minimizing the worst case search cost, we present algorithms that require $O(n)$ space and give a $(2+\epsilon+o(1))$-approximated solution in $O(n)$ time for any fixed value of $\epsilon > 0$.

Key words. search trees, approximation algorithms, scaling

AMS subject classifications. 68W05, 68W25, 68W40

PII. S0097539700381991

1. Introduction.

Let $A = [a_1 < \cdots < a_n]$ be an ordered list of keys with each key having an associated access cost $c(a_i)$. Any search strategy for A can be represented by a binary search tree (BST) with n nodes, where each node corresponds to a key of A. Let $N(a_i, T)$ be the set of ancestors of node a_i in a BST T. The cost of searching a key a in a BST T is defined by

$$\sum_{x \in N(a,T)} c(x).$$

The average cost problem (ACP) for the input list A consists of finding a BST T^* that minimizes the expected search cost

$$EC(T) = \frac{1}{n} \sum_{i=1}^{n} \sum_{x \in N(a_i,T)} c(x) \tag{1}$$

in the set of all BSTs for A. Observe that we are assuming uniform access probabilities. We can discard the $1/n$ factor and reformulate the ACP as follows: find a BST T^* that minimizes the search cost

$$c(T) = \sum_{i=1}^{n} \sum_{x \in N(a_i,T)} c(x) \tag{2}$$

in the set of all BSTs for A.

*Received by the editors December 4, 2000; accepted for publication (in revised form) October 1, 2001; published electronically March 13, 2002. The first version of this paper appeared in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, D.C., 2001.
http://www.siam.org/journals/sicomp/31-4/38199.html

†Departamento de Informática, PUC-Rio, Rua Marques de São Vicente 225, RDC 5 andar, Brasil (laber@inf.puc-rio.br, milidiu@inf.puc-rio.br, artur@inf.puc-rio.br). The first author's research was supported by CNPQ process 300428/99-5.

On the other hand, the worst cost problem (WCP) for the input list A consists of finding the BST T^* that minimizes the worst case access cost

$$w(T) = \max_{i=1,\ldots,n} \left\{ \sum_{x \in N(a_i,T)} c(x) \right\} \tag{3}$$

in the set of all BSTs for A. Throughout this paper, we assume without loss of generality that $\sum_{i=1}^{n} c(a_i) = 1$.

Although the problem of searching a sorted array with nonuniform access costs is a basic problem, an efficient solution for it (in the sense of many applications) is not known. In [3], Knight uses dynamic programming to devise an $O(n^3)$ time algorithm that requires $O(n^2)$ space to construct a search strategy that minimizes the expected access cost, where n is the number of array positions. His motivation is to solve a filter design problem. Knight also considers the particular cost structure where the cost of accessing the ith key is $c_i = i^k$ for a fixed integer value of k. In this case, he proves that the expected cost of a standard binary search is at most at a constant factor from the expected cost of an optimal strategy.

In [7], Navarro et al. considered a more general problem, where the access cost of a position depends on the previously accessed position. Their motivation was to minimize the time spent by the disk head during a search on a text indexed by a suffix array. If the disk head is on track i, then it requires time $c(i, j)$ to go to track j. They also used dynamic programming to devise two algorithms: one to minimize the expected search cost and another to minimize the worst case search cost. Both algorithms run in $O(n^3)$ time with $O(n^2)$ space requirement. For a particular cost structure that arises on a tape search problem, Wachs [9] extended Yao quadrangle inequalities [10] to obtain an $O(n^2)$ time algorithm to construct an optimal search strategy.

As far as we know, all the algorithms proposed to construct optimal strategies for searching on a sorted array with nonuniform costs are based on dynamic programming. As a consequence, they do not give much insight on the combinatorial structure of the optimal search strategy. Moreover, these algorithms run in $\Omega(n^2)$ time, which is extremely slow for several applications. On the other hand, simpler and faster strategies, such as the standard binary search or a greedy strategy that always accesses the "cheapest" position, achieve only good results for particular cost structures. In a recent paper [5], the authors proposed an $O(n^2)$ time algorithm that constructs a strategy whose expected cost, under uniform access probabilities, is at most at a $O(\log n)$ factor from the optimal one. In this case, no particular cost structure is assumed.

1.1. Results. In this paper, our major contribution is to efficiently construct search strategies that are at most at a constant factor from the optimal one without assuming anything about the cost structure. These constructions take linear time and use only linear space.

We propose two algorithms: ACT and WCT. ACT requires $O(n)$ space and gives a $(2+\epsilon+o(1))$-approximated solution for the ACP in $O(n)$ time for any fixed value of $\epsilon > 0$. On the other hand, WCT also requires $O(n)$ space and gives a $(2 + \epsilon + o(1))$-approximated solution for the WCP in $O(n)$ time for any fixed value of $\epsilon > 0$.

1.2. Notation. Let L be a contiguous sublist of the input list A. We use the following notation throughout this paper:

$c(L)$: search cost of the BST constructed by ACT for the sublist L;
$w(L)$: worst case cost of the BST constructed by WCT for the sublist L;
ACP(L): average cost problem (ACP) for the input sublist L;
WCP(L): worst cost problem (WCP) for the input sublist L;
$c^*(L)$: search cost of an optimal solution for ACP(L);
$w^*(L)$: worst case access cost of an optimal solution for WCP(L).

A strictly BST is a BST where every internal node has exactly two children. The level of a node v in a BST T is the number of edges in the path that connects v to the root of T. Let us define the concept of ancestor in a tree. Every node is an ancestor of itself. If s is a node, then the parent of any ancestor of s is also an ancestor of s. We say that a node s_1 is a strict ancestor of a node s_2 if and only if s_1 is an ancestor of s_2 and $s_2 \neq s_1$.

1.3. Paper organization. This paper is organized as follows. In section 2, we detail the algorithms ACT and WCT. In section 3, we introduce the partitioning technique and prove lower bounds on both $c^*(L)$ and $w^*(L)$. In section 4, we analyze the ACT algorithm. In section 5, we analyze the WCT algorithm. In section 6, we describe how the two algorithms can be implemented in linear time. Finally, in section 7 we present our conclusions and some open questions.

2. The algorithms. Both ACT and WCT try to balance the following goals:
(i) test the keys with small access costs first;
(ii) produce a reasonably balanced BST.

For that, each key in the input list is ranked according to its cost. In order to construct a BST, both algorithms use a "smallest-rank-first" rule; that is, the keys with smaller ranks are always searched before those with greater ranks. This rule reflects goal (i). Goal (ii) is considered to decide which key, among the keys with equal ranks, shall be tested first.

Both algorithms use a top-down approach to construct a BST. First, they select all the keys with the smallest rank in the current input list. Furthermore, every contiguous sublist of nonselected keys is condensed into a single node. After that, both algorithms construct an auxiliary strictly BST T_D, where each internal node corresponds to a selected key and each leaf corresponds to a (possibly empty) condensed sublist. The only point where ACT and WCT differ is the way that T_D is constructed. Assume that we have already solved the subproblem associated to each sublist represented by a condensed leaf. Then, both ACT and WCT obtain the final BST by replacing each condensed leaf of T_D by the BST constructed for the associated subproblem. In fact, the construction of each BST that will replace each corresponding condensed leaf is performed by a recursive call. Later in this section, we explain the construction of T_D.

Now, we illustrate ACT with an example. Let us consider a list of keys $L = [a_1, \ldots, a_9]$ with corresponding costs [0.1, 0.1, **0.05**, **0.03**, 0.3, 0.1, 0.08, 0.2, **0.04**], where the bold printed costs correspond to the keys with the smallest rank. Figure 1 illustrates the steps performed by ACT when L is the input list. ACT splits L into seven sublists: $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$ and $L_7 = []$. Each of the sublists with even index (L_2, L_4, and L_6) contains exactly one key. These keys are associated to the internal nodes of the auxiliary BST T_D represented by Figure 1(a). On the other hand, the sublists with odd indices (L_1, L_3, L_5, and L_7) correspond to the condensed leaves of T_D. Figures 1(b) and 1(c) represent the BSTs constructed for L_1 and L_5, respectively. At the end of the execution, these BSTs replace their corresponding leaves in T_D. Moreover, since both

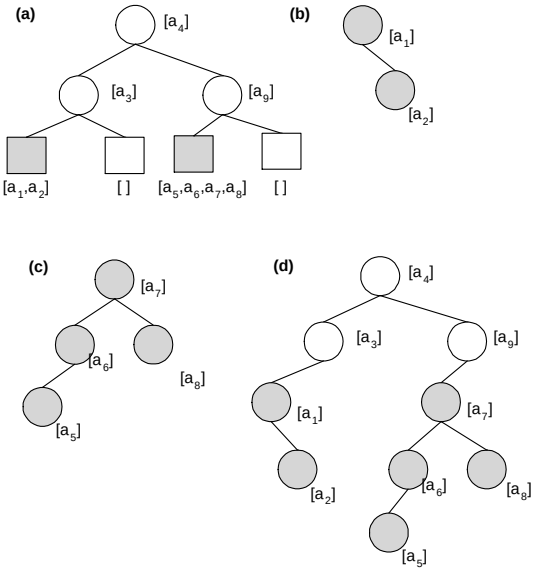


FIG. 1. (a) *The auxiliary strictly BST T_D with a_3, a_4, and a_9 corresponding to internal nodes.* (b) *The BST constructed by ACT for the subproblem ACP(L_1), where $L_1 = [a_1, a_2]$.* (c) *The BST constructed by ACT for the subproblem ACP(L_5), where $L_5 = [a_5, a_6, a_7, a_8]$.* (d) *The final BST T_L.*

the sublists L_3 and L_7 are empty, their corresponding leaves are removed from T_D. The resulting BST T_L is represented by Figure 1(d).

Now, let us formalize the notion of ranks. Let t be a positive integer number and let α be a real scaling constant greater than one. In the approximation analysis, we set the appropriate values for α and t. Assume that the real interval $[0, 1)$ is partitioned into $t + 1$ subintervals, where the first subinterval is given by $[0, 1/\alpha^t)$ and the ith interval, for $i = 2, \ldots, t+1$, is given by $[1/\alpha^{t-i+2}, 1/\alpha^{t-i+1})$. We define the rank of a key a_j as the index of the subinterval that contains $c(a_j)$. Furthermore, the rank of a sublist L is defined as the rank of the key with smallest rank in L. We use $r(L)$ to denote the rank of L.

Observe that, for the example of Figure 1, we may have $t = 4$ and $\alpha = 2$. In this case, since the first subinterval is $[0, 1/16)$, only the keys a_3, a_4, and a_9 have rank one.

A unique pseudocode for both algorithms is presented in Figure 2. The only difference between them is the construction of the auxiliary strictly BST T_D, which is performed by the CONSACT procedure for ACT and by the CONSWCT procedure for WCT. Observe that each parameter of CONSACT gives the number of keys that correspond to a node (leaf or internal node) in the auxiliary tree. On the other hand, CONSWCT receives as parameter the cost of each BST that will replace each leaf of T_D. Procedure BUILD returns a BST for L and its worst case cost $w(T_L)$. The latter returned value is not used if the problem is an ACP.

In order to improve the approximation ratio of both algorithms, BUILD constructs an optimal BST for every list L such that $|L| \leq K$ for a fixed integer parameter K. The optimal tree is constructed through the cubic time dynamic programming algorithm proposed in [3].

2.1. The CONSACT procedure. Let us recall the well-known *weighted path length problem*. Given a list of weights $[w_1, \ldots, w_{2n+1}]$, the weighted path length of a

Function BUILD(L : list of keys)

If $|L| \leq K$
  $T_L \leftarrow$ optimal BST for L constructed through dynamic programming
  return T_L and its worst case cost $w(T_L)$;
Let $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ be the keys of L with rank $r(L)$;
Let also $i_0 = 0$ and $i_{k+1} = |L| + 1$;
For $j = 1, \ldots, k$ **do**
  Create L_{2j} as the one element list $[a_{i_j}]$;
For $j = 0, \ldots, k$ **do**
  Create L_{2j+1} as the list $[a_{i_j+1}, a_{i_j+2}, \ldots, a_{i_{j+1}-1}]$;
For $j = 0, \ldots, k$ **do**
  $(T_{L_{2j+1}}, w(T_{L_{2j+1}})) \leftarrow$ BUILD(L_{2j+1}) ;
If THE PROBLEM IS ACP, **then**
  $T_D \leftarrow$ CONSACT$(|L_1|, |L_2|, \ldots, |L_{2k+1}|)$
Else
  $T_D \leftarrow$ CONSWCT$(w(T_{L_1}), w(T_{L_3}), \ldots, w(T_{L_{2k+1}}))$
For $j = 1, \ldots, k$ **do**
  Associate a_{i_j} to the jth internal node visited in an in-order traversal of T_D
For $j = 0, \ldots, k$ **do**
  Replace the $(j+1)$th leaf of T_D, from left to right, by $T_{L_{2j+1}}$;
Let T_L be the resulting tree;
Return T_L and $w(T_L)$;

FIG. 2. *Pseudocode for both ACT and WCT.*

strictly binary tree T with $2n+1$ nodes is defined by

$$\sum_{i=0}^{n} w_{2i+1} l_{2i+1} + \sum_{i=1}^{n} w_{2i}(l_{2i} + 1),$$

where l_i is the level of the ith visited node during an in-order traversal of T. Observe that the leaf levels are associated to the odd indices, while the internal nodes levels are associated to the even indices. The weighted path length problem consists of finding a strictly binary tree that minimizes the weighted path length for a given list of weights. There are several algorithms to address this problem [4, 6, 2]. CONSACT uses the linear time approximate algorithm proposed in [2] to construct a strictly binary tree T_D for the list of weights defined by the cardinality of each list L_i, that is, the list of weights $[|L_1|, |L_2|, \ldots, |L_{2k+1}|]$. In our example, CONSACT gets the parameters $[2, 1, 0, 1, 4, 1, 0]$ and constructs the tree presented in Figure 1(a).

2.2. The CONSWCT procedure. In [8], Sebö and Waksman proposed an algorithm to solve the following problem: Given a list of numbers $w = [w_1, w_2, \ldots, w_q]$, find a binary tree T, with q leaves, that minimizes

$$f(T, w) = \max_{i=1,\ldots,q} \{l_i + w_i\},$$

where l_i is the level of the ith leaf in T from left to right. Their algorithm runs in $O(q)$ time.

Let T_D be the tree that solves this problem for the list of numbers

$$[w(L_1)\alpha^{t-r(L)+1}, w(L_3)\alpha^{t-r(L)+1}, \ldots, w(L_{2k+1})\alpha^{t-r(L)+1}].$$

The procedure CONSWCT receives as parameters $w(L_1)$, $w(L_3), \ldots, w(L_{2k+1})$ and returns the tree T_D, which is constructed by Sebö's algorithm.

We must note that T_D also solves the following problem: Find a tree T that minimizes

$$w'(T) = \max_{i=0,\ldots,k} \{l_{2i+1}/\alpha^{t-r(L)+1} + w(L_{2i+1})\}, \tag{4}$$

where l_{2i+1} is the level of the ith leaf, from left to right, in T. This is true because we have multiplied only the objective function of the first problem by $1/\alpha^{t-r(L)+1}$.

3. Partitioning. Now, let us introduce a technique that provides lower bounds on both the ACP and WCP.

Let $L = [a_1, \ldots, a_m]$ be a list of keys and let T be a BST for L. Now, we consider a generic partition of L into k nonempty contiguous sublists $[a_1, \ldots, a_{i_1}]$, $[a_{i_1+1}, \ldots, a_{i_2}], \ldots, [a_{i_{k-1}+1}, \ldots, a_m]$. We use L_j to denote the jth sublist

$$[a_{i_{j-1}+1}, \ldots, a_{i_j}],$$

where $i_0 = 0$ and $i_k = m$. We show how to obtain a lower bound on the average (worst case) search cost of an optimal BST for L as a function of the average (worst case) search cost of the optimal BSTs for $L_1, \ldots, L_k$. We start with a simple lemma.

LEMMA 3.1. *For all i, with $1 \le i \le k$, there is a key in L_i that is an ancestor in T of all other keys of L_i.*

Proof. If $|L_i| = 1$, the result is obvious. Hence, we assume that $|L_i| > 1$. In this case, let a be the key of L_i with maximum number of descendants in T. If all the keys of L_i are descendants of a, we are done. Hence, we assume that there is a key b in L_i that is not a descendant of a. We have two cases. If b is an ancestor of a, we contradict the assumption that a has the maximum number of descendants among the keys of L_i. On the other hand, if b is not an ancestor of a, then let c be the lowest common ancestor of both a and b. Since T is a BST and L_i is a contiguous sublist of L, then we have that c also belongs to L_i. Hence, c has more descendants than a, which contradicts our assumption that a has the maximum number of descendants in L_i. As a result, there is a key in L_i that is an ancestor of all other keys of L_i in T. □

If a key a of L_i is an ancestor of all other keys of L_i in T, then we say that a is the *pivot* of L_i in T. Given a key $a \in L_i$, if a is not the pivot of L_i in T, let $close(a)$ be the closest strict ancestor of a in T that also belongs to L_i. Otherwise, let $close(a) = a$. We construct a tree T_i for the keys of the sublist L_i by using the following procedure.

PROCEDURE **CONSTRUCT** (T, L_i)**.**
For all $a \in L_i$, with $close(a) \neq a$, **do**
    If a is in the left subtree rooted at $close(a)$ in T, **then**
        turn a into a left child of $close(a)$ in T_i
    else turn a into a right child of $close(a)$ in T_i.

As an example, let $L = [a_1, \ldots, a_{11}]$, $L_1 = [a_1, a_2, a_3]$, $L_2 = [a_4, a_5, a_6, a_7, a_8]$, and $L_3 = [a_9, a_{10}, a_{11}]$. A BST T for L is shown in Figure 3(a). The trees T_1, T_2, and T_3 are shown in Figure 3(b). The number inside each node represents the index of the corresponding key. In addition, the nodes of L_2 are shaded in Figure 3. The pivots of the sublists L_1, L_2, and L_3 in T are a_2, a_8, and a_{10}, respectively.

We claim that, for any cost structure, the sum of the search costs of the trees T_1, T_2 and T_3 in Figure 3(b) is not greater than the search cost of T. This is true because every ancestor of a key in T_i is also an ancestor of this key in T.

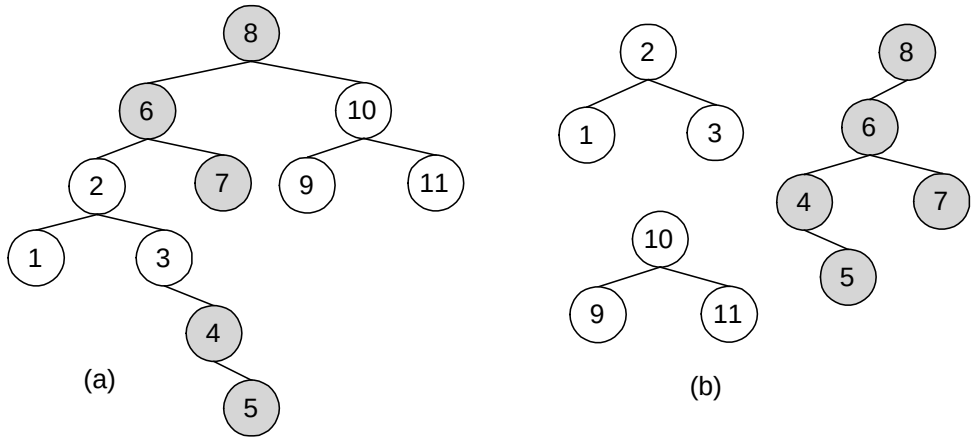


FIG. 3. (a) *BST T for the list $[a_1, \ldots, a_{11}]$.* (b) *BST's T_1, T_2 and T_3 for the lists $L_1 = [a_1, a_2, a_3]$, $L_2 = [a_4, \ldots, a_8]$ and $L_3 = [a_9, a_{10}, a_{11}]$, respectively.*

Now, let us state another simple result.

LEMMA 3.2. *Given a list L, let T be a BST for L and let L_i be a contiguous sublist of L. Furthermore, let T_i be the tree constructed by CONSTRUCT(T, L_i). Then, T_i is a binary search tree.*

Proof. The proof is in Appendix A. □

This partition technique can be used to obtain lower bounds for both the ACP and WCP. As an example, let us assume that the BST T presented in Figure 3(a) solves ACP(L). Since T_i is a feasible solution for ACP(L_i), for $i = 1, 2, 3$, then the search cost of T_i is bounded below by $c^*(L_i)$. Hence, we can conclude that

$$c^*(L) \geq c^*(L_1) + c^*(L_2) + c^*(L_3). \tag{5}$$

In the same fashion, one can conclude that

$$w^*(L) \geq \max\{w^*(L_1), w^*(L_2), w^*(L_3)\}.$$

These observations yield the following lemma.

LEMMA 3.3. *Let us assume that L is partitioned into k nonempty contiguous sublists $L_1, L_2, \ldots, L_k$. Then,*

$$c^*(L) \geq \sum_{i=1}^{k} c^*(L_i)$$

and

$$w^*(L) \geq \max_{i=1,\ldots,k} \{w^*(L_i)\}.$$

Nevertheless, this kind of lower bound is not enough to assure a constant approximation factor for both WCT and ACT. In the next sections, we show how to sharpen these lower bounds.

3.1. Lower bounds. Now, we develop specific lower bounds using the partitions suggested by both the ACT and the WCT algorithms. For that, let us assume that L is partitioned by BUILD into the sublists $L_1, L_2, \ldots, L_{2k+1}$ (lines 5–8 of the pseudocode presented in Figure 2). The reader should observe that we are not considering the subpartitioning of L due to the recursive calls to BUILD. Moreover, let us define

$$I(L) = \{i| \ |L_i| > 0 \text{ and } r(L_i) > r(L)\}$$

and

$$I'(L) = \{2, 4, \ldots, 2k\}.$$

3.1.1. The partitioning lower bound for the ACP. Here, we construct a parametric lower bound on the cost of the tree that solves ACP(L).

First, let us define a parametric problem ACP(L,d). Given a real number d, with $d \leq \min_{a\in L}\{c(a)\}$ and an input list L, the problem ACP(L, d) is to find a BST that minimizes

$$c_d(T) = \sum_{a_i \in L} \sum_{x \in N(a_i,T)} (c(x) - d)$$

in the set of all BSTs for the list L.

The key idea to improve the lower bound given by Lemma 3.3 is to decompose the cost $c(a)$ of each key $a \in L$ into two components d and $c(a) - d$ and then obtain a lower bound on $c^*(L)$ as the sum of two distinct lower bounds: a lower bound on a particular instance of ACP(L), where every key in L has the same cost d, and a lower bound on ACP(L, d). The next theorem formalizes this idea. First, we need some definitions. We use $T^*_L(d)$ to denote the tree that solves the ACP(L, d). For the sake of simplicity, we use $c^*(L, d)$, and not $c_d(T^*_L(d))$, to denote the cost of the optimal tree for ACP(L, d). When $d = 0$, we drop d, that is, $c^*(L) = c^*(L, 0)$. Now, we generalize (improve) the lower bound on $c^*(L)$ given by Lemma 3.3.

THEOREM 3.4. *Let d_1 and d_2 be two nonnegative numbers such that $d_1 + d_2 \leq \min_{a\in L}\{c(a)\}$. Then,*

$$c^*(L, d_1) \geq d_2 \times |L|(\log |L| - 1) + \sum_{i \in I(L)\cup I'(L)} c^*(L_i, d_1 + d_2).$$

Proof. Let us consider a key a that belongs to a list L_i. We use $l(a)$ to denote the level of a in $T^*_L(d_1)$ and $N_L(a)$ to denote the set of ancestors of a in $T^*_L(d_1)$. Hence, we have that

$$c^*(L, d_1) = \sum_{a\in L} \sum_{x\in N_L(a)} d_2 + \sum_{a\in L} \sum_{x\in N_L(a)} (c(x) - d_1 - d_2)$$

$$\geq \sum_{a\in L} d_2(l(a) + 1) + c^*(L, d_1 + d_2), \tag{6}$$

where the last inequality holds since $T^*_L(d_1)$ is a feasible solution for ACP($L, d_1 + d_2$).

Moreover, it follows from Lemma 3.3 that

$$c^*(L, d_1 + d_2) \geq \sum_{i \in I(L)\cup I'(L)} c^*(L_i, d_1 + d_2). \tag{7}$$

On the other hand, since $\sum_{a\in L}(l(a)+1)$ is the internal path length of a binary tree, we have that $d_2\sum_{a\in L}(l(a)+1) \geq d_2|L|(\log|L|-1)$. Hence, from (6) and (7), we can conclude that

$$c^*(L,d_1) \geq d_2|L|(\log|L|-1) + \sum_{i\in I(L)\cup I'(L)} c^*(L_i, d_1+d_2). \qquad \square$$

For example, if we apply the previous theorem for the partition of Figure 3, we obtain that

$$c^*(L,d_1) \geq 27d_2 + c^*(L_1,d_1+d_2) + c^*(L_2,d_1+d_2) + c^*(L_3,d_1+d_2).$$

In particular, for $d_1 = d_2 = 0$, we get the lower bound given by (5).

The main lower bound for the ACP, stated later in Theorem 4.1, is obtained by recursively using the previous result for every sublist of A generated by procedure BUILD. The values of d_1 and d_2 are adequately set in that theorem.

In the next theorem, we present a technical result that relates $c^*(L)$ to $c^*(L,d_1)$. This result is used only in the proof of Theorem 4.5, the constant approximation theorem.

THEOREM 3.5. *Let* $0 \leq d_1 < d_2 \leq \min_{a\in L}\{c(a)\}$. *Then*

$$\frac{c^*(L)}{c^*(L,d_1)} \leq \frac{d_2}{d_2-d_1}.$$

Proof. Let $T_L^*(d_1)$ be the tree that solves ACP(L,d_1). Also let $N_L(a)$ be the set of all ancestors of the key a in $T_L^*(d_1)$. By definition, we have that

$$c^*(L,d_1) = \sum_{a\in L}\sum_{x\in N_L(a)} (c(x)-d_1).$$

In addition, we have

$$c^*(L) \leq \sum_{a\in L}\sum_{x\in N_L(a)} c(x),$$

since $c^*(L)$ is the optimum cost of ACP(L) and $T_L^*(d_1)$ is a feasible solution for this problem. Hence,

$$\frac{c^*(L)}{c^*(L,d_1)} \leq \frac{\displaystyle\sum_{a\in L}\sum_{x\in N_L(a)} c(x)}{\displaystyle\sum_{a\in L}\sum_{x\in N_L(a)} (c(x)-d_1)} \leq \max_{a\in L}\left\{\frac{c(a)}{c(a)-d_1}\right\} = \frac{c(a^*)}{c(a^*)-d_1},$$

where $a^* = argmin_{a\in L}\{c(a)\}$.

Since $d_2 \leq c(a^*)$, it follows that

$$\frac{c^*(L)}{c^*(L,d_1)} \leq \frac{c(a^*)}{c(a^*)-d_1} \leq \frac{d_2}{d_2-d_1}. \qquad \square$$

3.1.2. The partitioning lower bound for the WCP. In this section, we improve the lower bound on $w^*(L)$ given by Lemma 3.3. For that, we take into account the level of the pivots in $T^*_W(L)$, the BST that solves WCP(L). As an example, in Figure 3, the worst case search cost of the BST T_1 corresponding to L_1 is a lower bound on the worst case search cost of the BST T for the list L. Let d be a real number with $d \leq \min_{a\in L}\{c(a)\}$. If we take into account the fact that the pivot of L_1 in T, the node with label 2, is on level 2 in T, then we can conclude that the worst case cost of T_1 added to $2d$ is also a lower bound on the worst case search cost of T. The next lemma formalizes this idea.

For $j \in I(L) \cup I'(L)$, let y_j be the pivot of the sublist L_j in the tree $T^*_W(L)$. Furthermore, let l_j^W be the level of y_j in $T^*_W(L)$. We can state the following result.

LEMMA 3.6. *Let d be a real number such that $0 \leq d \leq \min_{a\in L}\{c(a)\}$. Then,*

$$w^*(L) \geq \max_{i\in I(L)\cup I'(L)}\{d \times l_i^W + w^*(L_i)\}. \tag{8}$$

Proof. Let us consider the key a of L_i. The key a has two kinds of ancestors in $T^*_W(L)$: the ones that belong to L_i and the ones that do not belong to L_i. Since the strict ancestors of y_i in $T^*_W(L)$ do not belong to L_i, then we can conclude that the cost of searching a in $T^*_W(L)$ is at least

$$d \times l_i^W + \sum_{x\in(N_L(a)\cap L_i)} c(x),$$

where $N_L(a)$ denotes the set of all ancestors of the key a in $T^*_W(L)$. Hence, the cost of searching the key of L_i with worst case access cost in $T^*_W(L)$ is bounded below by

$$d \times l_i^W + \max_{a\in L_i}\left\{\sum_{x\in(N_L(a)\cap L_i)} c(x)\right\}. \tag{9}$$

Observe that the second term of (9) is exactly the worst case access cost of the BST T_i constructed by CONSTRUCT($T^*_W(L), L_i$). Since T_i is a feasible solution for WCP(L_i), it follows that its worst case access cost is bounded below by $w^*(L_i)$. Hence, the cost of searching the key of L_i with worst case access cost in $T^*_W(L)$ is bounded below by

$$d \times l_i^W + w^*(L_i).$$

The result is established by taking the maximum lower bound among all sublists $\{L_i | i \in I(L) \cup I'(L)\}$. ∎

Later, in the analysis of WCT, we relate d to the rank of the list L. More precisely, we set $d = 1/\alpha^{t-r(L)+2}$ when $r(L) > 1$ and $d = 0$, otherwise. The reader should observe that, for these settings, $d \leq \min_{a\in L}\{c(a)\}$.

4. Analysis of ACT. In this section, we analyze the approximation ratio of ACT. For that, we need to introduce some ideas and definitions.

Let Z be the set of all nonempty sublists of A that are passed as a parameter in some call to function BUILD. In addition, let Z_K be the set of all sublists of Z with cardinality greater than K. Recall that, for every list L, with $|L| \leq K$, BUILD constructs an optimal search tree. The main lower bound for the ACP, stated later in Theorem 4.1, is basically a function of the rank and the cardinality of the lists in Z.

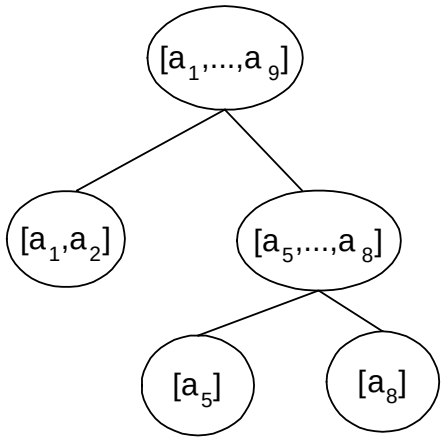


FIG. 4. *A tree T(Build).*

Similarly, the main upper bound on the search cost of the BST constructed by ACT (Theorem 4.4) is a function of the rank and the cardinality of the lists in Z.

The lists of Z can be arranged in a tree $T(Build)$ in the following way. Each node of $T(Build)$ corresponds to a sublist that belongs to Z. The root is the input list A. Given a sublist $L \in Z$, the children of L in $T(Build)$ correspond to the sublists $\{L_i | i \in I(L)\}$. Note that the sublists $L_2, L_4, \ldots, L_{2k}$ are not passed as a parameter of function BUILD, and, as a consequence, they do not appear in $T(Build)$.

Observe that Theorem 3.4 gives a lower bound on $c^*(L)$ as a function of L and its children in $T(Build)$. On the other hand, an upper bound on $c(L)$ presented later in this section (Corollary 4.3) is also a function of L and its children in $T(Build)$. As a result, by recursively evaluating these upper (lower) bounds, we can express an upper (lower) bound on $c(A)$ ($c^*(A)$) as a function of all the lists corresponding to the nodes of $T(Build)$. That is exactly the approach used in the proofs of Theorems 4.4 and 4.1.

As an example of $T(Build)$, consider the input list $A = [a_1, \ldots, a_9]$ with corresponding costs $[0.1, 0.1, 0.05, 0.03, 0.3, 0.1, 0.08, 0.2, 0.04]$. Moreover, assume that $\alpha = 2$, $t = 4$, and $K = 2$. Figure 4 shows a tree $T(Build)$ for this example. In this case, we have that $Z_2 = \{[a_1, \ldots, a_9], [a_5, \ldots, a_8]\}$ and $Z - Z_2 = \{[a_1, a_2], [a_5], [a_8]\}$.

We use $g(L)$ to denote the rank of the parent of L in $T(Build)$; that is, if L is a child of L' in $T(Build)$, then $g(L) = r(L')$. For the input list A, we define $g(A) = -\infty$. Finally, we use T_L to denote the tree constructed by ACT for the list L.

4.1. Lower bounds. In this section, we obtain a lower bound on the cost of the optimal BST for the input list A. For that, we apply the partition technique developed in section 3. We define

$$R(L) = \begin{cases} 0 \text{ if } r(L) = 1, \\ 1/\alpha^{t-r(L)+2} \text{ if } r(L) > 1. \end{cases}$$

Observe that $R(L)$ is a lower bound on the cost of the keys of L, that is, $R(L) \leq \min_{a \in L}\{c(a)\}$. If L' is the parent of L in $T(Build)$, then we define $G(L) = R(L')$. For the input list A, $G(A) = 0$.

The following theorem gives a lower bound on the cost of the tree that solves the ACP for an input list A.

THEOREM 4.1.

$$c^*(A) \geq \max\left\{1, \sum_{L\in Z_K} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L\in(Z-Z_K)} c^*(L, G(L))\right\}.$$

Proof. Basically, the proof is obtained by applying Theorem 3.4, recursively, starting with the input list A. For a complete proof, see Appendix B. □

4.2. Upper bounds. In this section, we obtain an upper bound on the cost $c(A)$ of the tree T_A constructed by the ACT algorithm for the input list A. The main upper bound stated in Theorem 4.4 is a function of both the cardinality and the rank of the lists in Z.

Let L be a list that belongs to Z. We consider two cases separately: (i) $|L| \leq K$ and (ii) $|L| > K$.

In case (i), we have that

$$c(L) \leq c^*(L), \tag{10}$$

since ACT constructs an optimal BST for L when $|L| \leq K$.

Now, we consider case (ii). The next theorem gives an upper bound on $c(L)$ as a function of the levels of nodes of the BST constructed by CONSACT.

THEOREM 4.2. *For $j \in I(L) \cup I'(L)$, let l_j be the level of the node associated to the sublist L_j in the tree T_D constructed by CONSACT for the input parameters $|L_1|, |L_2|, \ldots, |L_{2k+1}|$. Then,*

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}}\left(\sum_{j\in I(L)} |L_j| l_j + \sum_{j\in I'(L)} (l_j + 1)\right) + \sum_{j\in I(L)} c(L_j).$$

Proof. Let us consider a key a that belongs to a sublist L_j with $j \in I'(L) \cup I(L)$. The key a has two kinds of ancestors in T_L: the ones that belong to L_j and the ones that do not belong to L_j. The construction of T_D implies that every ancestor of a that does not belong to L_j is an internal node in T_D with rank equal to $r(L)$. Moreover, the number of ancestors of a that do not belong to L_j is l_j. Hence, the cost of searching a in T_L is bounded above by

$$\frac{l_j}{\alpha^{t-r(L)+1}} + \sum_{x\in(N_L(a)\cap L_j)} c(x),$$

where $N_L(a)$ denotes the set of all ancestors of the key a in T_L.

Adding the previous upper bound for all the keys of L_j, we obtain the following upper bound on the cost of searching the keys of L_j in T_L:

$$\sum_{a\in L_j}\left(\frac{l_j}{\alpha^{t-r(L)+1}} + \sum_{x\in(N_L(a)\cap L_j)} c(x)\right) = \frac{|L_j| l_j}{\alpha^{t-r(L)+1}} + c(L_j).$$

Hence, by adding the previous upper bound for all sublists L_j, with $j \in I(L) \cup I'(L)$, we obtain that

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}}\left(\sum_{j\in I(L)} |L_j| l_j + \sum_{j\in I'(L)} l_j\right) + \sum_{j\in I'(L)} c(L_j) + \sum_{j\in I(L)} c(L_j).$$

The result is established by observing that $c(L_j) \leq 1/\alpha^{t-r(L)+1}$ for $j \in I'(L)$. ∎

Now, we refine the previous result obtaining an upper bound on $c(L)$ as a function of both the cardinality and the rank of the lists that are children of L in $T(Build)$. For that, we take into account the fact that CONSACT uses the algorithm presented in [2] to construct a BST. We remark that the following result also holds if CONSACT uses the algorithm proposed in [6] instead of the one presented in [2].

COROLLARY 4.3.

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i).$$

Proof. In [2], an approximation algorithm, say DPDS, is presented to solve the weighted path length problem. In particular, Theorem 7 of [2] states that if $h_1, \ldots, h_{2n+1}$ are the levels of the nodes of the BST constructed by DPDS for the list of weights $w_1, \ldots, w_{2n+1}$, then

$$\sum_{i=0}^{n} w_{2i+1} h_{2i+1} + \sum_{i=1}^{n} w_{2i} h_{2i} \leq \left(\sum_{i=1}^{2n+1} w_i \right) \log \left(\sum_{i=1}^{2n+1} w_i \right) - \sum_{i=1}^{2n+1} w_i \log w_i$$

$$+ w_{\max} - w_1 - w_{2n+1} + \sum_{i=1}^{2n+1} w_i,$$

where $w_{\max} = \max_{i=0,\ldots,n}\{w_{2i+1}\}$. We point out that the previous inequality is stated in [2] as a function of probabilities $p_1, \ldots, p_{2n+1}$, where $p_i = w_i / \sum_{i=1}^{2n+1} w_i$.

Here, we need the weaker upper bound

$$\sum_{i=0}^{n} w_{2i+1} h_{2i+1} + \sum_{i=1}^{n} w_{2i} h_{2i} \leq \left(\sum_{i=1}^{2n+1} w_i \right) \log \left(\sum_{i=1}^{2n+1} w_i \right) - \sum_{i=1}^{2n+1} w_i \log w_i + 2 \sum_{i=1}^{2n+1} w_i.$$

Since $l_1, \ldots, l_{2k+1}$ are the levels of the nodes of the BST T_D, constructed by CONSACT for the list of weights $[|L_1|, \ldots, |L_{2k+1}|]$, it follows from the discussion above that

$$\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} (l_j + 1) \leq |L| \log |L| + 2|L| - \sum_{j \in I(L)} |L_j| \log |L_j|.$$

Hence, it follows from the previous inequality and from Theorem 4.2 that

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i). \quad ∎$$

Now, we can state the main upper bound on the cost of the BST constructed by ACT for the input list A.

THEOREM 4.4.

$$c(A) \leq \sum_{L \in Z_K} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right)$$

$$+\sum_{L\in(Z-Z_K)}\left(c^*(L)-\frac{|L|\log|L|}{\alpha^{t-g(L)+1}}\right).$$

Proof. This result can be proved by induction on the level of $T(Build)$ as we did in Theorem 4.1. For a complete proof we refer to Appendix C. □

4.3. Approximation. In this section, we prove that ACT obtains a $(2+\epsilon+o(1))$-approximated solution for the ACP. For that, we basically compare the lower bound given by Theorem 4.1 with the upper bound given by Theorem 4.4. Let us assume that $K \geq 2$.

THEOREM 4.5.

$$\frac{c(A)}{c^*(A)} \leq \frac{n\log n+2n}{\alpha^t}+\max\left\{\frac{\alpha}{\alpha-1},\frac{\alpha(\alpha-1)\log(K+1)+2\alpha^2}{(\alpha-1)(\log(K+1)-1)}\right\}.$$

Proof. If $|A| \leq K$, ACT generates the optimal BST. Hence, we assume that $|A| > K$. We divide our analysis into two cases:

(i) $r(A) = 1$;

(ii) $r(A) > 1$.

First, we analyze case (i). It follows from Theorem 4.4 that

$$c(A) \leq \frac{|A|(\log|A|+2)}{\alpha^t} \tag{11}$$

$$+\sum_{L\in(Z_K-\{A\})}\left(\frac{|L|\log|L|}{\alpha^{t-r(L)+1}}-\frac{|L|\log|L|}{\alpha^{t-g(L)+1}}+\frac{2|L|}{\alpha^{t-r(L)+1}}\right)+\sum_{L\in(Z-Z_K)}c^*(L).$$

On the other hand, Theorem 4.1 assures that

$$c^*(A)\geq\max\left\{1,\sum_{L\in Z_K}(R(L)-G(L))\,|L|(\log|L|-1)+\sum_{L\in(Z-Z_K)}c^*(L,G(L))\right\}.$$

Dividing (11) by the previous inequality, and using the facts that $1/\alpha^{t-r(L)+1} = \alpha R(L)$ and $\alpha G(L) \leq 1/\alpha^{t-g(L)+1}$, for $L \in Z_K - \{A\}$, we obtain that

$$\frac{c(A)}{c^*(A)} \leq \frac{|A|\log|A|+2|A|}{\alpha^t}$$

$$+\frac{\displaystyle\sum_{L\in(Z_K-A)}\alpha(R(L)-G(L))|L|\log|L|+2\alpha R(L)|L|+\sum_{L\in(Z-Z_K)}c^*(L)}{\displaystyle\sum_{L\in Z_K}(R(L)-G(L))\,|L|(\log|L|-1)+\sum_{L\in(Z-Z_K)}c^*(L,G(L))}.$$

Now, let

$$P=\max_{L\in(Z-Z_K)}\left\{\frac{c^*(L)}{c^*(L,G(L))}\right\}$$

and

$$Q = \max_{L \in (Z_K - A)} \left\{ \frac{\alpha(R(L) - G(L)) \log |L| + 2\alpha R(L)}{(R(L) - G(L))(\log |L| - 1)} \right\}.$$

Hence, we have that

$$\frac{c(A)}{c^*(A)} \leq \frac{|A| \log |A| + 2|A|}{\alpha^t} + \max\{P, Q\}. \tag{12}$$

Now, let us analyze P. Observe that $0 \leq G(L) < R(L) \leq \min_{a \in L}\{c(a)\}$ for all $L \in (Z - Z_K)$. Hence, it follows from Theorem 3.5 that

$$P = \max_{L \in (Z - Z_K)} \left\{ \frac{c^*(L)}{c^*(L, G(L))} \right\} \leq \max_{L \in (Z - Z_K)} \left\{ \frac{R(L)}{R(L) - G(L)} \right\}. \tag{13}$$

Since $G(L) \leq R(L)/\alpha$, then

$$P \leq \max_{L \in (Z - Z_K)} \left\{ \frac{R(L)}{R(L) - G(L)} \right\} \leq \frac{\alpha}{\alpha - 1}. \tag{14}$$

Now, let us analyze Q. Since $G(L) \leq R(L)/\alpha$ and $|L| > K$, for all $L \in (Z_K - A)$, we have that

$$Q \leq \frac{\alpha \log(K+1)}{\log(K+1) - 1} + \frac{2\alpha^2}{(\alpha - 1)(\log(K+1) - 1)}. \tag{15}$$

Let $|A| = n$. It follows from (12), (14), and (15) that

$$\frac{c(A)}{c^*(A)} \leq \frac{n \log n + 2n}{\alpha^t} + \max\left\{ \frac{\alpha}{\alpha - 1}, \frac{\alpha(\alpha - 1)\log(K+1) + 2\alpha^2}{(\alpha - 1)(\log(K+1) - 1)} \right\}. \tag{16}$$

Case (ii), where $r(A) > 1$, is similar. The only difference is that we do not need to consider the list A separately. Hence, the term $(n \log n + 2n)/\alpha^t$ does not appear in the right-hand side of (16). ∎

COROLLARY 4.6. *The ACT algorithm is* $(2 + \epsilon + o(1))$*-approximated for any* $\epsilon > 0$.

Proof. Let $\epsilon > 0$. Then, set $\alpha = 2$, $t = 2 \log n$, and $K = 2^{1+10/\epsilon} - 1$. It follows from Theorem 4.5 that

$$\frac{c(A)}{c^*(A)} \leq \frac{n \log n + 2n}{n^2} + 2 + \epsilon. \qquad \blacksquare$$

5. WCT analysis. Here, we prove a constant approximation ratio for WCT. Recall that we have defined $T^*_W(L)$ as the optimal tree for WCP(L). We define $T_{pv}(L)$ as the tree that connects the pivots of the sublists $\{L_i | i \in I'(L) \cup I(L)\}$ in $T^*_W(L)$. More precisely, given a pivot y_i, let *close*(y_i) be the closest strict ancestor of y_i in $T^*_W(L)$ that also belongs to $\{y_i | i \in I'(L) \cup I(L)\}$. $T_{pv}(L)$ is constructed by the following procedure:

For all $y \in \{y_i | i \in I'(L) \cup I(L)\}$ **do**
  If y is in the left subtree rooted at $close(y)$ in $T^*_W(L)$, **then**
    turn y into a left child of $close(y)$ in $T_{pv}(L)$
  else turn y into a right child of $close(y)$ in $T_{pv}(L)$

Figure 5(b) shows an example of a tree $T_{pv}(L)$ for the case where $L = [a_1, \ldots, a_9]$, $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$, $L_7 = []$ and $T^*_W(L)$ is the tree of Figure 5(a).

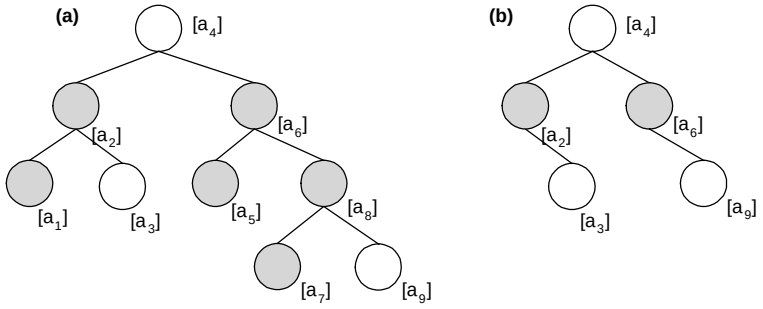


FIG. 5. (a) *An optimal tree* $T^*_W(L)$ *for* $L = [a_1, \ldots, a_9]$. (b) *The corresponding tree* $T_{pv}(L)$ *for* $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$, *and* $L_7 = []$.

PROPOSITION 5.1. *$T_{pv}(L)$ is a BST.*

Proof. We omit this proof, since it is similar to that of Lemma 3.2. □

Our strategy to prove the constant approximation ratio for WCT is to obtain a lower bound on $w^*(L)$ as a function of the levels of the nodes in $T_{pv}(L)$ and to obtain an upper bound on $w(L)$ as a function of these same levels.

Now, we state a combinatorial lemma that will be used later to relate the levels of leaves in the tree T_D constructed by CONSWCT with the levels of the nodes in $T_{pv}(L)$.

LEMMA 5.2. *For $i \in I(L) \cup I'(L)$, let l_i be the level of y_i in $T_{pv}(L)$. Then, there is a BST T' formed by the pivots $\{y_i : i \in I(L) \cup I'(L)\}$ such that*

(i) *if $i \in I(L)$, then y_i is a leaf in T', where its level is at most $2l_i + 2$;*
(ii) *if $i \in I'(L)$, then y_i is a node (leaf or internal node) in T', where its level is at most $2l_i$.*

Proof. The proof is in Appendix D. □

5.1. Lower bounds. In this section, we present lower bounds on $w^*(L)$. The main lower bound, stated in the next lemma, follows from a direct application of Lemma 3.6.

LEMMA 5.3. *Let l_i be the level of y_i in $T_{pv}(L)$. If $r(L) = 1$, then*

$$w^*(L) \geq \max_{i \in I(L) \cup I'(L)} \{w^*(L_i)\}. \tag{17}$$

On the other hand, if $r(L) > 1$, then

$$w^*(L) \geq \max_{i \in I(L) \cup I'(L)} \{l_i/\alpha^{t-r(L)+2} + w^*(L_i)\}. \tag{18}$$

Proof. Clearly, if l_i^W is the level of y_i in $T_W^*(L)$, then we have $l_i \leq l_i^W$ for all $i \in I(L) \cup I'(L)$. Hence, we establish the lower bounds given by (17) and (18) applying Lemma 3.6 with $d = 0$ and with $d = 1/(\alpha^{t-r(L)+2})$, respectively. ∎

Next, we prove a simple lower bound on $w^*(L)$ that is exclusively used in the approximation analysis for the case where $L \leq K$. The reader should observe that the lower bound given by Lemma 5.3 cannot be applied for such a list since it is not partitioned by procedure BUILD, that is, $I(L) \cup I'(L) = \emptyset$.

PROPOSITION 5.4. *Let L be a sublist of A with $r(L) > 1$. Then,*

$$w^*(L) \geq \frac{\log(|L|+1)}{\alpha^{t-r(L)+2}}.$$

Proof. Observe that $w^*(L)$ is bounded below by the cost of an optimal tree T^* for a list of $|L|$ keys with costs equal to $1/\alpha^{t-r(L)+2}$. Furthermore, observe that T^* is a balanced tree, and its worst case access cost is not smaller than $\log(|L|+1)/\alpha^{t-r(L)+2}$, which establishes the proposition. ∎

5.2. Upper bounds. In this section, we obtain upper bounds on $w(L)$. We start with a simple lemma that gives an upper bound as a function of the leaf levels of the tree constructed by CONSWCT.

LEMMA 5.5. *Let T_D be the tree constructed by CONSWCT for the input parameters $w(L_1), w(L_3), \ldots, w(L_{2k+1})$. Then,*

$$w(L) \leq w'(T_D) = \max_{i=0,\ldots,k} \left\{ l_{2i+1}^D/\alpha^{t-r(L)+1} + w(L_{2i+1}) \right\},$$

where $l_1^D, l_3^D, \ldots, l_{2k+1}^D$ are the leaf levels of T_D.

Proof. Let a be the key with maximum search cost in T_L and let $N_L(a)$ be the set of ancestors of a in T_L. Moreover, assume that $a \in L_i$, where $i \in I(L) \cup I'(L)$.

If $i \in I(L)$, then

$$w(L) = \sum_{x \in N_L(a) \cap L_i} c(x) + \sum_{x \in N_L(a) - L_i} c(x) = w(L_i) + \sum_{x \in N_L(a) - L_i} c(x),$$

where the last inequality follows from the fact that a is also the key with maximum search cost in T_{L_i}. Moreover, since the cost of every internal node of T_D is bounded above by $1/\alpha^{t-r(L)+1}$, we obtain that

$$w(L) = \; w(L_i) + \sum_{x \in N_L(a) - L_i} c(x) \leq w(L_i) + l_i^D/\alpha^{t-r(L)+1} \leq w'(T_D).$$

On the other hand, if $i \in I'(L)$, then

$$w(L) = \sum_{x \in N_L(a)} c(x) \leq \max\{l_{i-1}^D, l_{i+1}^D\}/\alpha^{t-r(L)+1} \leq w'(T_D). \qquad ∎$$

Now, we prove the main upper bound, which is a function of the node levels of the tree $T_{pv}(L)$. Roughly speaking, this upper bound is a factor of 2α larger than the lower bound given by Lemma 5.3. The key idea in the proof of the main upper bound is to show that $w'(T_D) \leq w'(T'')$, where T'' is a BST obtained by adding some particular leaves to the BST T' given by Lemma 5.2.

LEMMA 5.6. *Let l_i be the level of y_i in $T_{pv}(L)$. Then,*

$$w(L) \leq \max_{i \in I(L) \cup I'(L)} \left\{ \frac{2l_i + 2}{\alpha^{t-r(L)+1}} + w(L_i) \right\}. \tag{19}$$

Proof. For the sake of simplicity, let $\gamma(L) = 1/\alpha^{t-r(L)+1}$. Let T' be the BST given by Lemma 5.2 and let l'_i be the level of y_i in T' for $i \in I(L) \cup I'(L)$. For $i \in \{1, 3, \ldots, 2k+1\} - I(L)$, define

$$l'_i = \begin{cases} l'_2 + 1 \text{ if } i = 1, \\ \max\{l'_{i-1}, l'_{i+1}\} + 1 \text{ if } 1 < i < 2k+1, \\ l'_{2k} + 1 \text{ if } i = 2k+1. \end{cases}$$

This last definition gives the levels of the leaves corresponding to the empty sublists $\{L_i | i \in \{1, 3, \ldots, 2k+1\} - I(L)\}$ in a BST T'' obtained from T' by adding such leaves. Moreover, we have that

$$w'(T'') = \max_{i=1,3,\ldots,2k+1} \{l'_i \gamma(L) + w(L_i)\}. \tag{20}$$

Now, let i^* be the index that maximizes the right-hand side of (20). If $i^* \in I(L)$, it follows from Lemma 5.2 that $l'_{i^*} \leq 2l_{i^*} + 2$. Hence, if $i^* \in I(L)$, we have that $l'_{i^*}\gamma(L) + w(L_{i^*}) \leq (2l_{i^*} + 2)\gamma(L) + w(L_{i^*})$. On the other hand, if $i^* \in \{1, 3, \ldots, 2k+1\} - I(L)$, we have that $l'_{i^*} = l'_j + 1$ for some $j \in I'(L)$. Then, it follows from Lemma 5.2 that $l'_{i^*} \leq 2l_j + 1$, and, as a consequence, $l'_{i^*}\gamma(L) + w(L_{i^*}) = l'_{i^*}\gamma(L) \leq (2l_j + 2)\gamma(L) + w(L_j)$. This analysis allows us to conclude that

$$w'(T'') \leq \max_{i \in I(L) \cup I'(L)} \{(2l_i + 2)\gamma(L) + w(L_i)\}. \tag{21}$$

Since T'' is a feasible solution for the problem defined at the end of section 2.2 and T_D is an optimal solution for this same problem, we have that $w'(T_D) \leq w'(T'')$. Hence, it follows from inequality (21) and from Lemma 5.5 that

$$w(L) \leq w'(T_D) \leq w'(T'') \leq \max_{i \in I(L) \cup I'(L)} \{(2l_i + 2)\gamma(L) + w(L_i)\}. \qquad \square$$

5.3. Approximation. Here, we prove a constant approximation for WCT. The next lemma shows how to relate $w(A)$ and $w^*(A)$ when $r(A) = 1$, while Lemma 5.8 shows how to relate $w(L)$ and $w^*(L)$ when $r(L) > 1$. The constant approximation ratio stated later in Theorem 5.9 is obtained by recursively applying Lemmas 5.7 and 5.8.

LEMMA 5.7. *Let us assume that the input list A is partitioned by BUILD into the sublists $A_1, A_2, \ldots, A_{2k+1}$. Moreover, let us assume that $r(A) = 1$ and that $A_m = argmax\{w(A_i) | i \in \{1, 2, \ldots, 2k+1\}\}$. Then,*

$$\frac{w(A)}{w^*(A)} \leq \frac{n \log n + n}{\alpha^t} + \frac{w(A_m)}{w^*(A_m)}.$$

Proof. For the sake of simplicity, we assume that CONSWCT builds a balanced tree when $r(A) = 1$. Hence, we have that $w(A) \leq (\log n + 1)/\alpha^t + w(A_m)$. The lemma is established by using the facts that $w^*(A) \geq w^*(A_m) > 0$ and $w^*(A) \geq 1/n$. $\square$

Now, we show how to compare the upper bound given by Lemma 5.6 with the lower bounds given by Lemma 5.3 and by Proposition 5.4.

LEMMA 5.8. *Let us assume that BUILD is called with parameter L such that $r(L) > 1$ and $|L| > K$. Furthermore, let p be the index that maximizes the right-hand side of* (19). *If $|L_p| > K$, then*

$$\frac{w(L)}{w^*(L)} \leq \max\left\{2\alpha, \frac{2/\alpha^{t-r(L)+1} + w(L_p)}{w^*(L_p)}\right\}. \tag{22}$$

Otherwise, if $|L_p| \leq K$, then

$$\frac{w(L)}{w^*(L)} \leq \frac{2\alpha}{\log(K+1)} + 2\alpha. \tag{23}$$

Proof. First, let us consider that case where $|L_p| > K$. By Lemma 5.6, we have that $w(L) \leq (2l_p + 2)/\alpha^{t-r(L)+1} + w(L_p)$. Moreover, it follows from Lemma 5.3 that $w^*(L) \geq l_p/\alpha^{t-r(L)+2} + w^*(L_p)$. Dividing the upper bound on $w(L)$ by the lower bound on $w^*(L)$, we obtain that

$$\frac{w(L)}{w^*(L)} \leq \frac{2l_p/\alpha^{t-r(L)+1} + 2/\alpha^{t-r(L)+1} + w(L_p)}{l_p/\alpha^{t-r(L)+2} + w^*(L_p)},$$

which yields to (22). Next, let us consider that case where $|L_p| \leq K$. In this case, we have that $w(L) \leq (2l_p + 2)/\alpha^{t-r(L)+1} + w^*(L_p)$, since BUILD constructs an optimal BST for L_p. Hence, we have that

$$\frac{w(L)}{w^*(L)} \leq \frac{2/\alpha^{t-r(L)+1}}{w^*(L)} + \frac{2l_p/\alpha^{t-r(L)+1} + w^*(L_p)}{l_p/\alpha^{t-r(L)+2} + w^*(L_p)} \leq \frac{2/\alpha^{t-r(L)+1}}{w^*(L)} + 2\alpha. \tag{24}$$

Nevertheless, it follows from Proposition 5.4 that $w^*(L) \geq \log(K+1)/\alpha^{t-r(L)+2}$. The result is established replacing this lower bound in the right-hand side of (24). ∎

The following theorem provides an approximation factor for WCT.

THEOREM 5.9.

$$\frac{w(A)}{w^*(A)} \leq \frac{n\log n + n}{\alpha^t} + \frac{2\alpha^2}{(\alpha-1)\log(K+1)} + 2\alpha.$$

Proof. If $|A| \leq K$, then $w(A)/w^*(A) = 1$. Hence, we assume that $|A| > K$. An upper bound on the ratio $w(A)/w^*(A)$ can be constructively obtained as follows.

Step 1. Obtain an initial upper bound on the ratio $w(A)/w^*(A)$ by applying either Lemma 5.7 or Lemma 5.8, depending whether or not $r(A) = 1$.

Step 2. While the current upper bound depends on the ratio $w(L)/w^*(L)$ for a sublist L of A, replace the ratio $w(L)/w^*(L)$ by the upper bound on this ratio given by Lemma 5.8.

First, let us assume that $r(A) = 1$. It follows from Lemma 5.7 that

$$\frac{w(A)}{w^*(A)} \leq \frac{n\log n + n}{\alpha^t} + \frac{w(A_m)}{w^*(A_m)}.$$

Let $L^0 = A_m$. Given L^i, define L^{i+1} as the sublist of L^i that is used by Lemma 5.8 to provide an upper bound on $w(L^i)/w^*(L^i)$. Let g be the largest integer such that $|L^g| > K$. By applying Step 2 g times, we obtain that

$$\frac{w(A)}{w^*(A)} \leq \frac{n\log n + n}{\alpha^t} + \sum_{i=1}^{g} \frac{2/\alpha^{t-r(L^{i-1})+1}}{w^*(L^i)} + \frac{w(L^g)}{w^*(L^g)}. \tag{25}$$

Moreover, we have that

$$\sum_{i=1}^{g}\left(\frac{2}{\alpha^{t-r(L^{i-1})+1}}\right) \le \frac{2}{\alpha^{t-r(L^{g-1})+1}}\sum_{i=0}^{\infty}\left(\frac{1}{\alpha^{i}}\right) \le \frac{2/\alpha^{t-r(L^{g})+1}}{\alpha-1} \tag{26}$$

and that

$$\frac{\log(K+1)}{\alpha^{t-r(L^{g})+2}} \le w^*(L^g) \le w^*(L^i) \tag{27}$$

for $i = 0, \ldots, g-1$. By combining (25), (26), and (27), we obtain that

$$\frac{w(A)}{w^*(A)} \le \frac{n\log n + n}{\alpha^t} + \frac{2\alpha}{\log(K+1)(\alpha-1)} + \frac{w(L^g)}{w^*(L^g)}.$$

Since $|L^{g+1}| \le K$, it follows from Lemma 5.8 that

$$\frac{w(A)}{w^*(A)} \le \frac{n\log n + n}{\alpha^t} + \frac{2\alpha^2}{\log(K+1)(\alpha-1)} + 2\alpha.$$

If $r(A) > 1$, then the term $(n\log n + n)/\alpha^t$ is removed from the approximation ratio. □

COROLLARY 5.10. *The WCT algorithm is* $(2+\epsilon+o(1))$*-approximated for any* $\epsilon > 0$.

Proof. For any $\epsilon > 0$, we can set $\alpha = 1+\epsilon/4$, $t = \lceil 2\log_\alpha n\rceil$ and $K = 2^{16\alpha^2/\epsilon^2} - 1$. In this case, it follows from Theorem 5.9 that

$$\frac{w(A)}{w^*(A)} \le \frac{n\log n + n}{n^2} + \epsilon/2 + (2+\epsilon/2). \qquad \square$$

6. Implementation. In this section, we describe how to implement both algorithms in $O(n)$ time. For that, we initially set $\alpha = 2$ and $t = \lceil \log n^2\rceil$.

Let $r(a_i)$ be the rank of the key a_i. We can calculate the ranks of all keys in $O(n)$ time by constructing a table T with $2n$ entries, where the value of the jth entry $T[j]$ is equal to $\lceil\log(j+1)\rceil + 1$ for $j = 0, \ldots, 2n-1$. Observe that this table can be computed in $O(n)$ time because the expression $\lceil\log(j+1)\rceil + 1$ has only $O(\log n)$ distinct values. In order to calculate the rank of a_i, we check if $c(a_i) < 2^{\lceil\log n\rceil - t}$. In the affirmative case, the rank of a_i is $T[\lfloor c(a_i)\times 2^t\rfloor]$. Otherwise, if $c(a_i) < 1$, then the rank of a_i is given by $T[\lfloor c(a_i)\times 2^{t-\lceil\log n\rceil}\rfloor] + \lceil\log n\rceil$. Finally, if $c(a_i) = 1$, then we set $r(a_i) = 1$. In Appendix E, we show how to compute the ranks for $\alpha = 2^{1/p}$, where p is a given integer larger than 1.

Let us assume that BUILD is called with the input parameter L. If $|L| \le K$, then the optimal tree is constructed through a dynamic programming algorithm, which takes $O(|L|^3)$ time. Now, we assume that $|L| > K$. If BUILD traverses the list L to obtain the keys with rank $r(L)$, then it spends $O(|L|)$ time. Let $k(L)$ be the number of keys in L with rank $r(L)$. The other operations performed by BUILD, including the call to CONSACT (CONSWCT), spend $O(k(L))$. Hence, the time complexity of both algorithms can be calculated by adding the time effort due to every call to BUILD. We get the following time complexity:

$$O\left(\sum_{L\in Z_K} k(L) + \sum_{L\in Z_K}|L| + \sum_{L\in(Z-Z_K)}|L|^3\right),$$

where both Z and Z_K are defined at the beginning of section 4.

Since two different lists in $(Z - Z_K)$ do not overlap, then

$$\sum_{L \in (Z - Z_K)} |L|^3 = O(nK^2),$$

which is linear in n for a fixed K. Moreover,

$$\sum_{L \in Z_K} k(L) = O(n).$$

Unfortunately, for some bad instances, $\sum_{L \in Z_K} |L|$ is $\Theta(n \log n)$. Hence, a direct implementation takes $O(n \log n)$ time.

In order to reduce the time complexity, we shall avoid traversing the entire list L to find the keys with rank $r(L)$. If we find these keys in $O(k(L))$ time, then the overall time complexity is reduced to $O(n)$. For that, we do some preprocessing.

6.1. Preprocessing. First, we spend $O(n)$ time to create $t + 1$ doubly linked lists $R_1, \ldots, R_{t+1}$. The list R_i contains the keys of A with rank i ordered by their indices.

Afterwards, we solve the *range minimum query* (RMQ) problem for the rank vector $[r(a_1), r(a_2), \ldots, r(a_n)]$. Given a vector of numbers $[x_1, x_2, \ldots, x_n]$ and $i < j$, the RMQ problem is to find the index of a minimum element in the range $[x_i, x_{i+1}, \ldots, x_j]$. Bender and Farach-Colton showed that every query can be responded in $O(1)$ time after preprocessing the input vector $[x_1, x_2, \ldots, x_n]$ in linear time [1]. Therefore, for every list L, one can obtain the keys of L with rank $r(L)$ in $O(k(L))$ time as follows:

1. Find a key with minimum rank in L in $O(1)$ time using the Bender's algorithm.
2. Traverse the $k(L)$ keys of L with rank $r(L)$ using the double linked list $R_{r(L)}$.

7. Conclusions. In this paper, we proposed two linear time algorithms with constant approximation ratios for binary searching with nonuniform costs. We have considered both the worst case and the average case problems. For the average case, we have assumed uniform probabilities. It would be interesting to devise a fast algorithm with a good approximation ratio for the general case where the probabilities are also nonuniform. Another interesting question is to determine whether there exists a unique approximate algorithm for both criteria: the worst case cost and the average cost.

Appendix A. Proof of Lemma 3.2. We can prove by induction on the size of L_i that T_i is a BST rooted by the pivot of L_i. If $|L_i| = 1$, the result holds since a unique node is a BST. Let us assume that the result holds for every sublist with less than k keys and let us consider the case where $|L_i| = k$. Let a be the pivot of L_i and let L_i^1 and L_i^2 be the sublist of L containing the keys smaller than a and the sublist of L containing the keys greater than a, respectively. Let T_i^1 (T_i^2) be the tree constructed by CONSTRUCT(T, L_i^1) (CONSTRUCT(T, L_i^2)). By induction, both T_i^1 and T_i^2 are BSTs rooted by the pivots of L_i^1 and L_i^2, respectively. Moreover, only the pivot of L_i^1 (L_i^2) is a left (right) child of a in T_i because for all other nodes of L_i^1 (L_i^2), the pivot of L_i^1 (L_i^2) is closer than a in T. As a result T_i is a BST. ∎

Appendix B. Proof of Theorem 4.1. Since $\sum_{i=1}^{n} c_i = 1$, then $c^*(A)$ cannot be smaller than 1.

We define Z^i and Z^i_K, respectively, by

$$Z^i = \{L | L \in Z \text{ and } L \text{ is at level at most } i \text{ in } T(Build)\}$$

and

$$Z^i_K = \{L | L \in Z^i \text{ and } |L| > K\}.$$

Now, let l be a nonnegative integer number. In order to establish the theorem, we prove that

$$c^*(A) \geq \sum_{L \in Z^l_K} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L \in (Z^{l+1} - Z^l_K)} c^*(L, G(L)). \tag{28}$$

Observe that (28) implies on the correctness of the theorem, since if we set l equal to the height of $T(Build)$, then we obtain that $Z^l_K = Z_K$ and $Z^{l+1} = Z$.

Now, we prove the correctness of (28) by induction on l. The basis is $l = 0$. If $|A| \leq K$, then the result follows from the fact that $Z^1 = \{A\}$, $Z^0_K = \emptyset$, and $G(A) = 0$. Hence, we assume that $|A| > K$. First, we observe that $Z^0_K = \{A\}$ and that $Z^1 - Z^0_K = \{A_i | i \in I(A)\}$. Hence, we must prove that

$$c^*(A) \geq R(A) \times |A|(\log|A| - 1) + \sum_{i \in I(A)} c^*(A_i, G(A_i)).$$

Since $G(A) = 0$, it follows that $R(A) + G(A) \leq \min_{a \in A}\{c(a)\}$. By applying Theorem 3.4 for the input list A with $d_1 = G(A) = 0$ and $d_2 = R(A)$, we obtain that

$$\begin{aligned} c^*(A) &\geq R(A)|A|(\log|A| - 1) + \sum_{i \in I'(A)} c^*(A_i, R(A)) + \sum_{i \in I(A)} c^*(A_i, R(A)) \\ &\geq R(A)|A|(\log|A| - 1) + \sum_{i \in I(A)} c^*(A_i, G(A_i)). \end{aligned}$$

Hence, (28) holds for $l = 0$. Now, we assume that the result holds for $l = p$ and we prove that it also holds for $l = p + 1$. Applying the inductive hypothesis for $l = p$, we obtain that

$$c^*(A) \geq \sum_{L \in Z^p_K} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L \in (Z^{p+1} - Z^p_K)} c^*(L, G(L)).$$

Since the set $Z^{p+1} - Z^p_K$ can be rewritten as the union of two disjoint sets $Z^{p+1} - Z^{p+1}_K$ and $Z^{p+1}_K - Z^p_K$, it follows that

$$\begin{aligned} c^*(A) \geq \sum_{L \in Z^p_K} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L \in (Z^{p+1} - Z^{p+1}_K)} c^*(L, G(L)) \\ + \sum_{L \in (Z^{p+1}_K - Z^p_K)} c^*(L, G(L)). \end{aligned} \tag{29}$$

Let $L \in (Z^{p+1}_K - Z^p_K)$. Since $R(L) \leq \min_{a_i \in L}\{c(a_i)\}$, it follows that $G(L) + R(L) - G(L) \leq \min_{a_i \in L}\{c(a_i)\}$. By applying Theorem 3.4 for the input list L, with

$d_1 = G(L)$ and $d_2 = R(L) - G(L)$, we obtain that

$$\begin{aligned} c^*(L, G(L)) &\geq (R(L) - G(L))|L|(\log|L| - 1) + \sum_{i \in I'(L)} c^*(L_i, R(L)) + \sum_{i \in I(L)} c^*(L_i, R(L)) \\ &\geq (R(L) - G(L))|L|(\log|L| - 1) + \sum_{i \in I(L)} c^*(L_i, G(L_i)). \end{aligned}$$

By adding the previous inequality for all sublists that belong to $(Z_K^{p+1} - Z_K^p)$, we obtain that

$$\begin{aligned} \sum_{L \in (Z_K^{p+1} - Z_K^p)} c^*(L, G(L)) \geq & \sum_{L \in (Z_K^{p+1} - Z_K^p)} (R(L) - G(L))|L|(\log|L| - 1) \\ & + \sum_{L \in (Z^{p+2} - Z^{p+1})} c^*(L, G(L)). \end{aligned} \tag{30}$$

Replacing the rightmost operand of (29) by the right-hand side of (30) and using the fact that the two sets $Z^{p+2} - Z^{p+1}$ and $Z^{p+1} - Z_K^{p+1}$ form a partition for $Z^{p+2} - Z_K^{p+1}$, we obtain that

$$c^*(A) \geq \sum_{L \in Z_K^{p+1}} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L \in (Z^{p+2} - Z_K^{p+1})} c^*(L, G(L)),$$

which completes the induction, and, as a consequence, establishes the result. ∎

Appendix C. Proof of Theorem 4.4. This result can be proved by induction on the level of $T(Build)$ as we did in Theorem 4.1.

In fact, let l be an integer; we must prove that

$$\begin{aligned} c(A) \leq & \sum_{L \in Z_K^l} \left(\frac{|L|\log|L|}{\alpha^{t-r(L)+1}} - \frac{|L|\log|L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) \\ & + \sum_{L \in (Z^{l+1} - Z_K^l)} \left(c(L) - \frac{|L|\log|L|}{\alpha^{t-g(L)+1}} \right), \end{aligned} \tag{31}$$

where Z^l and Z_K^l are the sets defined at the beginning of the proof of Theorem 4.1. Observe that (31) implies the correctness of the theorem, since if we set l equal to the height of $T(Build)$, we obtain that $Z_K^l = Z_K$ and $Z^{l+1} = Z$. Furthermore, for every L, with $L \in Z - Z_K$, we have $c(L) = c^*(L)$.

Now, we prove the correctness of (31) by induction on l. The basis is $l = 0$. If $|A| \leq K$, then the result follows from the fact that $Z^1 = \{A\}$, $Z_K^0 = \emptyset$ and $g(A) = -\infty$. Hence, we assume that $|A| > K$. First, we observe that $Z_K^0 = \{A\}$ and that $Z^1 - Z_K^0 = \{A_i | i \in I(A)\}$. Hence, we must prove that

$$c(A) \leq \frac{1}{\alpha^{t-r(A)+1}} \left(|A|\log|A| + 2|A| - \sum_{i \in I(A)} |A_i|\log|A_i| \right) + \sum_{i \in I(A)} c(A_i).$$

This result follows by Corollary 4.3 when $L = A$.

Hence, (31) holds for $l = 0$. Now, we assume that the result holds for $l = p$, and we prove that it also holds for $l = p + 1$. Applying the inductive hypothesis for $l = p$, we obtain that

$$c(A) \leq \sum_{L \in Z_K^p} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z^{p+1} - Z_K^p)} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right). \tag{32}$$

Since the set $Z^{p+1} - Z_K^p$ can be rewritten as the union of two disjoint sets $Z^{p+1} - Z_K^{p+1}$ and $Z_K^{p+1} - Z_K^p$, it follows that

$$c(A) \leq \sum_{L \in Z_K^p} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z^{p+1} - Z_K^{p+1})} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right) + \sum_{L \in (Z_K^{p+1} - Z_K^p)} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right). \tag{33}$$

Let $L \in (Z_K^{p+1} - Z_K^p)$. Applying Corollary 4.3 we obtain that

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i).$$

By adding the previous inequality for all sublists that belong to $(Z_K^{p+1} - Z_K^p)$, we obtain that

$$\sum_{L \in (Z_K^{p+1} - Z_K^p)} c(L) \leq \sum_{L \in (Z_K^{p+1} - Z_K^p)} \frac{|L| \log |L| + 2|L|}{\alpha^{t-r(L)+1}} - \sum_{L \in (Z^{p+2} - Z^{p+1})} \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \sum_{L \in (Z^{p+2} - Z^{p+1})} c(L). \tag{34}$$

Replacing the rightmost operand of (33) by the right-hand side of (34) and using the fact that $(Z^{p+2} - Z^{p+1})$ and $(Z^{p+1} - Z_K^{p+1})$ form a partition of $Z^{p+2} - Z_K^{p+1}$, we conclude that the result also holds for $l = p + 1$, which completes the induction, and, as a consequence, establishes the result. □

Appendix D. Proof of Lemma 5.2. The proof is by induction on the number of nodes of $T_{pv}(L)$. Clearly, the result holds if $|T_{pv}(L)| = 1$. Now, assume that the result holds for all trees $T_{pv}(L)$ with $|T_{pv}(L)| < m$. We shall prove that the result also holds for all trees $T_{pv}(L)$ with $|T_{pv}(L)| = m$. Let y_r be the root of $T_{pv}(L)$. Then, we have two cases:

(a) $r \in I'(L)$;
(b) $r \in I(L)$.

Let T^ℓ and T^r be the left and the right subtrees of y_r in $T_{pv}(L)$, respectively. Moreover, let $\hat{T}^\ell$ and $\hat{T}^r$ be the two trees obtained by using the inductive hypothesis on T^ℓ and T^r, respectively. We observe that the level of y_i, for $i \neq r$, in T^ℓ (T^r) is $l_i - 1$.

In case (a), the tree T' is constructed as follows: y_r becomes the root of T'; $\hat{T}^\ell$ and $\hat{T}^r$ become, respectively, the left and the right subtrees of y_r. By construction, the level of y_i in T', for all $i \in I(L)$, is at most $2(l_i - 1) + 2 + 1 < 2l_i + 2$, which satisfies condition (i). Moreover, the level of y_i in T', for all $i \in (I'(L) - \{r\})$, is at most $2(l_i - 1) + 1 < 2l_i$, which satisfies condition (ii).

In case (b), let us initially assume that $2 < r < 2k$. We use two artificial nodes, say y and y'. The tree T' is constructed as follows: y becomes the root of T'; y' becomes the left child of y and y_r becomes the right child of y'; $\hat{T}^\ell$ becomes the left subtree of y' and $\hat{T}^r$ becomes the right subtree of y. Figure 6 illustrates this construction. The trees $T_{pv}(L)$ and T' for this case are represented by Figures 6(a) and 6(b), respectively. We observe that the node y_r is a leaf at level 2 in T'. Moreover, if y_i is a node of T', with $i \in I(L) - \{r\}$, then its level is at most $2(l_i - 1) + 2 + 2 = 2l_i + 2$. Furthermore, if y_i is a node of T', with $i \in I'(L)$, then its level is at most $2(l_i - 1) + 2 = 2l_i$. Hence, the conditions concerning the levels are respected. Now, we must choose two convenient pivots of $T_{pv}(L)$ to replace both the artificial nodes y and y'. The nodes y' and y are respectively replaced by the rightmost node y_{r-1} of $\hat{T}^\ell$ and by the leftmost node y_{r+1} of $\hat{T}^r$. Since both $r - 1$ and $r + 1$ belong to $I'(L)$ and the levels of y_{r-1} and y_{r+1} do not increase due to the replacement, then both conditions (i) and (ii) still hold.

If we have $r = 1$ ($r = 2k+1$), then the construction of T' is analogous. We remark that the artificial node y' (y) is not used in this case because T^ℓ (T^r) is empty. □

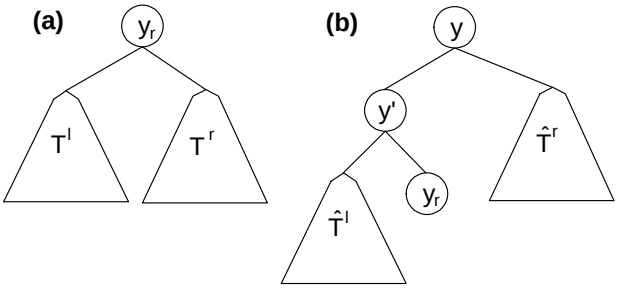


FIG. 6. (a) *A tree $T_{pv}(L)$ of pivots, where $r \in I(L)$.* (b) *The tree T' obtained from $T_{pv}(L)$, satisfying the conditions of Lemma* 5.2.

Appendix E. Calculating the ranks for fractional α. Now, let p be a fixed positive integer greater than 1. We show how to compute the key ranks for $\alpha = 2^{1/p}$ and $t = p\lceil \log n^2 \rceil + p - 1$ in $O(n)$ time. This can be useful for the WCT algorithm since better approximations are attained making α close to 1.

Let $r'(a_i)$ be the rank of a_i when the scaling parameter is equal to 2 and the total number of ranges is $t' = \lceil \log n^2 \rceil$. These auxiliary ranks can be computed in $O(n)$ time as described in the implementation section. Now, consider the following proposition.

PROPOSITION E.1. *For every $a_i \in A$, we have that $r'(a_i) = \lceil \frac{r(a_i)}{p} \rceil$.*

Proof. We have three cases:

(i) $r(a_i) = 1$;

(ii) $2 \leq r(a_i) \leq p-1$;

(iii) $r(a_i) \geq p$.

For case (iii), it is easy to check that $c(a_i) \in [1/\alpha^{t-r(a_i)+2}, 1/\alpha^{t-r(a_i)+1}) \subset [1/2^{t'-r'(a_i)+2}, 1/2^{t'-r'(a_i)+1})$. The proofs for the remaining two cases are similar. □

By the previous proposition, if the value of $r'(a_i)$ is given, then a_i has only p possible ranks: $pr'(a_i) - p + 1, pr'(a_i) - p + 2, \ldots, pr'(a_i)$. In order to compute $r(a_i)$, we construct two additional tables U and U' at the preprocessing step. We set $U[l] = \alpha^l$ for $l = 0, \ldots, p-1$, and $U'[j] = 1/2^{t'-j+1}$, for $j = 1, \ldots, t'$. Let l be the greatest index such that $c(a_i) < U'[r'(a_i)]/U[l]$. Then, $r(a_i)$ is given by $pr'(a_i) - l$, which can be computed in $O(1)$ time for a fixed value of p.

REFERENCES

[1] M. BENDER AND M. FARACH-COLTON, *The lca problem revisited*, in Proceedings of LATIN 2000, Lecture Notes in Comput. Sci. 1776, G. Gonnet, D. Panario, and A. Viola, eds., Springer-Verlag, Berlin, 2000, pp. 88–94.

[2] R. DE PRISCO AND A. DE SANTIS, *On binary search trees*, Inform. Process. Lett., 45 (1993), pp. 249–253.

[3] W. J. KNIGHT, *Search in an ordered array having variable probe cost*, SIAM J. Comput., 17 (1988), pp. 1203–1214.

[4] D. E. KNUTH, *Optimum binary search trees*, Acta Informat., 1 (1971), pp. 14–25.

[5] E. S. LABER, R. L. MILIDIU, AND A. A. PESSOA, *Strategies for searching with different access costs*, in Proceedings of ESA'99, Lecture Notes in Comput. Sci. 1645, Springer-Verlag, Berlin, 1999, pp. 236–247.

[6] K. MEHLHORN, *Nearly optimal binary search trees*, Acta Informat., 5 (1975), pp. 287–295.

[7] G. NAVARRO, E. BARBOSA, R. BAEZA-YATES, W. CUNTO, AND N. ZIVIANI, *Binary searching with non-uniform costs and its application to text retrieval*, Algorithmica, 27 (2000), pp. 145–169.

[8] A. SEBÖ AND Z. WAKSMAN, *Optimal binary trees with order constraints*, Discrete Appl. Math., 91 (1999), pp. 305–311.

[9] M. WACHS, *On an efficient dynamic programming technique of F. F. Yao*, J. Algorithms, 10 (1989), pp. 518–530.

[10] F. F. YAO, *Efficient dynamic programming using quadrangle inequalities*, in Proceedings of the 12th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1980, pp. 429–435.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1048–1075

THE EFFICIENCY OF RESOLUTION AND DAVIS–PUTNAM PROCEDURES*

PAUL BEAME†, RICHARD KARP‡, TONIANN PITASSI§, AND MICHAEL SAKS¶

Abstract. We consider several problems related to the use of resolution-based methods for determining whether a given boolean formula in conjunctive normal form is satisfiable. First, building on the work of Clegg, Edmonds, and Impagliazzo in [*Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Philadelphia, PA, 1996, ACM, New York, 1996, pp. 174–183], we give an algorithm for unsatisfiability that when given an unsatisfiable formula of F finds a resolution proof of F. The runtime of our algorithm is subexponential in the size of the shortest resolution proof of F. Next, we investigate a class of backtrack search algorithms for producing resolution refutations of unsatisfiability, commonly known as Davis–Putnam procedures, and provide the first asymptotically tight average-case complexity analysis for their behavior on random formulas. In particular, for a simple algorithm in this class, called *ordered DLL*, we prove that the running time of the algorithm on a randomly generated k-CNF formula with n variables and m clauses is $2^{\Theta(n(n/m)^{1/(k-2)})}$ with probability $1 - o(1)$. Finally, we give new lower bounds on $\mathrm{res}(F)$, the size of the smallest resolution refutation of F, for a class of formulas representing the pigeonhole principle and for randomly generated formulas. For random formulas, Chvátal and Szemerédi [*J. ACM*, 35 (1988), pp. 759–768] had shown that random 3-CNF formulas with a linear number of clauses require exponential size resolution proofs, and Fu [*On the Complexity of Proof Systems*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1995] extended their results to k-CNF formulas. These proofs apply only when the number of clauses is $\Omega(n \log n)$. We show that a lower bound of the form 2^{n^γ} holds with high probability even when the number of clauses is $n^{(k+2)/4-\epsilon}$.

Key words. proof complexity, resolution, Davis–Putnam procedure, satisfiability, random formulas, search algorithms, lower bounds

AMS subject classifications. 03F20, 68Q25, 68T15, 68T20, 03B35, 68Q17, 68W40

PII. S0097539700369156

1. Introduction.

The satisfiability problem for boolean formulas in conjunctive normal form (*CNF formulas*) plays a central role in computer science. Historically, it was the "first" NP-complete problem. It is the natural setting in which to formulate a wide variety of constraint satisfaction problems. Its companion problem, finding a proof of unsatisfiability of a given unsatisfiable formula, plays an important role in artificial intelligence, where it is referred to as *propositional theorem proving*, and also in circuit testing.

*Received by the editors March 9, 2000; accepted for publication (in revised form) October 18, 2001; published electronically March 13, 2002. Preliminary versions of these results appeared in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science [BP96] and the 30th ACM Symposium on Theory of Computing [BKPS98].

http://www.siam.org/journals/sicomp/31-4/36915.html

†Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195 (beame@cs.washington.edu). This author's research supported by NSF grants CCR-9303017 and CCR-9800124.

‡Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 (karp@cs.berkeley.edu). This research was done while the author was at the University of Washington.

§Computer Science Department, University of Toronto, Toronto, ON, Canada (toni@cs.toronto.edu). This author's research was supported by NSF grant CCR-9457782 and US-Israel BSF grant 95-00238 and was done while the author was at the University of Arizona.

¶Department of Mathematics, Rutgers University, New Brunswick, NJ 08901 (saks@math.rutgers.edu). This author's research was supported by NSF grant CCR-9700239 and was done while the author was on sabbatical at the University of Washington.

In the last three decades, a tremendous amount of research has been directed towards understanding the mathematical structure of the satisfiability problem and developing algorithms for satisfiability testing and propositional theorem proving. Much of this research has centered around the method of *resolution*. The *resolution principle* says that if C and D are clauses and x is a variable, then any assignment that satisfies both of the clauses $C \vee x$ and $D \vee \neg\, x$ also satisfies $C \vee D$. The clause $C \vee D$ is said to be a *resolvent* of the clauses $C \vee x$ and $D \vee \neg\, x$ *on the variable* x. A *resolution refutation* for a CNF formula F consists of a sequence of clauses $C_1, C_2, \ldots, C_s$ where (i) each clause C_i is either a clause of F, or is a resolvent of two previous clauses, and (ii) C_s is the empty clause, denoted by Λ. We can represent the proof as an acyclic directed graph on vertices $C_1, \ldots, C_s$ where each clause of F has in-degree 0, and any other clause has in-degree 2 with its two in-arcs from the two clauses that produced it. It is well known that resolution is a *sound* and *complete* propositional proof system; i.e., a formula F is unsatisfiable if and only if there is a resolution refutation for F. Resolution is the most widely studied approach to propositional theorem proving, and there is a large body of research exploring *resolution algorithms*, i.e., algorithms that on input an unsatisfiable formula F, output a resolution refutation of F.

Any resolution algorithm can be used trivially to test satisfiability of an arbitrary (satisfiable or unsatisfiable) formula F, since F is satisfiable if and only if the algorithm finds no refutation. Nearly all satisfiability testers that have been studied in the literature can be derived in this way from resolution algorithms, and we say that such satisfiability testers are *resolution-based*.

One fundamental approach to satisfiability testing is to use backtrack search to look for a satisfying assignment. Algorithms that use this approach are commonly called Davis–Putnam procedures, but we will refer to them as *DLL algorithms* after Davis, Logemann, and Loveland, who first considered them [DLL62]. A DLL algorithm can be described recursively as follows. First check whether F is *trivially satisfiable* (has no clauses) or is *trivially unsatisfiable* (contains an empty clause) and if so, stop. Otherwise, select a literal l_i (a variable or the complement of a variable) and apply the search algorithm recursively to search for a satisfying assignment for the formula $F\lceil_{l_i=0}$ obtained by setting $l_i = 0$ in F. If the search succeeds, then we have an assignment for F. Otherwise, repeat the search with the formula $F\lceil_{l_i=1}$. If neither of these searches finds a satisfying assignment, then F is not satisfiable.

A particular DLL algorithm is specified by a *splitting rule*, which is a subroutine that for each recursively constructed formula determines the next splitter (literal to recurse on) and the assignment to try first. In general, the splitting rule may depend on the details of the structure of the original formula and on the results of the computation in other recursive calls. For a particular formula F, different splitting rules may result in vastly different running times.

If the splitter for some given formula F is a literal l such that l is contained in a *unit clause* (a clause C of size one), then the $l = 0$ branch falsifies C and thus terminates immediately. Effectively, the algorithm fixes $l = 1$. A splitting rule is said to use *unit propagation* if, for any formula F that has a *unit clause* (clause of size one), the splitter is chosen to be a literal in such a clause. Virtually all splitting rules considered in the literature use unit propagation, and it can be shown that any splitting rule can be modified so that it uses unit propagation at the cost of a factor of at most $O(n)$ in the running time of the algorithm, where n is the number of underlying variables. We will consider only algorithms that use unit propagation.

The simplest such splitting rule is as follows: fix an ordering of the variables $x_1, \ldots, x_n$. For a subformula F' obtained by fixing some variables, if there is a unit

clause, the splitter is the first literal belonging to such a clause. Otherwise, select the first unfixed variable. The algorithm obtained from this splitting rule is called *ordered DLL*.

The execution of a DLL algorithm A on formula F can be represented by a labeled rooted binary tree, denoted $T_A(F)$, in the usual way. Each node corresponds to a recursive call. Each internal node is labeled by its splitter, and the two out-edges correspond to the possible assignments. For any node, the path from the root to that node defines a partial assignment (*restriction*) of the variables, and the recursive call at that node is applied to the subformula obtained by applying the restriction to the original formula. Each leaf is either a *success leaf*, i.e., all of the original clauses are satisfied by the associated restriction, or a *failure leaf*, i.e., at least one original clause is falsified by the restriction. Each failure leaf is labeled by one of the original clauses that if falsifies. F is unsatisfiable if and only if all leaves are failure leaves, in which case the tree as labeled above (with internal nodes labeled by splitters and leaves labeled by falsified clauses) is called a *DLL refutation* of F. The size of the refutation is defined to be the number of nodes of the tree. It is easy to see that a formula F has a DLL refutation if and only if it is unsatisfiable. Thus DLL refutations form a complete and soundproof system.

Given a DLL refutation, it is not hard to show by induction that if we start from the clauses labeling the leaves, and work towards the root, we can label each internal node by a clause which is a resolvent of the clauses labeling its two children, and the root will be labeled by the empty clause. This tree is now the directed graph representation of a resolution refutation, and thus the DLL proof system can be viewed naturally as a restricted version of resolution.

This paper focuses on some problems concerning resolution refutations, DLL refutations, and algorithms for satisfiability. Of central importance are two parameters defined for any unsatisfiable formula F:

(i) $\mathrm{res}(F)$, the size of the smallest resolution refutation of F,
(ii) $\mathrm{DLL}(F)$, the size of the smallest DLL refutation of F.

We define $\mathrm{res}(F) = \mathrm{DLL}(F) = \infty$ for satisfiable formulas. It follows from the above discussion that $\mathrm{DLL}(F) \geq \mathrm{res}(F)$ for all formulas F. Furthermore, for any DLL-procedure for satisfiability, $\mathrm{DLL}(F)$ is a lower bound for its running time on F.

Our results fall into three groups. The first group of results shows that the resolution and DLL proof systems are, to some extent, *automatizable* in the sense that for each of these systems, there is an algorithm that on input an unsatisfiable formula F, finds a refutation of F within the proof system in time that can be upper bounded nontrivially in terms of the size of the optimal refutation within that system. In particular, we show that for any formula F in n variables and m clauses that has a DLL refutation of size at most S, there is an algorithm running in $n^{O(\log S)}m$ time that *finds* a DLL refutation of F. The analogous algorithm for resolution uses $2^{O(\sqrt{n \log S} \log n)}m$ time to find a resolution refutation for a formula F possessing one of size S. Clegg, Edmonds, and Impagliazzo [CEI96] have given algorithms to determine unsatisfiability within these time bounds, but they do not produce resolution or DLL refutations.

The second group of results concerns lower bounds for general resolution proofs. We develop simpler methods for obtaining resolution lower bounds than previously used. We illustrate this by first showing a very simple lower bound on the resolution proof complexity of the pigeonhole principle which also significantly improves on the best previous lower bounds for this problem. Our main object of study for

resolution, however, is that of randomly chosen k-CNF formulas, and we use our technique to obtain much stronger lower bounds on the resolution proof complexity of such formulas.

Our third group of results analyzes the complexity of proving unsatisfiability for random formulas (above the threshold) using DLL algorithms, the algorithms that are used most commonly in practice for satisfiability testing. We obtain the first nontrivial upper bound for resolution proofs of unsatisfiability of random formulas by showing that one of the simplest of all DLL algorithms, ordered DLL, has a running time that is qualitatively similar to the size of the best possible resolution proofs. There is still a gap between our upper bounds for DLL and the lower bounds for resolution, but we show that our analysis for ordered DLL is tight. We also make progress towards showing that our upper bound is tight for all DLL algorithms by extending our lower bound for ordered DLL to a broader class of DLL algorithms.

Our techniques result in significant simplifications and improvements of previous algorithms and lower bounds. A preliminary version of this work [BP96] pointed out that further simplification could be obtained by finding a direct relationship between $\mathrm{res}(F)$ and the minimum b for which F has a proof with all clauses at most b. Recently, Ben-Sasson and Wigderson [BSW99] have developed such a characterization of $\mathrm{res}(F)$ and $\mathrm{DLL}(F)$. Using this characterization, one can derive some of our general resolution bounds more simply. We conclude our paper with a discussion of this improvement and other directions for further research.

Because our bounds for random formulas for both DLL and general resolution are our most significant results, but are necessarily spread over several sections of the paper, we discuss them now in more detail.

Random k-CNF formulas. We consider the usual random k-CNF model, which is defined in terms of three integer parameters n, k, and m. A formula is generated by selecting m clauses of size k independently and uniformly from the set of all clauses of size k on n variables. We denote this distribution by $\mathcal{F}_m^{k,n}$ and write $F \sim \mathcal{F}_m^{k,n}$ to mean that F is selected from this distribution. The ratio $\Delta = m/n$ is referred to as the *clause density*.

The random k-CNF model has been widely studied for several good reasons. First, it is an intrinsically natural model, analogous to the random graph model, that sheds light on fundamental structural properties of the satisfiability problem. Second, for appropriate choice of parameters, randomly chosen formulas are empirically difficult for satisfiability and are a commonly used benchmark for testing satisfiability algorithms. (See, for example, the encyclopedic survey of the SAT problem in [GPFW97].) Last, it is a useful model for evaluating the effectiveness of a particular propositional proof system: strong lower bounds on proof size for random k-CNF formulas attest to the fact that the proof system in question is ineffective on average.

A fundamental conjecture about the random k-CNF formula model (see [CS88, BFU93, CF90, CR92, FS96, KKKS98]) says that there is a constant θ_k, the *satisfiability threshold*, such that a random k-CNF formula of clause density Δ is almost certainly satisfiable for Δ bounded below θ_k (as n gets large) and almost certainly unsatisfiable if Δ is bounded above θ_k. There is considerable empirical and analytic evidence for this. Recently, Friedgut [Fri99] showed that for each n and k there is a threshold $\theta_k(n)$ such that for any $\epsilon > 0$, random k-CNF formulas with clause density $\Delta \leq \theta_k(n) - \epsilon$ are almost certainly satisfiable and those with clause density $\Delta \geq \theta_k(n) + \epsilon$ are almost certainly unsatisfiable. However, he does not rule out the possibility that $\theta_k(n)$ fails to converge to a constant. It is known that $\theta_2 = 1$ is

independent of n [CR92, Goe96] and that for each k, $\theta_k(n)$ is bounded between two constants, b_k and d_k, that are independent of n; e.g., $3.26 \leq \theta_3(n) \leq 4.596$ [AS00, JSV00].

The threshold indicates three distinct ranges of clause density for investigating complexity. For Δ at the threshold, an effective algorithm must be able to distinguish between unsatisfiable and satisfiable instances. Below the threshold, a random formula is almost certainly satisfiable, and the problem of interest is to find a satisfying assignment quickly.

Above the threshold, the formula is almost certainly unsatisfiable, and we have the two closely related questions, (i) What is the typical size of the smallest unsatisfiability proof? and (ii) How quickly can an algorithm find a proof?

Several empirical studies of DLL procedures on random k-CNF formulas have been done, e.g., by Selman, Mitchell, and Levesque [SML96] and Crawford and Auton [CA96]. The former applies ordered DLL (defined earlier) to random k-CNF formulas for various values of Δ. The curves in [SML96, CA96] show very low complexity for Δ below the threshold, a precipitous increase in complexity at the threshold, and a speedy decline to low complexity above the threshold.

Much has been made of the analogy with statistical physics [KS94], and there has been a suggestion that the computational complexity at the threshold is evidence of a critical phenomenon in complex systems and based on underlying edge-of-chaos behavior present only near the threshold. The empirical observation that satisfiability is easy below the threshold is supported by analytical work. The proofs of the aforementioned lower bounds on θ_k were obtained by analyzing some DLL algorithm and showing that it almost certainly finds a satisfying assignment in linear time, provided that Δ is below some specified constant.

In their seminal paper, Chvátal and Szemerédi [CS88] showed that for any fixed Δ above the threshold there is a constant $\kappa_\Delta > 0$ such that $\text{res}(F) \geq 2^{\kappa_\Delta n}$ almost certainly if F is a random k-CNF formula of clause density Δ. (This result substantially improved the previous work of Franco and Paull [FP83] which showed subexponential time lower bounds for refuting the same class of formulas using particular DLL algorithms.) On the other hand, Fu [Fu95] showed that $\text{res}(F)$ is almost certainly polynomial in n for $m = \Omega(n^{k-1})$. These results together with the empirical work motivate the problem of determining the best constant κ_Δ for which $\text{res}(F) \geq 2^{\kappa_\Delta n}$ with prob $1 - o(1)$ for random k-CNF formulas F of density Δ. The lower bound in [CS88] as presented does not give bounds on the dependence of κ_Δ on Δ, but rough estimates show that for 3-CNF formulas the bound decreases as $1/\Delta^{\Omega(\Delta^4)}$. This implies that the lower bound declines extremely quickly and becomes trivial when the number of clauses grows above $n \log^{1/4} n$. For larger clause size k Fu [Fu95] obtained better bounds but with a similar exponential drop-off. Is there really such a sharp decline in complexity for random formulas above the threshold?

Our new lower bounds show that the drop-off in κ_Δ is at most polynomial in Δ. We show that for any constant $\epsilon > 0$ there is a constant $a_\epsilon > 0$ such that for random 3-CNF formulas the complexity of resolution proofs is almost certainly at least $2^{a_\epsilon(n/\Delta^{4+\epsilon})}$ and obtain similar results for random k-CNF formulas having larger values of k. In particular, our results imply that even random formulas with moderately large clause densities require proofs of weakly exponential size. More precisely, for any $k \geq 3$ and $\epsilon > 0$, we show that there is a $\gamma > 0$ such that almost all k-CNF formulas in n variables with at most $n^{(k+2)/4-\epsilon}$ clauses require resolution

refutations of size at least 2^{n^γ}. For example, a random 3-CNF formula with $n^{5/4-\epsilon}$ clauses requires exponentially large resolution proofs.

Although these resolution bounds show that the decline in κ_Δ is at most inverse polynomial in Δ, it is not immediately clear that even a polynomial decline with Δ is achievable. There seem to be no previously known nontrivial upper bounds on the running time of algorithms on random instances above the threshold. We prove the first such nontrivial upper bound by showing that for $F \sim \mathcal{F}_m^{k,n}$, the size of the DLL refutation of F produced by ordered DLL is $2^{O(n/\Delta^{1/(k-2)})}n^{O(1)}$. Thus κ_Δ does indeed decline as a fixed power of Δ.

At the upper end, our result shows that when $m = \Omega(n^{k-1}/\log^{k-2} n)$, the algorithm runs in polynomial time, improving on Fu's $\Omega(n^{k-1})$ bound on the number of clauses needed for polynomial-size resolution proofs but also giving an algorithm to find such proofs.

There is a gap between the exponents on Δ given by our lower bounds for general resolution and our upper bound for ordered DLL. What is the optimal exponent for general resolution or for DLL, which is more interesting since such bounds would have implications for practical satisfiability testing? We show that our upper bound for ordered DLL is indeed tight in that ordered DLL requires proofs of $2^{\Omega(n/\Delta^{1/(k-2)})}$ size.

Can a different splitting rule achieve a better exponent than ordered DLL on random formulas? We expand our understanding of DLL algorithms by showing that in the case $k = 3$, for $m = \Omega(n^{3/2}\log^\epsilon n)$, the lower bound for ordered DLL extends to a larger class of algorithms, whose splitting rule (aside from unit propagation) is independent of the formula. The key step in proving this lower bound is Lemma 6.7, which applies to *any* DLL splitting rule and therefore may be of independent interest. It shows that with high probability, along every path of a DLL tree that is "not too long," the number of unit clauses generated cannot be very large. While Lemma 6.7 is general, the remainder of the proof of the lower bound unfortunately depends heavily on the independence of the splitting rule from the formula.

Overall, our results show that for $F \sim \mathcal{F}_{\Delta n}^{k,n}$, $\log_2 \mathrm{res}(F)$ decays as a fixed power of Δ, suggesting that there is not an isolated point of complexity at the threshold but rather a slow and gradual decline in complexity as Δ increases.

2. Preliminary definitions. Let $X = \{x_1, \ldots, x_n\}$ be a set of boolean variables. Following usual parlance, an assignment ρ of 0-1 values to some subset of the variables is called a *restriction*. We will abuse notation and identify ρ with the set of literals set to 1 by ρ. We write $v(\rho)$ for the set of variables that are assigned values by ρ.

Similarly, a clause C over the variables X can be viewed as a set of literals, and we write $v(C)$ for the underlying set of variables. If $\mathcal{C}$ is a set of clauses, or F is a CNF formula, we write $v(\mathcal{C})$ or $v(F)$ for the underlying sets of variables.

If C is a clause and ρ is a restriction, then ρ satisfies C if it sets some literal of C to 1. If ρ does not satisfy C, we define $C\lceil_\rho$, the *restriction of* C *by* ρ, to be the clause obtained from C by deleting all literals set to 0 by ρ. For a formula F the *restriction of* F *by* ρ, $F\lceil_\rho$, is the formula obtained by removing all clauses satisfied by ρ and replacing any other clause C of F by $C\lceil_\rho$.

If $\mathcal{P} = (C_1, \ldots, C_r)$ is a resolution refutation of a formula F and ρ is a restriction we can construct a refutation of $F\lceil_\rho$, denoted $\mathcal{P}\lceil_\rho = (D_1, \ldots, D_r)$, as follows. For ease of description, we allow the proof to contain clauses that are identically 1. For $i \in [r]$ having defined $D_1, \ldots, D_{i-1}$, if ρ satisfies C_i, then let D_i be the identically 1 clause. Otherwise, (i) if C_i is a clause of F, let $D_i = C_i\lceil_\rho$, and (ii) if C_i is the resolvent of C_j

and C_k on variable x, then if either D_j or D_k is 1, then let D_i equal the other one, and otherwise let D_i be the resolvent of D_j and D_k on x. It is not hard to show that after deleting the 1-clauses, the result is a resolution proof of $F\lceil_\rho$.

A resolution refutation $\mathcal{P}$ is said to be *b-bounded* if all of the clauses appearing in it have size at most b.

For the purpose of generating test formulas, the most natural model of a random k-CNF formula on n variables with clause density Δ is to choose $m = \Delta n$ clauses independently with replacement. This distribution, which we denote $\mathcal{F}_m^{k,n}$, is the one analyzed in [CS88]. Another model, which is used in [Fri99], is to choose each of the possible clauses independently with probability $p = m/\binom{n}{k}2^k$; call this $\mathcal{F}^{k,n}(p)$. An easy argument shows that when considering properties of formulas that are monotone (or antimonotone) with respect to sets of clauses, the almost certain properties under both distributions are the same up to a change from m to $m \pm o(m)$. This is just a natural extension of the similar (and more precise) equivalences for the random graph model as shown, for example, in [AV79]. We generally assume the distribution $\mathcal{F}_m^{k,n}$. We write $F \sim \mathcal{F}$ to mean F is a random formula selected according to distribution $\mathcal{F}$.

We make frequent use of two well-known tail bounds for the binomial distribution (see [ASE92, Appendix A]).

PROPOSITION 2.1. *If Y is a random variable distributed according to the binomial distribution $B(n,p)$, then*

1. $Pr[Y < np/4] \leq 2^{-(np)/2}$,
2. $Pr[Y > Cnp] \leq (\frac{e}{C})^{-Cnp}$.

3. Automatizability of DLL and resolution. The quantity $\mathrm{res}(F)$ (resp., $\mathrm{DLL}(F)$) tells us the size of the smallest resolution refutation (resp., DLL refutation) of F. A fundamental problem is to find effective algorithms for constructing resolution refutations and DLL refutations whose size is "close" to optimal. This is the *automatizibility problem* for proof systems, which was formalized in [BPR97].

DEFINITION 3.1. *Let $\mathcal{S}$ be an arbitrary propositional proof system.*[1] *For the unsatisfiable formula F, let $s(F)$ denote the size of the smallest refutation of F in $\mathcal{S}$. Then $\mathcal{S}$ is said to be automatizable if there exists a deterministic algorithm that takes as input an unsatisfiable formula F on n variables and m clauses, and outputs an $\mathcal{S}$-refutation of f in time polynomial in $s(F)$ and n and m. More generally $\mathcal{S}$ is $q(s,n,m)$-automatizable if there exists a deterministic algorithm that runs in time $q(s(F),n,m)$ and outputs an $\mathcal{S}$-refutation of F (whose size is necessarily also bounded by $q(s(F),n,m)$).*

THEOREM 3.2.

1. *The DLL proof system is $q(s,n,m)$-automatizable for $q(s,n,m) = n^{O(\log s)}m$.*
2. *The resolution proof system is $q(s,n,m)$-automatizable for $q(s,n,m) = 2^{O(\sqrt{n \log s}\log n)}m$.*

These results, especially the second, fall short of the desired polynomial automatizability. Nevertheless, even the second is strong enough that if $\mathrm{res}(F)$ is subexponential, $2^{o(n)}$, then our algorithm finds a subexponential size resolution refutation in subexponential time. The results are closely related to, and motivated by, previous results of Clegg, Edmonds, and Impagliazzo. Their results concern the polynomial calculus proof system (called Groebner in [CEI96]), which is more general than the

[1] We will not provide a general definition of propositional proof system, since we are focusing exclusively on the two concrete systems, resolution and DLL, that we have defined above. The interested reader can readily formulate a general definition or consult [CR77].

resolution system. In their paper, they proposed and analyzed satisfiability testing algorithms based on the Groebner basis algorithm from commutative algebra. When run on an unsatisfiable formula F, their algorithm produces a refutation in the polynomial calculus proof system (but not necessarily a resolution refutation). They give two algorithms for this, the first of which finds a refutation in time bounded above by $O((\mathrm{DLL}(F))^{\log n})$, and the second finds a refutation in time bounded above by $O(2^{\sqrt{n\mathrm{res}(F)}\log n})$. In other words, provided that F has a short DLL refutation (resp., resolution refutation), their first (resp., second) algorithm finds a refutation that is "not too big" but in the stronger polynomial calculus proof system. Our two algorithms, which closely parallel theirs, achieve comparable running times, but produce, respectively, a DLL refutation and a resolution refutation.

Proof of Theorem 3.2. The theorem asserts the existence of two algorithms, which on input an unsatisfiable formula F, find, respectively, a DLL refutation and a resolution refutation within a specified time bound. The two algorithms are most easily described together.

First, we need a subroutine, called **Bounded-search**, which takes as input F and an integer parameter b and finds a b-bounded resolution refutation of F if one exists. It is not hard to implement this subroutine in time $T_0(n, m, b) = n^{O(b)+O(m)poly(n)}$, e.g., by listing the b-bounded clauses of F and, for each clause on the list, resolve it with each clause preceding it (if possible) and add the resolvent to the end of the list, if it is of size at most b and does not duplicate anything on the list. If the algorithm constructs the empty clause, we have the desired refutation; otherwise, there is no such refutation.

The main algorithm called **Resolution-search** also takes as input F and an auxiliary parameter b. First we use **Bounded-search**(F, b) to find a b-bounded resolution refutation for F if it exists. If not, then for each of the $2v(F)$ literals l, apply **Resolution-search** to the formula $F\lceil_{l=1}$ in order to identify the literal l for which **Resolution-search**$(F\lceil_{l=1})$ terminates fastest. These $2n$ calls to **Resolution-search** are executed in a sequence of parallel rounds; in round i the ith step of each of the $2n$ calls is performed. As soon as the first of the calls terminates, say for literal l^*, all of the other calls are aborted, except the call corresponding to $\neg\, l^*$, which is run to completion. The output of **Resolution-search**(F, b) consists of the derivation of the singleton clause l^*, followed by the derivation of the singleton clause $\neg\, l^*$ followed by $\emptyset$. (Note that the parallel search for the literal l^* described above can be replaced by a more space efficient "doubling search" which in iteration i runs each of the recursive calls one by one from the beginning for 2^i steps, stopping the first time that one of the calls terminates. The time analysis below can be modified to apply to this variant.)

The analysis of the algorithm will rely on the following technical fact, which is easily proved by induction.

PROPOSITION 3.3. *Suppose that $T(n, s)$ is a function defined for nonnegative integers n and $s > 0$ that satisfies, for some positive increasing function $h(n)$, positive constant C and $\lambda > 1$:*

$$\begin{array}{lcll} T(0,s) & \leq & h(0), & \\ T(n,s) & \leq & h(n) & \text{if } s \leq 1, \\ T(n,s) & \leq & h(n) + CnT(n-1, \frac{s}{\lambda}) + T(n-1,s) & \text{if } n \geq 1 \text{ and } s > 1. \end{array}$$

Then $T(n, s) \leq h(n)(1 + C^{\log_\lambda s} n^{2\log_\lambda s})$.

We now prove the first part of the automatization theorem. Here we use the above algorithm with $b = 0$. It is not hard to see that in this case, the output by **Resolution-**

search can be viewed as a DLL refutation, since the DAG associated to the proof is a tree. We upper bound the running time of the algorithm in terms of DLL(F). Let $T_1(n, s; m)$ denote the maximum running time of **Resolution-search**$(F, 0)$ over all formulas F with at most n variables and m clauses and for which DLL(F) $\leq s$. Consider a DLL refutation of size at most s and let x_i be the splitting variable at the root. The left and right branches of the tree give refutations for $F\lceil_{x_i}$ and $F\lceil_{\neg x_i}$, and the smaller of these is of size at most $s/2$. Hence at least one of the recursive calls terminates after at most $T_1(n-1, s/2; m)$ steps, and so the literal l^* is found after at most that number of rounds. The time for each round can be bounded above by Cn for some constant C. Once l^* is found, it takes at most $T_1(n-1, s; m)$ steps to complete the call to **Resolution-search**$(F\lceil_{\neg\, l^*})$. Thus, we conclude that for fixed m, $T_1(n, s; m)$ satisfies the recurrence of the above proposition with $h(n) = T_0(n, m, b)$ and $\lambda = 2$. We conclude that $T_1(n, s; m) = n^{O(\log s)}O(m)$ as required to prove the first automatization result.

For the second result, first define, for a set $\mathcal{P}$ of clauses, $\mathcal{P}[b]$ to be the subset of clauses of size greater than b. For a formula F, let res(F, b) denote the minimum of $|\mathcal{P}[b]|$ over all resolution refutations of F (so that for $b < 0$, res(F) = res(F, b)). Let $T_2(n, s; m, b)$ denote the maximum time needed by **Resolution-search**(F, b) on (n, m)-formulas F satisfying res$(F, b) \leq s$. Note that $T_2(n, s; m, b) \leq T_0(n, b)$ if $s < 1$ and $T_2(0, s; m, b) = O(1)$. Suppose n and s are both at least 1. Let F be an (n, m)-formula and let $\mathcal{P}$ be a resolution refutation of F with $|\mathcal{P}[b]| \leq s$. For a literal l, let $c(b, l)$ be the number of clauses of $\mathcal{P}[b]$ containing l. The average of $c(b, l)$ over literals is greater than $|\mathcal{P}[b]|b/2n$, and hence there exists a literal l with $c(b, l) > b|\mathcal{P}[b]|/2n$. Note that the refutation $\mathcal{P}\lceil_{l=1}$ of $F\lceil_{l=1}$ has at most $|\mathcal{P}[b]|(1 - \frac{b}{2n})$ clauses, and hence $T_2(n, s; m, b)$ satisfies the recurrence for T in the proposition with $\lambda = \frac{2n}{2n-b}$ and $h(n) = T_0(n, b)$. Applying the proposition, we conclude that $T_2(n, b) \leq T_0(n, b)n^{O(\frac{n}{b}\log s)}$. Choosing $b = \sqrt{n \log s}$ yields an upper bound of $2^{O(\sqrt{n \log s}\log n)}O(m)$ to complete the proof of the theorem. □

Remark. The result of Ben-Sasson and Wigderson mentioned in the introduction implies that the **Bounded-search** routine is sufficient to automatize resolution. More specifically, they show that for any formula F, if DLL(F) $\leq s$, then **Bounded-search** with $b = O(\log s)$ finds a resolution refutation of F, and if res$(F) \leq s$, then **Bounded-search** with $b = O(\sqrt{n \log s})$ finds a resolution refutation of F.

4. Lower bounding resolution proof complexity. For any unsatisfiable formula on n variables, res$(F) \leq$ DLL$(F) \leq 2^n + 1$, since a DLL proof of an n-variable formula is a binary tree of maximum depth n. Unless $coNP = NP$ one would expect that there are formulas where res(F) is superpolynomial in $|F|$, but it is not obvious how to prove such lower bounds. In a breakthrough paper, Haken [Hak85] proved the first exponential lower bounds in general resolution for a class of formulas related to the pigeonhole principle. Haken obtained his bounds using an elegant new technique called "bottleneck counting." The technique was developed further in [Urq87] to give more general bounds on resolution refutations. Building on Haken's and Urquhart's arguments, Chvátal and Szemerédi [CS88] used the bottleneck counting method to show that for a sufficiently large constant c, almost certainly a random 3-CNF formula with cn clauses requires an exponential length resolution refutation. Fu [Fu95] recently extended this bound to apply when the number of clauses is larger, but for 3-CNF formulas it gives no improvement on [CS88].

All of these proofs have the same general structure. To prove a lower bound on res(F) for all F belonging to some specified class of formulas $\mathcal{F}$, the first step

is to prove a result of the following type: for some larger class $\mathcal{F}'$ of formulas, any resolution refutation of $G \in \mathcal{F}'$ must have a "large" clause.

In the second step, which is typically the more involved part, an arbitrary resolution refutation of F is considered. Each clause in the proof is viewed as allowing certain truth assignments to flow through it, namely those that it falsifies. Using the result of the first step, one shows that every truth assignment must flow through some "complex" or "large" clause that permits only a small number of truth assignments to pass (thus the term "bottleneck counting"). Therefore, the number of clauses in the refutation must be large. The complications in the argument come in making the association between complex clauses and truth assignments.

Our method uses something very much like the first step but replaces the second step by an argument that leads to stronger results with simpler arguments. We assume for contradiction that $F \in \mathcal{F}$ has a small proof. We use this assumption to show that F can be modified to a formula $F' \in \mathcal{F}'$ that has a proof with no large clauses, contradicting the first step.

We consider two methods for modifying the formula F to get F'. The first is to apply a restriction, i.e., fix a small set of variables. The second is to *augment* F, i.e., add some additional clauses to F. The restriction method is equivalent to the special case of the augmentation method where the clauses that are added are all unit clauses.

4.1. Lower bounds for the pigeonhole principle. We illustrate our approach to lower bounds for general resolution proofs by giving a very simple proof of the exponential lower bounds for the class $\{\neg\ PHP_n^m : m > n\}$ of pigeonhole principle formulas considered by Haken. The variables of the formula $\neg\ PHP_n^m$ correspond to the entries $P_{i,j}$ of an $m \times n$ boolean matrix. (We think of the rows as corresponding to"pigeons" and the columns as corresponding to "holes".) Its clauses are (1) $P_{i,1} \vee P_{i,2} \vee \cdots \vee P_{i,n}$ for each $i \leq m$ (each row has at least one 1, or every pigeon goes into a hole) and (2) $\neg P_{i,k} \vee \neg P_{j,k}$ for each $i, j \leq m$, $k \leq n$, $i \neq j$ (each column has at most one 1, or every hole gets at most one pigeon). Since $m > n$, this is trivially unsatisfiable. Note that the number of clauses in $\neg\ PHP_n^m$ is $m + \binom{m}{2} n \leq m^3$. We prove the following theorem.

THEOREM 4.1. *For $n \geq 2$, any resolution refutation of $\neg PHP_{n-1}^n$ has size at least $2^{n/20}$.*

Proof. As in the lower bound proof of Haken [Hak85], a truth assignment to the underlying variables $P_{i,j}$ is *critical* if it defines a one-to-one, onto map from $n-1$ rows (pigeons) to $n-1$ columns (pigeonholes), with the remaining pigeon not mapped to any hole. A critical assignment where i is the pigeon left out is called *i-critical*. In what follows we will be interested only in critical truth assignments.

Let C be a clause. The monotone clause $M(C)$ associated to C is obtained by replacing each occurrence of a negative literal $\neg\ P_{i,k}$ by the set of literals $\{P_{l,k} \mid l \neq i\}$. It is easy to check that C and $M(C)$ are satisfied by precisely the same set of critical assignments.

We will be interested in restrictions corresponding to partial matchings that one obtains by repeatedly choosing an i, j, and setting $P_{i,j} = 1$, $P_{i,j'} = 0$ for $j' \neq j$, and $P_{i',j} = 0$ for $i' \neq i$. Observe that if one begins with a resolution refutation of $\neg PHP_{n-1}^n$ and one chooses such a partial matching restriction that sets t variables to 1, then the result is resolution refutation of $\neg PHP_{n-t-1}^{n-t}$. Furthermore, the restriction applied to the monotone conversion of each clause results in the same clause as doing the monotone conversion of the clause first and then applying the restriction.

(The transformation to monotone clauses, due to Buss, is not essential, but it will make our argument slightly cleaner.)

So let $C_1, \ldots, C_S$ be a resolution refutation of $\neg PHP^n_{n-1}$ and $M_1 = M(C_1), \ldots, M_S = M(C_S)$ be its monotone conversion. Say that a clause M_t is *large* if it has at least $n^2/10$ (positive) literals, i.e., at least one-tenth of all the variables. To show that $S \geq 2^{n/20}$, we will show that the number L of large clauses is at least $2^{n/20}$. Assume for contradiction that $L < 2^{n/20}$. Let $d_{i,j}$ denote the number of large clauses containing $P_{i,j}$. By averaging, there is an i, j with $d_{i,j} \geq L/10$. Choose such an i, j, and apply the restriction $P_{i,j} = 1$, $P_{i,j'} = 0$ for $j' \neq j$, and $P_{i',j} = 0$ for $i' \neq i$. Applying this restriction we obtain a monotone conversion of a refutation of $\neg PHP^{n-1}_{n-2}$ with at most $9L/10$ large clauses. Applying this argument iteratively $\log_{10/9} L$ many times, we are guaranteed to have knocked out all large clauses. Thus, we are left with a refutation of $\neg PHP^{n'}_{n'-1}$, where

$$n' \geq n - \log_{10/9} L = (1 - (\log_{10/9} 2)/20)n > 0.671n$$

and where no clause in the refutation is large. However, this contradicts the following lemma (originally due to Haken [Hak85]) which states that such a refutation must have a clause whose monotone conversion has size at least $2(n')^2/9 > n^2/10$. □

LEMMA 4.2. *Any resolution refutation of $\neg PHP^n_{n-1}$ must contain a clause C such that $M(C)$ has at least $2n^2/9$ literals.*

Proof. Given a clause C, let

$$\mathit{badpigeons}(C) = \{i \mid \text{there is some } i\text{-critical assignment } \alpha \text{ falsifying } C\}.$$

Define the complexity $\mathrm{comp}(C) = |\mathit{badpigeons}(C)|$.

Let $\mathcal{P}$ be a resolution refutation of $\neg PHP^n_{n-1}$ and consider the complexity of the clauses that appear in $\mathcal{P}$. The complexity of each initial clause is at most 1, and the complexity of the final false clause is n.

Note that if we use the resolution rule to derive a clause C from two previous clauses C' and C'', we have that $\mathrm{comp}(C) \leq \mathrm{comp}(C') + \mathrm{comp}(C'')$, since any assignment falsifying C must also falsify at least one of C' or C''. If C is the first clause in the proof with $\mathrm{comp}(C) > n/3$, we must have $n/3 < \mathrm{comp}(C) \leq 2n/3$. We will show that $M(C)$ contains a large number of variables.

For $\mathit{comp}(C) = t$ will now show that $M(C)$ has at least $(n-t)t \geq 2n^2/9$ distinct literals mentioned. Fix some $i \in \mathit{badpigeons}(C)$ and let α be an i-critical truth assignment falsifying C. For each $j \notin \mathit{badpigeons}(C)$, consider the j-critical assignment, α', obtained from α by replacing i by j, that is, by mapping i to the place that j was mapped to in α. This assignment satisfies C and differs from α only in one place: if α mapped j to l, then α' maps i to l. Since C and $M(C)$ agree on all critical assignments and $M(C)$ is monotone, it must contain the variable $P_{i,l}$.

Running over all $n - t$ j's not in $\mathit{badpigeons}(C)$ (using the same α), it follows that $M(C)$ must contain at least $n - t$ distinct variables $P_{i,l}$, $l \leq n$. Repeating the argument for all $i \in \mathit{badpigeons}(C)$ shows that C contains at least $(n-t)t$ positive literals. □

We note that Theorem 4.1 improves somewhat upon Haken's bound of $2^{n/577}$, although our major interest is in its simpler proof rather than in the better size bound. Buss and Turán [BT88] extend Haken's argument to show that $\neg PHP^m_n$ requires superpolynomial size resolution lower bounds as long as $m < n^2/\log n$. Our argument can be extended to rederive their result.

5. Resolution lower bounds for random formulas. The previous section gave a simple proof that res(F) is large for a specific class of formulas. Abstractly, we can summarize the approach as follows. We assume for contradiction that F has a small proof. In particular, F has a proof with a small number of large clauses. We then modify F (in the above case, restrict some variables) to obtain another formula F' having a proof with no large clauses. We then obtain a contradiction by showing that any proof of F' contains a large clause.

In this section we show how the same idea can be used to obtain simple and improved lower bounds on res(F) that hold with high probability when F is a randomly chosen formula of a given clause density.

5.1. Resolution refutations usually require big clauses. The first main ingredient is a result (essentially from [CS88]) that provides a set of parameterized conditions on a formula F that imply that any resolution refutation of F has at least one large clause. We then show that when F is a random formula of clause density Δ, these conditions hold for certain values of the parameters. We need some definitions.

DEFINITION 5.1. *For a real number σ, a set of clauses $\mathcal{C}$ is* σ-sparse *if $|\mathcal{C}| \leq \sigma|v(\mathcal{C})|$, where $v(\mathcal{C})$ is the set of variables appearing in $\mathcal{C}$.*

DEFINITION 5.2. *If $\mathcal{C}$ is a set of clauses and l is a literal, we say that l is* pure *in $\mathcal{C}$ if some clause of $\mathcal{C}$ contains l and no clause of $\mathcal{C}$ contains $\neg\, l$.*

DEFINITION 5.3. *For $s \geq 1$ and $\epsilon \in (0,1)$, the following properties are defined for formulas F:*

Property $A(s)$: Every set of $r \leq s$ clauses of F is 1-sparse.

Property $B_\epsilon(s)$: For r satisfying $s/2 < r \leq s$, every subset of r clauses of F has at least ϵr pure literals.

The following result is essentially due to Chvátal and Szemerédi (and is closely related to Haken's argument in Lemma 4.2).

PROPOSITION 5.4. *Let $s > 0$ be an integer and F be a CNF formula. If properties $A(s)$ and $B_\epsilon(s)$ both hold for F, then F has no $\epsilon s/2$-bounded proof.*

Proof. The result holds trivially if F is satisfiable, so assume that F is unsatisfiable. We say that a set S of clauses implies a clause C if every assignment that satisfies all of the clauses in S satisfies C. Since F is unsatisfiable, F implies any clause C. The complexity of a clause C with respect to F, comp(C) is the minimum size of a set of clauses that implies C.

Let $\mathcal{P}$ be a resolution refutation of F.

Claim. If F satisfies $A(s)$, then there is a clause $C \in \mathcal{P}$ for which $s/2 < \text{comp}(C) \leq s$.

It is easy to see that if S is a minimal set of clauses that implies C and the literal l is pure in S, then l is in C. Hence for the C given by the above claim, property $B_\epsilon(F)$ implies that $|C| \geq \epsilon\text{comp}(C) \geq \epsilon s/2$.

So it suffices to prove the claim. We first note that if $\mathcal{C}$ is a set of clauses such that any subset is 1-sparse, then it is satisfiable. Indeed, the sparsity condition is equivalent to the hypothesis of the Hall theorem on systems of distinct representatives, and the conclusion of the theorem is that there is a one-to-one mapping sending each clause $C \in \mathcal{C}$ to a variable $v_C \in C$. We can thus satisfy each clause C by appropriately fixing v_C.

Now if S implies Λ, then S is unsatisfiable, so $A(s)$ implies comp(Λ) $> s$. Choose C to be the first clause in $\mathcal{P}$ with comp(C) $\geq s/2$. Since C is the resolvent of two previous clauses C_h, C_j and comp(C) $\leq$ comp(C_h) + comp(C_j) we conclude that comp(C) $< s$. ∎

LEMMA 5.5. *For each integer $k \geq 3$ and $\epsilon > 0$, there are constants $C(k), c_\epsilon(k) > 0$ such that the following holds. Let m, n be integers with $m = \Delta n$ for some $\Delta \geq 1$. Let $F \sim \mathcal{F}_m^{k,n}$.*

1. *If $s \leq C(k)n/\Delta^{1/(k-2)}$, then F satisfies $A(s)$ probability $1 - o(1)$ in s.*
2. *If $s \leq c_\epsilon(k)n/\Delta^{2/(k-2-\epsilon)}$, then F satisfies $B_\epsilon(s)$ with probability $1-o(1)$ in s.*

This lemma is proved by elementary combinatorial probability. We defer the proof until section 5.3 where we state and prove a generalization (Lemma 5.11).

5.2. The formula augmentation method. Armed with these results we give a very simple proof that a random k-CNF F of density Δ satisfies $res(F) \geq 2^{n/\Delta^{O(1)}}$ with probability $1 - o(1)$. An *augmentation* of a formula F is a formula obtained by adding additional clauses to F. As we now describe, augmentations can simplify proofs.

We say that a clause C *subsumes* a clause D if $C \subset D$. Suppose that $\mathcal{P}$ is a proof of F and let G be a CNF formula. We can obtain a proof of the augmented formula $F \wedge G$, denoted $\mathcal{P}\lceil_G$, as follows: For each clause $D \in \mathcal{P}$, if there is a clause C in G such that C subsumes D replace D by C and propagate this simplification forward through the rest of the proof by (possibly) shortening clauses that were produced using D.

Observe that in the case that G consists of clauses of size 1, G corresponds naturally to a restriction ρ, and there is a close correspondence between the proofs $\mathcal{P}\lceil_G$ and $\mathcal{P}\lceil_\rho$.

Following our general approach, suppose we want to prove that $res(F)$ is big. Assuming for contradiction that F has a small proof $\mathcal{P}$, we show that for some integer s and $\epsilon > 0$, there is a G such that (i) G subsumes all clauses of size $\epsilon s/2$ of $\mathcal{P}$ and such that (ii) $F \wedge G$ satisfies $A(s)$ and $B_\epsilon(s)$. This is a contradiction since (i) implies that $\mathcal{P}\lceil_G$ is an $\epsilon s/2$ bounded refutation of $F \wedge G$, while (ii) and Proposition 5.4 imply that no such refutation is possible.

We will realize this approach by considering G chosen at random from some distribution $\mathcal{G}$. We say that the distribution $\mathcal{G}$ satisfies property $g(b, M)$, for $b, M > 0$ if for any clause C of size at least b, if $G \sim \mathcal{G}$, then $\Pr[G \text{ does not subsume } C] \leq 1/M$.

PROPOSITION 5.6. *Let F be a formula. Let $s, M \geq 1$ and $\epsilon > 0$, and let $\mathcal{G}$ be a distribution over formulas that satisfies $g(\epsilon s/2, M)$. Then*

$$res(F) \geq M \times Pr_{G \sim \mathcal{G}}[F \wedge G \text{ satisfies both } A(s) \text{ and } B_\epsilon(s)].$$

Proof. The conclusion follows immediately from the chain of inequalities:

$$\begin{aligned} \mathrm{res}(F)/M &\geq \Pr_{G \sim \mathcal{G}}[\mathcal{P}\lceil_G \text{ is not } \epsilon s/2 \text{ bounded}] \\ &\geq \Pr_{G \sim \mathcal{G}}[F \wedge G \text{ satisfies both } A(s) \text{ and } B_\epsilon(s)]. \end{aligned}$$

The second inequality is immediate from Proposition 5.4. For the first inequality, if $\mathcal{P}$ is a proof of F of size $\mathrm{res}(F)$, it has at most $\mathrm{res}(F)$ clauses of size at least $\epsilon s/2$, and by property $g(\epsilon s/2, M)$ the probability that there is a clause not subsumed by G is at most $\mathrm{res}(F)/M$. □

The above is stated for a fixed formula F. For distributions over formulae we have the following theorem.

THEOREM 5.7. *Let $\mathcal{F}$ be a distribution over formulas. Let $s, M \geq 1$ and $\epsilon > 0$ and suppose that $\mathcal{G}$ is a distribution over formulas that satisfies $g(\epsilon s/2, M)$. Then*

$$\begin{aligned} Pr_{F \sim \mathcal{F}}[res(F) < M/2] \leq 2(&Pr_{F \sim \mathcal{F}, G \sim \mathcal{G}}[F \wedge G \text{ does not satisfy } A(s)] \\ &+ Pr_{F \sim \mathcal{F}, G \sim \mathcal{G}}[F \wedge G \text{ does not satisfy } B_\epsilon(s)]). \end{aligned}$$

Proof. For a formula F, let

$$p(F) = Pr_{G\sim\mathcal{G}}[F \wedge G \text{ does not satisfy } A(s)] + Pr_{\rho}[F\lceil_{\rho} \text{ does not satisfy } B_{\epsilon}(s)].$$

By Proposition 5.6, $\text{res}(F) > (1 - p(F))M$. Thus

$$\text{Pr}_{F\sim\mathcal{F}}[\text{res}(F) < M/2] \leq \text{Pr}_{F}[p(F) > 1/2] < 2E_F[p(F)],$$

where $E[\cdot]$ denotes expectation. This last quantity is equal to the right-hand side of the claimed inequality. □

We now use a form of self-reduction to obtain the following theorem.

THEOREM 5.8. *For each $\mu > 0$, there exists a constant $a_\mu > 0$ such that if $F \sim \mathcal{F}_m^{k,n}$ for $m = \Delta n$ with $\Delta > 1$, then res$(F) \geq 2^{a_\mu n/\Delta^{1+4/(k-2)+\mu}}$ with probability $1 - o(1)$ in n. In particular, when $k = 3$ this reduces to res$(F) \geq 2^{a_\mu n/\Delta^{5+\mu}}$.*

Proof. We apply Theorem 5.7 to the case $\mathcal{F} = \mathcal{F}_m^{k,n}$ by choosing $\mathcal{G} = \mathcal{F}$. Note that $F \wedge G$ has the distribution $\mathcal{F}_{2m}^{k,n}$, so Lemma 5.5 implies that for any $\epsilon > 0$ and for some constant $c_\epsilon(k) > 0$, if $s \leq c_\epsilon(k)n/(2\Delta)^{2/(k-2-\epsilon)}$, then $F \wedge G$ satisfies $A(s)$ and $B_\epsilon(s)$ with probability $1 - o(1)$. (Note that the requirement on s for $B_\epsilon(s)$ is more stringent than that for $A(s)$.) Next, we choose M as large as possible so that $\mathcal{G}$ satisfies $g(\epsilon s/2, M)$. For a clause C of size at least $\epsilon s/2$, the probability that a single randomly chosen clause subsumes C is $\binom{\epsilon s/2}{k}/(2^k\binom{n}{k})$. If $G \sim \mathcal{G}$, then the probability that none of its m clauses subsume C is at most $(1-\binom{\epsilon s/2}{k}/(2^k\binom{n}{k})))^m \leq 2^{-d_\epsilon m s^k \Delta/n^{k-1}}$ for some constant d_ϵ. Substituting $s = c_\epsilon n/\Delta^{2/(k-2-\epsilon)}$, we have that, for sufficiently small ϵ, $\mathcal{G}$ satisfies $g(\epsilon s/2, 2^{e_\epsilon n\Delta^{1+4/(k-2)+O(\epsilon)}})$ for some constant e_ϵ. Hence with probability $1 - o(1)$, $\text{res}(F) \geq 2^{\Omega(n/\Delta^{1+4/(k-2)+\alpha})}$ for any $\alpha > 0$. □

COROLLARY 5.9. *For any $k \geq 3$ and $\epsilon > 0$, there is a constant γ such that almost all k-CNF formulas in n variables with at most $n^{2k/(k+2)-\epsilon}$ clauses require resolution proofs of size at least 2^{n^γ}.*

These results provide strong lower bounds on res(F) for random formulas. However, note that as k gets large, the exponent in the lower bound of res(F) tends to n/Δ, while in the upper bound obtained by using ordered DLL, the exponent is $n/\Delta^{1/(k-2)}$. We'd like to close this gap.

Observe that for property $A(s)$, Lemma 5.5 requires $s = O(n/\Delta^{1/(k-2)})$ while for property $B_\epsilon(s)$ it requires $s \leq O(n/\Delta^{2/(k-2-\epsilon)})$. It turns out that one way to significantly close the above gap would be to show that the second part of Lemma 5.5 holds if we weaken the bound on s.

Problem. Is it true that if $F \sim \mathcal{F}_m^{k,n}$, then F satisfies $B_\epsilon(s)$ with probability near 1 for $s \leq n\Delta^{1/(k-2-\epsilon)+o(1)}$?

If this were true, then the argument used in the above theorem would be improved substantially to the following: for $F \sim \mathcal{F}_m^{k,n}$, with probability near 1, res$(F) \geq 2^{n/\Delta^{2/(k-2-\epsilon)}}$, which is very comparable to the upper bound. The corollary is improved so that m can be as large as $n^{(k-\epsilon)/2}$. We discuss this problem further in section 7.

Lacking an affirmative answer to the above problem, we look for other ways to improve our result. In section 5.3, we will see that we can narrow the gap substantially using random restrictions instead of augmentations.

5.3. The random restriction method. We now apply an approach analogous to the above, using restrictions instead of augmentations. As mentioned above, applying a restriction can be viewed as applying an augmentation consisting of clauses of size one, but we use the language of restrictions because it is more familiar and natural.

Specializing Theorem 5.7 to the case of restrictions yields the following. If $\mathcal{R}$ is a probability distribution over the set of restrictions, we say that $\mathcal{R}$ has *property* $R(b, M)$ if for $b, M > 0$ for any clause C on X of size at least b, $\Pr[\rho \text{ does not satisfy } C] \leq 1/M$. Then we have the following theorem.

THEOREM 5.10. *Let $\mathcal{F}$ be a distribution over k-bounded formulae. Let s, $M \geq 1$ and $\epsilon > 0$ and suppose that $\mathcal{R}$ is a distribution over restrictions that satisfies $R(\epsilon s/2, M)$. Then*

$$Pr_{F\sim\mathcal{F}}[res(F) < M/2] \leq 2(Pr_{F\sim\mathcal{F},\rho\sim\mathcal{R}}[F\lceil_\rho \text{ does not satisfy } A(s)] + Pr_{F\sim\mathcal{F},\rho\sim\mathcal{R}}[F\lceil_\rho \text{ does not satisfy } B_\epsilon(s)]).$$

We will use this result to get a lower bound on $\mathrm{res}(F)$ for random k-CNF formulas, using the distribution $\mathcal{R}_t$ over restrictions where we first choose $v(\rho) \subseteq X$ by selecting each variable independently with probability t/n and then set the selected variables uniformly at random.

Lemma 5.5 needs to be generalized to formulas obtained from a random k-CNF by applying a random restriction.

LEMMA 5.11. *For each integer $k \geq 3$ and $\epsilon > 0$ there are constants $C(k), c_\epsilon(k) > 0$ such that the following holds. Let m, n, s, t be integers with $m = \Delta n$ for some $\Delta \geq 1$. Let $F \sim \mathcal{F}_m^{k,n}$ and $\rho \sim \mathcal{R}_t$.*

1. *If $t \leq C(k)n/m^{1/k}$ and $s \leq C(k)n/\Delta^{1/(k-2)}$, then $F\lceil_\rho$ satisfies $A(s)$ probability $1 - o(1)$ in s.*
2. *If $s, t \leq c_\epsilon(k)n/\Delta^{2/(k-2-\epsilon)}$, then $F\lceil_\rho$ satisfies $B_\epsilon(s)$ with probability $1 - o(1)$ in s.*

The proofs of these lemmas require a preliminary result. Let F and ρ be as in the statements of the lemmas. Let M denote the event that $F\lceil_\rho$ contains an empty clause. For $r, q > 0$, let $Q(r, q)$ denote the event that there exists a set R of at most r variables such that $v(C) \subseteq R$ for at least q nonempty clauses C of $F\lceil_\rho$.

PROPOSITION 5.12. *Let $m \geq n \geq t \geq k \geq 3$ be positive integers.*

1. *If $t \leq n/m^{1/(k-1)}$, then $Pr[M] \leq m^{-1/(k-1)}$.*
2. *Assume $n \geq 2k^2$. For $r, q \geq 1$, $Pr[Q(r, q)] \leq \left(\frac{ne}{r}\right)^r \left(\frac{2ekmr(t+r)^{k-1}}{qn^k}\right)^q$.*

Proof. For the first part, if C is a fixed k-clause, then $C\lceil_\rho$ is empty if and only if ρ sets all literals in C to 0. Thus $\Pr_C[C\lceil_\rho \text{ is empty}] = (t/2n)^k$. Therefore, $\Pr[M]$ is upper bounded by the expected number of empty clauses of $\mathcal{F}\lceil_\rho$ which is $m(\frac{t}{2n})^k \leq m^{-1/(k-1)}$.

For the second part, let $R \subseteq X$ with $|R| = r \geq 1$. Let C denote a randomly chosen clause and $I = v(C) \cap R$. Let $W(C)$ denote the event that $C\lceil_\rho$ is nonempty and its variables are contained in R. Then $\Pr[W(C)] = \sum_{i=1}^k \Pr[|I| = i]\Pr[v(C) - I \subseteq v(\rho) : |I| = i]$. Now $\Pr[v(C) - I \subseteq v(\rho) : |I| = i] = (t/n)^{k-i}$, while $\Pr[|I| = i] = \binom{r}{i}\binom{n-r}{k-i}/\binom{n}{k} \leq \binom{r}{i}\frac{k!}{(k-i)!}(n-k)^i \leq \binom{k}{i}(\frac{r}{n-k})^i \leq 2\binom{k}{i}(\frac{r}{n})^i$, where the last inequality holds since we assume $n \geq 2k^2$. Thus

$$\begin{aligned}\Pr[W(C)] &\leq 2\sum_{i=1}^{k}\binom{k}{i}\left(\frac{r}{n}\right)^i\left(\frac{t}{n}\right)^{k-i} \\ &\leq \frac{2kr}{n}\sum_{j=0}^{k-1}\binom{k-1}{j}\left(\frac{r}{n}\right)^j\left(\frac{t}{n}\right)^{k-1-j} \\ &= \frac{2kr}{n}\left(\frac{r+t}{n}\right)^{k-1}.\end{aligned}$$

Calling this latter probability p, if F is a random formula, the number of clauses of F for which $W(C)$ holds has the binomial distribution, $B(m,p)$. The probability that at least q clauses of F are contained in S after ρ is applied is bounded above by

$$\Pr[B(m,p) \geq q] \leq \binom{m}{q} p^q \leq \left(\frac{2ekmr(r+t)^{k-1}}{qn^k}\right)^q. \tag{5.1}$$

Summing this over the $\binom{n}{r} \leq (en/r)^r$ subsets of X of size r we obtain the desired upper bound on $\Pr[Q(r,q)]$. □

Proof of Lemma 5.11. We begin with the first part. The probability that $A(s)$ fails is at most $\sum_{r=1}^{s} \Pr[Q(r,r+1)] + \Pr[M]$. By the first part of Proposition 5.12, $\Pr[m] = o(1)$ in m and hence also in s. For $r \leq s$ we have

$$Q(r,r+1) \leq \left(\frac{ne}{r}\right)^r \left(\frac{2ekmr(r+t)^{k-1}}{(r+1)n^k}\right)^{r+1} \tag{5.2}$$

$$\leq \frac{r}{en}\left(\frac{2ekm(r+t)^{k-1}}{(r+1)n^{k-1}}\right)^{r+1}. \tag{5.3}$$

For $t \leq r \leq s$, and for some constant $C_1(k) > 0$, if $s \leq C_1(k)n/\Delta^{1/(k-2)}$, the quantity (5.3) is at most $\frac{r}{2^r n}$. Similarly, for $1 \leq r < t$, the quantity (5.3) is at most $\frac{r}{n}(C_2(k)m(\frac{t}{n})^{k-1})^{r+1}$ which is at most $\frac{r}{2^r n}$ for $t \leq C_3(k)n/m^{1/(k-1)}$ for some positive constants $C_2(k), C_3(k)$. Thus the $\sum_{r=1}^{s} Q(r,r+1)$ is at most $\frac{1}{n}\sum_{r=1}^{s}\frac{r}{2^r} < 2/n$ which is $o(1)$ in n and hence in s.

Finally, in the hypothesis of the lemma, we take $C(k) = \min(C_1(k), C_2(k))$.

Now for the second part of the lemma. We first observe

$$\Pr[\neg\, B_\epsilon(s)] \leq \sum_{\lfloor s/2\sigma \rceil \leq r \leq s/\sigma} \Pr[Q(r,\sigma r)] + \Pr[M],$$

where $\sigma = 2/(k+\epsilon)$. To see this, suppose that $B_\epsilon(s)$ fails. We want to show that either M holds or $Q(r,\sigma r)$ holds for some r in the given range. Assume M does not hold. Since $B_\epsilon(s)$ fails, there is a collection $\mathcal{C}$ of w clauses that has at most ϵw pure literals with $s/2 \leq w \leq s$. We upper bound $|v(\mathcal{C})|$. The sum of the clause sizes is wk and if u is the number of pure literals, the impure variables contribute at most $wk-u$ to this sum. Each impure variable appears in at least two clauses, so the number of impure variables is at most $\frac{wk-u}{2}$ and thus $|v(\mathcal{C})| \leq \frac{wk-u}{2} + u = \frac{wk+u}{2} \leq \frac{w(k+\epsilon)}{2}$. Extend the set $v(\mathcal{C})$ to a set R of size $r = \lfloor \frac{w(k+\epsilon)}{2} \rfloor$ so that $\lfloor \frac{s}{2\sigma} \rfloor \leq r \leq \frac{s}{\sigma}$ and R contains at least σr clauses. Thus $Q(r,\sigma r)$ holds.

So it suffices to upper bound the sum of $Q(r,\sigma r)$ for $\lfloor \frac{s}{2\sigma} \rfloor \leq r \leq \frac{s}{\sigma}$. We have

$$\begin{aligned} Q(r,\sigma r) &\leq \left(\frac{ne}{r}\right)^r \left(\frac{2ekm(r+t)^{k-1}}{\sigma n^k}\right)^{\sigma r} \\ &= \left(\frac{(ne)^{1/\sigma} C_1 m(r+t)^{k-1}}{\sigma r^{1/\sigma} n^k}\right)^{\sigma r} \\ &\leq \left(\frac{C_4 m r^{-1/\sigma}(r+t)^{k-1}}{n^{k-1/\sigma}}\right)^{\sigma r} \;<\; 2^{-\sigma r} \end{aligned}$$

for appropriate constant $C_4(k)$, provided that

$$\frac{C_4 m r^{-1/\sigma}(r+t)^{k-1}}{n^{k-1/\sigma}} \leq \frac{1}{2}.$$

For $\sigma = 2/(k+\epsilon)$, this last condition is satisfied if both s and t are at most $c_\epsilon(k)n \cdot (\frac{n}{m})^{2/(k-2-\epsilon)}$ for some constant $c_\epsilon(k) \geq 0$. Now the total failure probability for property $B_\epsilon(s)$ is at most $\sum_{r=\lfloor \frac{s}{2\sigma} \rfloor}^{s} 2^{-\sigma r} + \Pr[M]$ which is clearly $o(1)$ in s and part 2 is proved. □

Next, in order to apply Theorem 5.10, we determine as large an M as possible such that the distribution $\mathcal{R}_t$ satisfies $R(\epsilon s/2, M)$, where s and t are the largest numbers satisfying the hypotheses of both parts of the previous lemma.

LEMMA 5.13. *$\mathcal{R}_t$ satisfies $R(\epsilon s/2, e^{\epsilon s t/4n})$.*

Proof. For a fixed clause C and for $\rho \sim \mathcal{R}_t$, a variable x that is in C is fixed by ρ to satisfy C with probability $t/2n$, so the probability that ρ does not satisfy a given clause C is at most $(1-t/2n)^{|C|} \leq e^{-|C|t/2n}$. □

Applying Theorem 5.10 using Lemmas 5.11 and 5.13 yields the following theorem.

THEOREM 5.14. *For any integer $k \geq 3$ and $\epsilon > 0$ there is a constant $c'_\epsilon > 0$ such that the following holds. Let $F \sim \mathcal{F}_m^{k,n}$, where $m = \Delta n$ with Δ at least the satisfiability threshold θ_k.*

(i) *If $m \leq c'_\epsilon n^{2(k-1)/(k+\epsilon)}$, then res$(F) \geq 2^{\Omega(n^{1-1/(k-1)}/\Delta^{1/(k-1)+2/(k-2-\epsilon)})}$ with probability $1-o(1)$ in n.*

(ii) *If $m \geq c'_\epsilon n^{2(k-1)/(k+\epsilon)}$, then res$(F) \geq 2^{\Omega(n/\Delta^{4/(k-2-\epsilon)})}$ with probability $1-o(1)$ in n.*

Proof. Let $C(k)$ and $c_\epsilon(k)$ be the constants from Lemma 5.11. Let $s = c_\epsilon(k)n/\Delta^{2/(k-2-\epsilon)}$. The two cases of the theorem correspond to the two constraints on t in Lemma 5.11. The constraint $t \leq C(k)n/m^{1/(k-1)}$ is the more stringent constraint if and only if $m^{1/(k-1)} \geq (c_\epsilon(k)/C(k))(m/n)^{2/(k-2-\epsilon)}$. This holds if and only if $n \geq c''(k)m^{1-(k-2-\epsilon)/(2k-2)} = c''(k)m^{(k+\epsilon)/(2k-2)}$ for some constant $c''(k) > 0$, i.e., if $m \leq c'_\epsilon n^{(2k-2)/(k+\epsilon)}$ for some constant $c'_\epsilon > 0$. In this case, applying Lemma 5.13 yields the first bound. In the case that $m > c'_\epsilon n^{(2k-2)/(k+\epsilon)}$, the constraint $t \leq c_\epsilon n(n/m)^{2/(k-2-\epsilon)}$ is the more stringent and Lemmas 5.13 and 5.11 yield the second bound. □

When $k = 3$, the maximal possible value of m which still yields a lower bound of the form 2^{n^γ} is $n^{6/5-\epsilon}$, which matches the result obtained for $k = 3$ in Corollary 5.9. For $k \geq 4$ we obtain the following corollary.

COROLLARY 5.15. *For $k \geq 4$, and $\epsilon > 0$, there is a constant $\gamma > 0$ such that almost all k-CNF formulas with at most $n^{(k+2)/4-\epsilon}$ clauses require resolution proofs of size at least 2^{n^γ}.*

Proof. To see this, assume that $m = n^{(k+2)/4-\epsilon}$. Since $k \geq 4$ if we take $\epsilon' = 4\epsilon$, then $(k+2)/4-\epsilon = 1+(k-2-\epsilon')/4 > 1+(k-2-\epsilon')/(k+\epsilon') = (2k-2)/(k+\epsilon')$. Thus we can apply the second bound of Theorem 5.14 to derive that k-CNF formulas with at most m clauses almost certainly require resolution proofs of size $2^{\Omega(n/\Delta^{4/(k-2+\epsilon')})}$ which is $2^{\Omega(n^{2\epsilon'/(k-2+\epsilon')})}$. □

5.4. The deletion argument. In this subsection we sketch a variant of the restriction approach, which, in a preliminary version of this work [BKPS98], was shown to yield the following theorem.

THEOREM 5.16. *For each $\gamma > 0$, there exists a constant a_γ such that for all $m \geq n$, if F is a 3-CNF formula chosen according to $\mathcal{F}_m^{3,n}$, then with probability $1-o(1)$, res$(F) \geq 2^{a_\gamma(n/\Delta^{4+\gamma})}$.*

Observing that this bound is nontrivial for $m = o(n^{5/(4+\gamma)})$ and combining with Corollary 5.15 we obtain the following corollary.

COROLLARY 5.17. *For $k \geq 3$, and $\epsilon > 0$, there is a constant $\gamma > 0$ such that almost all k-CNF formulas with at most $n^{(k+2)/4-\epsilon}$ clauses require resolution proofs of size at least 2^{n^γ}.*

The proof of Theorem 5.16 in [BKPS98] is rather technical. The recent work of Ben-Sasson and Wigderson [BSW99] referred to earlier shows how one can derive the same bound in a substantially simpler way given Proposition 5.4 and Lemma 5.5. Therefore, we do not include our entire proof but instead give a short sketch.

The major bottleneck in the argument of section 5.1 is the upper bound on t needed for Lemma 5.11. Indeed, for t much larger than $n/\sqrt{m}$, there is a substantial probability that $A(s)$ does not hold for $F\lceil_\rho$. In particular, the bound computed in the proof of Lemma 5.11 on the probability that a clause of a random F becomes empty is nearly tight; an easy computation shows that the probability that no clause of a random F becomes empty when ρ is applied is $e^{-\Theta(mt^3/n^3)}$ and so, since the presence of an empty clause in $F\lceil_\rho$ violates $A(s)$, we have that $t = o(n/m^{1/3})$. (The creation of unit clauses under the restriction placed an even stronger limitation on t.)

To overcome this limitation, we want to avoid the creation of clauses of size 0 or 1 in $F\lceil_\rho$. To do this we modify the distribution on restrictions so that ρ may depend on F. The general idea is to first choose a random restriction and then delete any assignment that sets more than one variable in any clause of F. For technical reasons one must also delete assignments to variables that share some clause with too many other variables. By careful arguments one can show appropriate analogues of Theorem 5.10 and Lemma 5.11, allowing the condition $t \le cn/\sqrt{m}$ on the size of the restriction to be eliminated.

6. The behavior of DLL on random formulas. We now analyze particular DLL algorithms when applied to random formulas and show that we can obtain quite good upper bounds using a very simple splitting rule and that even certain generalizations of this splitting rule require much larger proofs than our current lower bound for general resolution shows.

6.1. Ordered DLL on random formulas.

THEOREM 6.1. *Let $k \ge 3$ and let $m = \Delta n$, where Δ is greater than the threshold $\theta_k(n)$ and $m = \Delta n$. Suppose that $F \sim \mathcal{F}_m^{k,n}$. Then with probability $1 - o(1)$ in n, the size of the refutation of F produced by ordered DLL is $2^{\Theta(n/\Delta^{1/(k-2)})}n^{\pm\Theta(1)}$. In particular, when $k = 3$, the refutation has size $2^{\Theta(n/\Delta)}n^{\pm\Theta(1)}$.*

The proof of this result has two parts, the upper bound, and lower bound, which we prove separately. To analyze the upper bound, we first consider a variant of ordered DLL that is easier to analyze.

Upper bound for ordered DLL. We first consider a variant of ordered DLL which is bit more complicated to state but easier to analyze.

Algorithm A. Set $t = 6k\lceil n(n/m)^{1/(k-2)}\rceil$; in particular, when $k = 3$ this is $18\lceil n^2/m\rceil$. Run ordered DLL, as long as the variables $x_1, \ldots, x_t$ are not all assigned. When reaching a partial assignment ρ in which $x_1, \ldots, x_t$ are all assigned (and possibly other variables by unit propagation), run the (polynomial-time) algorithm for 2-SAT on the subset $C_2(F, \rho)$ consisting of clauses of size at most 2 in the induced subformula $F\lceil_\rho$. The algorithm succeeds (finds a resolution refutation of F) provided that for each such ρ reached in the algorithm the subformula $C_2(F, \rho)$ is unsatisfiable.

To analyze Algorithm A we need the following lemma.

LEMMA 6.2. *Let F be a random 2-CNF formula chosen from $\mathcal{F}_{2n'}^{2,n'}$. Then the probability that F is satisfiable is $o(2^{-n'/9})$.*

Proof. Observe that the expected number of satisfying assignments for a 2-CNF formula with m' clauses and n' variables is $2^{n'}(3/4)^{m'}$ which is $o(2^{-n'/9})$

for $m' > 2.678n'$. (This bound can be reduced below 2 by using the techniques of [KKKS98].) □

THEOREM 6.3. *Let Δ be greater than the satisfiability threshold $\theta_k(n)$ and $m = \Delta n$. If $F \sim \mathcal{F}_m^{k,n}$, then, with probability $1 - o(1)$ in n, Algorithm A produces a resolution refutation in time $2^{O(n/\Delta^{1/(k-2)})}n^{O(1)}$; in particular, if $k = 3$ Algorithm A produces a resolution refutation in time $2^{O(n/\Delta)}n^{O(1)}$.*

Proof. Let $t = 6k\lceil n(n/m)^{1/(k-2)}\rceil$ and assume without loss of generality that $t < n/10$. Algorithm A clearly runs in time $2^{O(n/\Delta^{1/(k-2)})}n^{O(1)}$. To show that Algorithm A finds a refutation of F with probability $1 - o(1)$, it suffices to show that with probability $1 - o(1)$, $C_2(F, \rho)$ is unsatisfiable for all assignments ρ to $\{x_1, \ldots, x_t\}$. (Note that the restrictions occurring in the algorithm may have additional variables fixed by unit propagation, but this can only increase our probability of success.)

Fix ρ. Consider the set $\hat{C}_2(F, \rho)$ of clauses of size exactly 2. This size is a binomial random variable $B(m, q)$, where q is equal to the probability, for a random k-clause C, that $C\lceil_\rho$ is a 2-clause:

$$\begin{aligned}
\Pr[C\lceil_\rho \text{ is a 2-clause}] &= \frac{1}{2^{k-2}} \cdot \Pr[|\{x_1, \ldots, x_t\} \cap v(C)| = k-2] \\
&= \frac{1}{2^{k-2}} \cdot \left(\frac{\binom{n-t}{2}\binom{t}{k-2}}{\binom{n}{k}} \right) \\
&= \frac{1}{2^{k-2}} \cdot \left(\binom{k}{2} \frac{n-t}{n} \frac{n-t-1}{n-1} \frac{t}{n-2} \frac{t-1}{n-3} \cdots \frac{t-k+3}{n-k+1} \right) \\
&> \frac{1}{2^{k-2}} \binom{k}{2}^2 \frac{4}{5} \left(\frac{t}{2n} \right)^{k-2} \\
&> \binom{k}{2} \frac{4}{5} \left(\frac{3k}{2} \right)^{k-2} \frac{n}{m} \\
&> 8n/m.
\end{aligned}$$

Using the binomial tail bound of Proposition 2.1 (1), it follows that $\Pr[|\hat{C}_2(F, \rho)| \leq 2n] \leq 2^{-4n}$. By Lemma 6.2 and the fact that the clauses in $\hat{C}_2(F, \rho)$ are distributed uniformly at random on the remaining $n' = n - t$ variables,

$$\Pr[\hat{C}_2(F, \rho) \text{ is satisfiable} : |\hat{C}_2(F, \rho)| > 2n] = o(2^{-n'/9}).$$

Since there are 2^t choices for ρ, the total failure probability is $2^t \cdot (o(2^{-(n-t)/9}) + 2^{-n})$, which is $o(1)$ since $(n - t)/9 \geq t$ for $t \leq n/10$. □

Next we consider ordered DLL and prove the upper bound of Theorem 6.1. At a point in the execution of DLL, say that a variable is *critical* if setting that variable either to 0 or 1 and then applying unit propagation creates the empty clause. Thus, if the splitting rule chooses that variable the current branch will terminate simply by unit propagation.

A point in the execution of DLL corresponds to some restriction ρ. We give a sufficient condition for a variable to be critical in terms of the set $\hat{C}_2(F, \rho)$ of induced 2-clauses on the remaining set of n' variables. Define the standard directed graph $G(F, \rho)$ on $2n'$ vertices, one for each literal, that has directed edges $(\neg x, y)$, and $(\neg y, x)$ corresponding to each 2-clause $(x \vee y)$ in $\hat{C}_2(F, \rho)$. It is easy to see that a sufficient condition for the variable x_i to be critical is that there be directed paths from x_i to $\neg x_i$ and from $\neg x_i$ to x_i, i.e., that x_i and $\neg x_i$ lie in the same strongly connected component.

LEMMA 6.4. *For any $k \geq 3$, there exists a constant c such that if $F \sim \mathcal{F}_m^{k,n}$ and ρ is a fixed restriction of t variables with $n/2 \geq t \geq c\lceil n(n/m)^{1/(k-2)}\rceil$, then with probability at least $1-2^{-n}$, for at least half of the $n' = n-t$ unrestricted variables, x_i and $\neg x_i$ belong to the same strongly connected component of $G(F,\rho)$.*

Proof. Clearly, it suffices to show that with probability at least $1-2^{-n}$, $G(F,\rho)$ has a strongly connected component of size at least $3n'/2$. Let $C_1, C_2, \ldots, C_d$ be the strongly connected components ordered so that all edges between components go from lower to higher numbered components and consider the first j such that $|C_1 \cup \cdots \cup C_j| \geq n'/4$. We will show that the probability that $|C_j| < 3n'/2$ is at most 2^{-n}. If $|C_j| < 3n'/2$, then the set $S = C_1 \cup \cdots \cup C_j$ satisfies $n'/4 \leq |S| \leq 7n'/4$, and there is no edge from $\bar{S}$ to S.

So to upper bound the probability that $|C_j| < 3n'/2$ it suffices to upper bound the probability that there is a set S with $n'/4 \leq |S| \leq 7n'/4$ which is *bad* in the sense that there is no edge from $\bar{S}$ to S. Fix S of size s, with $n'/4 \leq s \leq 7n'/4$. The probability that a randomly chosen k-clause C, when restricted by ρ, gives an edge from $\bar{S}$ to S is at least $s(n'-s-1)\binom{t}{k-2}/2^k\binom{n}{k} \geq \beta'(t/n)^{k-2} \geq \beta\lceil n/m\rceil$ for some constants $\beta, \beta' > 0$ depending only on k. Hence the probability that none of the m clauses of F gives such an edge is at most $(1-\beta n/m)^m \leq e^{-\beta cn} \leq 2^{-3n}$ for c chosen greater than $3/\beta$. There are at most $2^{2n'}$ such sets S, so the probability that there is a bad set S of size between $n'/4$ and $7n'/4$ is at most 2^{-n}. □

Proof of the upper bound of Theorem 6.1. Without loss of generality we may assume that $m \geq (4c)^{k-2}n$ where c is the constant of the previous lemma, and let $t = cn(n/m)^{1/(k-2)}$ so that $t \leq n/4$.

Fix a restriction ρ of the first t variables. We claim that the probability that there is a branch of the DLL tree consistent with ρ that is still active (not terminated) after the first $4t$ variables are set and the resulting unit propagations are processed is at most 2^{-2t}. Since there are 2^t choices for ρ, this will imply that with probability $1-2^{-t}$, every branch of ordered DLL is completed after at most the first $4t$ variables are fixed and all resulting unit propagations are done, and so the tree has at most $n2^{4t}$ nodes (including nodes from unit propagation).

To prove the claim, condition on the size r of the set of critical variables for $F\lceil_\rho$. By Lemma 6.4, the probability that $r < n'/2$ is at most $2^{-n} \leq 2^{-4t}$, so we assume $r \geq n'/2$. The set of critical variables is equally likely to be any r-subset of the $n' = n-t$ unset variables, and so the probability that none of the next $3t$ variables in order are critical is at most $\binom{n'-3t}{r}/\binom{n'}{r} \leq (1-3t/n')^r \leq e^{-3t/2}$. Hence the probability that some branch consistent with ρ is unfinished after fixing the next $3t$ variables is at most $2^{-4t} + e^{-3t/2} \leq 2^{-2t}$.

Note that in order to obtain the claimed upper bound with probability $1-o(1)$ in n, we can assume without loss of generality that t is at least $\log n$. (If m, the number of clauses, is large enough so that t is less than $\log n$, we can carry out the analysis with fewer clauses since the complexity of ordered DLL decreases monotonically with the number of clauses.) □

Lower bound for ordered DLL. We now complete the proof of Theorem 6.1 by proving the lower bound.

Proof of the lower bound for Theorem 6.1. Fix $t < n(\frac{1}{4k\Delta})^{1/(k-2)}$ and let S be the first t variables with respect to the given ordering and $L(S)$ be the associated set of $2t$ literals. Let F' denote the set of all clauses of F that contain at least $k-1$ literals from $L(S)$.

Claim. With probability $1-o(1)$ in t, there is a partial assignment τ to $t/2$ of the variables of S that satisfies all clauses in F'.

Assuming the claim, we finish the proof by noting that for each of the at least $2^{t/2}$ restrictions ρ to S that are compatible with τ, all clauses of $F\lceil_\rho$ will have size at least 2. This implies that when applying ordered DLL along the path specified by ρ, no variables outside of S are fixed by unit propagation, and so there is a unique node corresponding to ρ, and hence there are at least $2^{t/2}$ nodes in the tree. Without loss of generality, we can assume that t is at least $\log n$; since $t < \log n$, our claimed lower bound is just 1.

So we prove the claim. For each $C \in F$, the probability q that it is in F' is at most $k(\frac{t}{n})^{k-1} \leq t\frac{k}{n}(\frac{t}{n})^{k-2} \leq \frac{t}{4m}$. Construct a 2-CNF F'' of size $|F'|$ by replacing each clause C' of F' by a clause C'' obtained by selecting two literals of $C' \cap L(S)$ uniformly at random. It is easy to see that F'' is a randomly chosen 2-CNF whose number of clauses is binomially distributed according to $B(m, q)$. The tail bound of Proposition 2.1 (2) implies that $|F''| < t/2$ with probability $1 - o(1)$. Conditioned on $|F''| < t/2$, the probability that F'' (and hence F') is satisfiable is $1 - o(1)$ because a random 2-CNF formula with $(1 - \epsilon)t$ variables on a set of size t is satisfiable with probability $1-o(1)$ [Goe96, CR92]. If F'' is satisfiable, there is a setting τ of at most $|F''| < t/2$ variables of S that satisfies it. This completes the proof of the claim and the theorem. □

6.2. Lower bounds for more general DLL procedures. Having analyzed ordered DLL, we next turn to the problem of proving lower bounds for a wider class of DLL procedures. In the following discussion it will be useful to introduce some additional terminology. Let A be a DLL algorithm and F be a formula, and let $T_A(F)$ denote the DLL tree associated to the execution of A on F. We classify the nodes of $T_A(F)$ into two types: *unit propagation nodes* (those corresponding to a variable in a unit clause) and *branching nodes*.

We will prove a lower bound for a class of algorithms called *oblivious DLL* algorithms. Let B_n denote the full binary tree of depth n and let λ be a labeling of the internal nodes by literals with the property that along each root-leaf path each variable occurs in exactly one literal. The labeling λ specifies an algorithm $A = A_\lambda$ as follows. On input F, A_λ recursively traverses B_n starting from the root. When arriving at node v it has a formula G which is a restriction of F, and it performs a procedure Test(G, v) defined as follows. While G has at least one unit clause but no empty clause, choose a unit clause and fix its literal to true, simplifying G accordingly. If the empty clause is produced, stop; the formula is unsatisfiable. If G ever has no clauses, then stop; the formula is satisfiable. Otherwise, when G has no empty or unit clauses, let v_0 and v_1 denote the left and right children of v. If the literal $\lambda(v)$ is already assigned a value $i \in \{0, 1\}$, move to v_i and run Test(v_i, G). Otherwise, run Test$(v_0, G\lceil_{\lambda(v)=0})$ and Test$(v_1, F\lceil_{\lambda(v)=1})$ and return unsatisfiable if both return unsatisfiable, and satisfiable if at least one of them returns satisfiable.

In the special case that B_n is labeled so that each node of distance i from the root is labeled x_i the above algorithm is ordered DLL. Another case of interest is that of *random DLL* in which the labeling of B_n is chosen at random. In general, we call such algorithms *oblivious* because (except for unit propagation) the choice of splitter does not depend on the function F.

It is worth emphasizing the distinction between the labeled tree (B_n, λ) (which is independent of F) and the DLL tree $T_{A_\lambda}(F)$ associated to an execution of the algorithm on F.

It is not hard to show that if $F \sim \mathcal{F}_m^{k,n}$, then the expected running time of any oblivious algorithm is the same. However, this does not rule out the possibility that

some oblivious algorithm may run much faster than ordered DLL on *most* instances by concentrating the bad behavior on a small set of instances. Here we show that provided that m is big enough, no oblivious algorithm can do much better than ordered DLL on most instances (see Theorem 6.8 below).

In the lower bound on ordered DLL, we showed that during the first t steps, with high probability unit propagation plays no role. We will prove something like this for oblivious algorithms. The key step is a lemma that implies that for any DLL procedure (oblivious or not) on a random formula, the number of variables fixed by unit propagation along any path in the DLL tree is not much more than the number of branching nodes along the path. For technical reasons, it is easier to consider a generalization of unit propagation that ignores the sign of variables.

The general idea is as follows. Let F be a formula and T a set of variables that have been set so far. We want to describe a natural algorithm that defines a set $\hat{T}$, where $\hat{T}$ will contain T plus the set of additional variables that are set by unit propagation. The algorithm is as follows. Initially $\hat{T} = T$. Given F, let S be the corresponding set system, where each k-clause of F corresponds to a k-set in S (over a universe of size n.) Given F (and hence S), and T, define $S' \subseteq S$ to be those clauses that intersect $\hat{T}$ in exactly $k-1$ elements. Add the elements occurring in S' that are not already in $\hat{T}$ to $\hat{T}$ and continue recursively until S' is empty. Note that the set $\hat{T}$ produced by this algorithm is a minimal set of variables that will be set by setting the variables in T, plus the additional variables set by unit-clause propagation, assuming that we ignore early termination as a result of the formula being set to 0 or 1. The intuition behind the proof of Theorem 6.8 is that with high probability, the size of $\hat{T}$ will be not much larger than the size of T. The proof of this fact will be a compression argument showing that for each fixed T, if F has the property that $\hat{T}$ is large, then F can be encoded succinctly. Intuitively, if there are a lot of variables that become unit clauses, then these clauses are not random with respect to T and therefore can be described succinctly.

We need some definitions. Given a k-CNF formula F, a set T of variables is *closed* with respect to F if no clause contains exactly $k-1$ variables of T (either positively or negatively). It is easy to see that the intersection of closed sets is closed, and hence for any set T of variables the set $\hat{T}$ obtained by intersecting all closed sets containing T yields the unique minimal closed set containing T. We call this the *closure of* T *(with respect to* F*)*. A clause is said to be *threatened* by (F, T) if it is contained in $\hat{T}$. Note that $\hat{T} = T \cup \cup_D v(D)$, where D ranges over all clauses of F threatened by (F, T). The following fact relates these notions to unit propagation.

PROPOSITION 6.5. *Let A be any DLL algorithm and F be a k-CNF formula. Suppose v is a node in the DLL tree of $T_A(F)$. Let T be the variables that appear at the branching nodes on the path to v. Then*

1. *the variables at the unit propagation nodes on the path to v are contained in $\hat{T}$;*
2. *the only clauses that can be empty upon reaching v are the clauses threatened by (F, T).*

Proof. For the first part, let $v_0, \ldots, v_j = v$ be the nodes on the path to v and let x_i be the variable whose literal l_i appears at v_i. Applying induction on i, we assume that $\{x_0, \ldots, x_{i-1}\} \subseteq \hat{T}$. Then if v_i is a branching node, then $x_i \in T$ and otherwise, if ρ is the restriction defined by the literals $l_0, \ldots, l_{i-1}$, there is a clause C of F such

that $C\lceil_\rho$ is the unit clause l_i. If $\hat{T}$ does not contain x_i, then C contains exactly $k-1$ variables of $\hat{T}$, which contradicts that $\hat{T}$ is closed.

For the second part, a clause that becomes empty must have all its literals set to 0, which means that all of its variables are in $\hat{T}$. □

We next present an algorithm for constructing $\hat{T}$ from T; the details of this algorithm are needed in what follows. Fix an ordering $x_1, \ldots, x_n$ of the variables. The algorithm $Close(F, T)$ takes as input a formula F (viewed as a list $C_1, \ldots, C_m$ of clauses) and subset T of variables with $t = |T|$, and outputs the following:

(i) A sequence $D_1, \ldots, D_u$ of distinct clauses of F, consisting of the set of clauses threatened by (F, T).

(ii) A sequence $z_1, \ldots, z_{t+u}$ of variables consisting of the variables of $\hat{T}$ (possibly with repetition). Furthermore, $T = \{z_1, \ldots, z_t\}$, listed according to the global variable ordering, and for $i \in [u]$, $z_{t+i} \in v(D_i) \subseteq \{z_1, \ldots, z_{t+i}\}$.

(iii) A sequence $j_1 \leq \cdots \leq j_u$ of integers in the range 1 to $t+u$, where j_i is the least index such that $|\{z_1, \ldots, z_{t+u}\} \cup D_i| = k-1$.

We will build these three lists in a series of iterations, which we divide into two phases. The initialization phase consists of the first t iterations, where we place the variables of T on the list, constructing $z_1, \ldots, z_t$. During the second (main) phase, in iteration $t+i$, we determine z_{t+i}, D_i, and j_i. During both phases, we maintain a list of *eligible clauses* of F. Each item on the list is a triple $(D, j(D), y(D))$, where D is a clause that contains at least $k-1$ variables on the list so far, $j(D)$ is the least index for which $|v(D) \cap \{z_1, \ldots, z_{j(D)}\}| = k-1$, and $y(D)$ is the unique variable of $v(D) - \{z_1, \ldots, z_{j(D)}\}$. At iteration j, after adding a variable z_j, the eligible list is updated by identifying all clauses C_s of F that contain z_j and that contain exactly $k-2$ distinct variables from $z_1, \ldots, z_{j-1}$. For each such clause, let $j(C_s) = j$ and let $y(C_s)$ be the unique variable of C_s that does not appear in the list. Order the set of all such triples $(C_s, j, y(C_s))$ according to the index s and append this to the list of eligible clauses.

In the initialization phase, during iteration i, we choose z_i to be the ith variable of T according to the global variable ordering, and then we update the eligible list. During iteration $t+i$ of the main phase, if there are no eligible clauses, then the algorithm terminates. Otherwise, remove the first triple (D, j, y) from the eligible list and set $z_{t+i} = y$, $D_i = D$ and $j_i = j$ and update the eligible list.

It is easy to show by induction on i that $\{z_1, \ldots, z_{t+i}\} \subseteq \hat{T}$ and that at termination the list $\{z_1, \ldots, z_{t+u}\}$ is closed and hence is $\hat{T}$. The other properties of the output asserted above are similarly obvious.

We are now ready to state and prove the key lemma.

LEMMA 6.6. *For $k \geq 3$ there is a constant $c(k)$ such that the following holds. Let n, m, t, w be positive integers and let $\Delta = m/n > 1$. Suppose that $t \leq w \leq c(k)n/\Delta^{1/(k-2)}$. Let T be a set of variables of size t. Then for $F \sim \mathcal{F}_m^{k,n}$, the probability that (F, T) threatens at least w clauses is at most 2^{-w}.*

Proof. Fix n, m, w, t and T as in the hypothesis of the lemma. We view an arbitrary formula $F \sim \mathcal{F}_m^{k,n}$ as an ordered sequence $C_1, \ldots, C_m$ of clauses. Thus F is uniformly chosen from $(2^k\binom{n}{k})^m$ possible formulas. Say that F is *bad* if (F, T) threatens at least w clauses. We will upper bound the probability that F is bad by showing that each bad F can be uniquely "encoded" in such a way that the number of encodings is a 2^{-w} fraction of the number of formulas.

Now suppose that F is bad, and thus the number u of threatened clauses is at least w. In this case, will show how to encode F efficiently. Our encoding will first

give enough information in order to recover the first w clauses, $D_1, \ldots, D_w$, output by the above algorithm, and then we will code more directly the remaining clauses of F. The encoding of $D_1, \ldots, D_w$ refers to some of the output of the above algorithm on F and will include the following:

(i) The list $z_{t+1}, \ldots, z_{t+w}$.

(ii) The list $j_1, j_2, \ldots, j_w$.

(iii) For $i \in [w]$, the sequence $d_i(1) < \cdots < d_i(k-2)$, where $d_i(j)$ is the least index such that in $|v(D_i) \cap \{z_1, \ldots, z_{d_i(j)}\}| = j$.

(iv) For each $i \in [w]$, the vector $\alpha_i \in \{+,-\}^k$, where $\alpha_i(j)$ is the sign of the appearance of $z_{d_i(j)}$ in D_i.

It is clear that this information is enough to reconstruct $D_1, \ldots, D_w$. To reconstruct the remaining clauses of F, $C_1, \ldots, C_m$, along with their ordering, we need two more things:

(i) The list $h_1, h_2, \ldots, h_w$ of indices such that $D_i = C_{h_i}$.

(ii) The ordered list of clauses $(C_i : i \notin \{h_1, \ldots, h_w\})$.

Let us count the number of such encodings. There are at most m^w ways to choose $h_1, \ldots, h_w$. There are $(2^k\binom{n}{k})^{m-w}$ ways to choose the sequence of clauses that are not among the D_i. There are at most n^w ways to choose $z_{t+1}, \ldots, z_{t+w}$. Since the list $j_1, \ldots, j_w$ satisfies $j_1 \le \cdots \le j_w$, the number of ways to choose this sequence is at most $\binom{t+2w-1}{w} \le 2^{t+2w}$. For each i the number of ways to choose $d_i(1), \ldots, d_i(k-2)$ is at most $(t+w)^{k-2}$, and the number of ways to choose α_i is at most 2^k. Multiplying these together and dividing by $(2^k\binom{n}{k})^m$ we get that the probability that (F,T) threatens at least w clauses is at most

$$\frac{m^w n^w 2^{t+2w+kw}(t+w)^{(k-2)w}}{(2^k\binom{n}{k})^w} \le \frac{m^w k^w 2^{t+2w}(t+w)^{(k-2)w}}{n^{(k-1)w}} \le \left(\frac{8km(2w)^{k-2}}{n^{k-1}}\right)^w.$$

For some constant $c(k) > 0$, if $w \le c(k)n(n/m)^{1/(k-2)}$ the above expression is at most 2^{-w}. □

Before we use this to prove the theorem, we derive an immediate corollary which we hope will be useful in analyzing other DLL algorithms. It says that for almost all formulas F every path in any DLL tree for F contains not many more unit propagation nodes than it does branching nodes.

LEMMA 6.7. *For $k \ge 3$ there is a constant $c(k)$ such that the following holds. Let n, m, t be positive integers, $\epsilon > 0$ $w = \lceil (1+\epsilon)\max(t, \log_2\binom{n}{t})\rceil$, and let $\Delta = m/n > 1$. Suppose that $t \le w \le c(k)n/\Delta^{1/(k-2)}$. Then for $F \sim \mathcal{F}_m^{k,n}$, the probability that there is a partial assignment of size t that generates at least w unit clauses is $o(1)$ in t.*

Proof. There are only $\binom{n}{t}$ sets of variables T of size t that can be the underlying set of variables of the partial assignment. Furthermore, any unit clause generated by such a partial assignment must be a clause threatened by (F,T). By Lemma 6.6, any single set T threatens w clauses with probability only 2^{-w}. Summing over all possible sets T yields the desired result. □

THEOREM 6.8. *Let $k \ge 3$ and let n, m be integers and $\Delta = m/n$. Let A be any oblivious DLL algorithm for k-CNF formulas on n variables and suppose $F \sim \mathcal{F}_m^{k,n}$. If $m = \omega(n^{k/2})$, then with probability $1 - o(1)$ in n, the running time of A on F is $2^{\Omega(n/\Delta^{1/(k-2)})}$.*

Proof. Fix k and let n, m satisfy the hypotheses of the theorem. There are two cases to consider, when F is satisfiable and when F is not satisfiable. Since

$m = \omega(n^{k/2})$, the probability that F is satisfiable is exponentially small in n. Thus, we can assume without loss of generality that F is unsatisfiable in what follows, and, in particular, every leaf node in any DLL tree for F is labeled by 0. Observe that the conclusion of the theorem is trivial if $m = \Omega(n^{k-1})$, so we may assume that $m = o(n^{k-1})$. Fix an oblivious DLL algorithm and let λ be the associated labeling of B_n. Let $t = c(k)n/\Delta^{1/(k-2)}$, where $c(k)$ is as in Lemma 6.6. The hypothesis and the assumption $m = o(n^{k-1})$ imply $t = o(\sqrt{n})$ and $t = \omega(1)$.

For a formula F, let $R = R(F)$ be the set of vertices at level t in B_n that are visited by the traversal of B_n in the execution $A_\lambda(F)$. Note that $v \in R$ means that no clause of F becomes empty along the path to v. We will show that the probability that $|R| \leq 2^{t-1}$ is $o(1)$, and since $|R|$ is a lower bound on the running time of $A_\lambda(F)$ this will prove the theorem.

First, for a fixed vertex v at level t in B_n we upper bound $\Pr[v \notin R]$. Let T be the set of variables labeling the nodes in B_n on the path to v. By the definition of the algorithm and by Proposition 6.5 the variables set by unit propagation along the path to v must be in $\hat{T}$, and any empty clause must be a clause threatened by (F, T). Consider the output $D_1, \ldots, D_u$ and $z_1, \ldots, z_{t+u}$ of the algorithm $Close(F, T)$. We claim that if the variables $z_1, \ldots, z_{t+u}$ are all distinct, then no clause of F becomes empty along the path to v, and since F is unsatisfiable, this implies that $A_\lambda(F)$ reaches v. For this, it suffices to show that for $i \in [u]$, if the variable z_{t+i} is set by unit propagation, then it can only be set because D_i becomes a unit clause. So assume for contradiction that z_{t+i} is set because some clause D_j with $j \neq i$ becomes a unit clause and that this is the first time that this happens. However, then since $z_{t+j} \in v(D_j)$, it must have been set before D_j became a unit clause, which contradicts the choice of z_{t+i}.

Thus we have

$$\begin{aligned}\Pr[v \notin R] &\leq \Pr[z_1, \ldots, z_{t+u} \text{ are not distinct}] \\ &\leq \Pr[u > t] + \Pr[z_1, \ldots, z_{t+u} \text{ are not distinct} : u \leq t].\end{aligned}$$

Now $\Pr[u > t] \leq 2^{-t}$ by Lemma 6.6. We claim that the second term is bounded above by $3t^2/2n$. To see this, note that by the definition of the algorithm *Close* the triple (D_i, j_i, z_{t+i}) is the ith item placed on the eligible clause list. Think of this triple as a random variable (which depends on the random formula F). The key observation is that at the time that we add (D_i, j_i, z_{t+i}) to the eligible list (at the end of iteration j_i) the conditional distribution of z_{t+i} is given by the uniform distribution over the set $\{x_1, \ldots, x_n\} - \{z_1, \ldots, z_{j_i}\}$. Thus the probability that $z_{t+i} \notin \{z_1, \ldots, z_{t+i-1}\}$ is $1 - (t + i - 1 - j_i)/(n - j_i) \geq 1 - (t + i - 1)/n$, and the probability that there is some $i \in [t]$ for which $z_{t+i} \in \{z_1, \ldots, z_{t+i-1}\}$ is at most $\sum_{i=1}^{t}(t + i - 1)/n \leq \frac{3t^2}{2n}$, as claimed.

We conclude that $\Pr[v \notin R] \leq 2^{-t} + \frac{3t^2}{n}$, which is $o(1)$ in n since $t = o(\sqrt{n})$ and $t = \omega(1)$. Therefore, the expected number of vertices $v \notin R$ is at most $o(2^t)$, which under the assumption on t is $o(2^t)$. Thus, by Markov's inequality, the probability that more than 2^{t-1} vertices are not in R is $o(1)$ and thus $|R| \geq 2^{t-1}$ with probability $1 - o(1)$, as required to complete the proof. □

7. Related work and further research. The question of whether or not resolution is automatizable is still open. In particular, what is the fastest deterministic or probabilistic search algorithm for resolution? The best that is known is presented in section 3 which is essentially due to Clegg, Edmonds, and Impagliazzo. It can be

shown that this is the best algorithm in the tree-like case, but we know of no negative results of this kind for the general case.

A problem pertinent to our results in section 4 is to prove exponential lower bounds for the weak pigeonhole principle, $\neg PHP_n^m$, where the number of pigeons, m, is large (say n^3). Buss and Pitassi [BP97] showed that there exists a resolution refutation of $\neg PHP_n^m$ when $m \geq 2^{\sqrt{n}\log n}$ of size $2^{\sqrt{n}\log n}$. It has been conjectured that when m is polynomial in n, any resolution refutation of $\neg PHP_n^m$ requires superpolynomial size. Razborov has shown that such a lower bound would imply that proving superpolynomial circuit lower bounds for an explicit function in NP is independent of certain systems of bounded arithmetic.[2]

In a preliminary version of this paper ([BP96]) we asked:

> Can one show that for any 3-CNF formula f, if f has a polynomial-size resolution refutation, then f also has a resolution refutation with maximum clause size $\sqrt{n}$? Such a result would justify the simple and natural deterministic simulation of resolution whereby we exhaustively search for proofs of maximum clause length i, for increasing i.

Based on the Clegg–Edmonds–Impagliazzo algorithm, a version of this clause-width conjecture has recently been proven by Ben-Sasson and Wigderson [BSW99]—in particular that size S resolution refutations can be made $O(\sqrt{n \log S})$-bounded and size S DLL proofs can be converted to $O(\log S)$-bounded resolution refutations. As stated above, this simplifies the algorithm in section 3 and can be used to show that random k-CNF formulas with n variables with clause density $\Delta \geq \theta_k(n)$ require resolution proofs of size $2^{\Omega(n/\Delta^{4/(k-2)+\epsilon})}$ and DLL proofs of size $2^{\Omega(n/\Delta^{2/(k-2)+\epsilon})}$. This size lower bound for general resolution refutations provides a simpler smooth tradeoff between Δ and proof size than our results, but it does not improve the largest densities for which exponential lower bounds for resolution are known to hold almost certainly. The size lower bound for DLL proofs is not as large as our lower bounds in section 6.1 but applies to all DLL procedures, not just oblivious ones.

The problem posed at the end of section 5.2 is whether the hypothesis on s in Lemma 5.5 needed for property $B_\epsilon(s)$ can be weakened. As mentioned, such a weakening would strengthen the lower bounds on $\mathrm{res}(F)$ for random formulas. Furthermore, when combined with the Ben-Sasson–Wigderson approach for lower bounding DLL refutation size, this improvement would yield a *tight* $2^{\Omega(n/\Delta^{1/(k-2)+o(1)})}$ lower bound on $\mathrm{DLL}(F)$ for F a random formula.[3]

Examining the proof of Lemma 5.11 we see that a critical place to look for improvement is in the last step of the proof of Proposition 5.12, where we obtain an upper bound on $\Pr[Q(r,q)]$ by multiplying $\Pr[B(m,p) \geq q]$ by $\binom{n}{r}$, which corresponds to upper bounding the probability of the union of $\binom{n}{r}$ events by their sum. Can this union bound be refined by a more careful analysis?

The essence of the question is captured by a natural question about random hypergraphs. Given a collection $\mathcal{H}$ of subsets of $[n]$, say that a subset A of $[n]$ is γ-crowded by $\mathcal{H}$ if A contains at least γn members of $\mathcal{H}$. Let $s(\mathcal{H},\gamma)$ be the largest s for which there is no set of size s that is γ-crowded by $\mathcal{H}$. The question is, If $\mathcal{H}$ is a

[2] Following a proof of subexponential lower bounds for a restricted version of resolution [PR01], such lower bounds have indeed been proved for general resolution by Raz [Raz01a] and simplified and strengthened by Razborov [Raz01b].

[3] Ben-Sasson and Galesi [BSG01] have recently shown just such a lower bound by replacing the property $B_\epsilon(s)$ with a weaker expansion property of such formulas.

random collection consisting of Δn subsets of $[n]$ each of size k what is the best lower bound on $s(\mathcal{H}, \gamma)$ that holds with probability $1 - o(1)$? A calculation analogous to that in Lemma 5.11 (using the union bound) shows that $s(\mathcal{H}, \gamma) \geq \Omega(n/\Delta^{1/(k-2-\frac{1}{\gamma})})$, while it is also not hard to show that $s(\mathcal{H}, \gamma) \leq O(n/\Delta^{1/(k-2)})$. An argument that the upper bound is tight could almost certainly be adapted to give an affirmative answer to the problem at the end of section 5.2.

A final open problem is to produce a better algorithm for finding unsatisfiability proofs for random formulas. In particular, is there a polynomial-time algorithm that succeeds in finding a proof of unsatisfiability with high probability for random formulas with cn clauses for some $c > 0$? We are not aware of *any* algorithm that provably beats the very simple ones we analyzed in section 6, even if we consider more powerful search methods that are not resolution-based.[4]

REFERENCES

[AS00] D. Achlioptas and G. Sorkin, *Optimal myopic algorithms for random 3-SAT*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 590–600.

[ASE92] N. Alon, J. Spencer, and P. Erdös, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[AV79] D. Angluin and L. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[BFU93] A. Broder, A. Frieze, and E. Upfal, *On the satisfiability and maximum satisfiability of random 3-CNF formulas*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, 1993, ACM, New York, pp. 322–330.

[BKPS98] P. Beame, R. Karp, T. Pitassi, and M. Saks, *On the complexity of unsatisfiability of random k-CNF formulas*, in Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, ACM, New York, 1999, pp. 561–571.

[BP96] Paul W. Beame and T. Pitassi, *Simplified and improved resolution lower bounds*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 274–282.

[BP97] S. Buss and T. Pitassi, *Resolution and the weak pigeonhole principle*, in Computer Science Logic, Lecture Notes in Comput. Sci. 1414, Springer-Verlag, Berlin, 1998, pp. 149–156.

[BPR97] M. Bonet, T. Pitassi, and R. Raz, *Lower bounds for cutting planes proofs with small coefficients*, J. Symbolic Logic, 62 (1997), pp. 708–728.

[BSG01] E. Ben-Sasson and N. Galesi, *Space complexity of random formulae in resolution*, in Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity, Chicago, IL, 2001.

[BSW99] E. Ben-Sasson and A. Wigderson, *Short proofs are narrow—resolution made simple*, in Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, ACM, New York, 1999, pp. 517–526.

[BT88] S. Buss and G. Turán, *Resolution proofs of generalized pigeonhole principles*, Theoret. Comput. Sci., 62 (1988), pp. 311–317.

[CA96] J.M. Crawford and L.D. Auton, *Experimental results on the crossover point in random 3SAT*, Artificial Intelligence, 81 (1996), pp. 31–57.

[4] Incomplete methods using spectral analysis of graphs associated with these formulas have subsequently been shown to yield polynomial-time proofs of unsatisfiability almost certainly for formulas at much lower densities than those we prove for resolution-based algorithms [GK01, FG01], but the question remains open for constant density.

[CEI96] M. Clegg, J. Edmonds, and R. Impagliazzo, *Using the Gröbner basis algorithm to find proofs of unsatisfiability*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, ACM, New York, 1996, pp. 174–183.

[CF90] M.T. Chao and J. Franco, *Probabilistic analysis of a generalization of the unit-clause literal selection heuristics*, Inform. Sci., 51 (1990), pp. 289–314.

[CR77] S.A. Cook and R.A. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1977), pp. 36–50.

[CR92] V. Chvátal and B. Reed, *Mick gets some (the odds are on his side)*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 620–627.

[CS88] V. Chvátal and E. Szemerédi, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.

[DLL62] M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem proving*, Comm. ACM, 5 (1962), pp. 394–397.

[FG01] J. Friedman and A. Goerdt, *Recognizing more unsatisfiable random* 3*-SAT instances efficiently*, in Automata, Languages, and Programming: 28th International Colloquium, Lecture Notes in Comput. Sci. 2076, J. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., Springer-Verlag, Berlin, 2001, pp. 310–321.

[FP83] J. Franco and M. Paull, *Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem*, Discrete Appl. Math., 5 (1983), pp. 77–87.

[Fri99] E. Friedgut, *Sharp thresholds of graph properties, and the k-sat problem*, J. Amer. Math. Soc., 12 (1999), pp. 1017–1054.

[FS96] A. Frieze and S. Suen, *Analysis of two simple heuristics on a random instance of k-SAT*, J. Algorithms, 20 (1996), pp. 312–355.

[Fu95] X. Fu, *On the Complexity of Proof Systems*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1995.

[GK01] A. Goerdt and M. Krivelevich, *Efficient recognition of random unsatisfiable k-SAT instances by spectral methods*, in Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2010, Springer-Verlag, Berlin, 2001, pp. 294–304.

[Goe96] A. Goerdt, *A threshold for unsatisfiability*, J. Comput. System Sci., 53 (1996), pp. 469–486.

[GPFW97] J. Gu, P.W. Purdom, J. Franco, and B.J. Wah, *Algorithms for the satisfiability (SAT) problem: A survey*, in Satisfiability (SAT) Problem, AMS, Providence, RI, 1997, pp. 19–151.

[Hak85] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–305.

[JSV00] S. Janson, Y.C. Stamatiou, and M. Vamvakari, *Bounding the unsatisfiability threshold of random* 3*-SAT*, Random Structures Algorithms, 17 (2000), pp. 103–116.

[KKKS98] L.M. Kirousis, E. Kranakis, D. Krizanc, and Y.C. Stamatiou, *Approximating the unsatisfiability threshold of random formulas*, Random Structures Algorithms, 12 (1998), pp. 253–269.

[KS94] S. Kirkpatrick and B. Selman, *Critical behavior in the satisfiability of random formulas*, Science, 264 (1994), pp. 1297–1301.

[PR01] T. Pitassi and R. Raz, *Lower bounds for regular resolution proofs of the weak pigeonhole principle*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, Hersonissos, Crete, Greece, 2001, pp. 347–355.

[Raz01a] R. Raz, *Resolution Lower Bounds for the Weak Pigeonhole Principle*, Technical Report TR01-021, Electronic Colloquium on Computational Complexity, Universität Trier, Trier, Germany, 2001; also available online from http://www.eccc.uni-trier.de/eccc/.

[Raz01b] A.A. Razborov, *Improved Resolution Lower Bounds for the Weak Pigeonhole Principle*, Technical Report TR01-055, Electronic Colloquium on Computational Complexity, Universität Trier, Trier, Germany, 2001; also available online from http://www.eccc.uni-trier.de/eccc/.

[SML96] B. Selman, D. Mitchell, and H. Levesque, *Generating hard satisfiability problems*, Artificial Intelligence, 81 (1996), pp. 17–29.

[Urq87] A. Urquhart, *Hard examples for resolution*, J. ACM, 34 (1987), pp. 209–219.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1076–1089

A DECISION PROCEDURE FOR UNITARY LINEAR QUANTUM CELLULAR AUTOMATA*

CHRISTOPH DÜRR† AND MIKLOS SANTHA‡

Abstract. Linear quantum cellular automata were introduced recently as one of the models of quantum computing. A basic postulate of quantum mechanics imposes a strong constraint on any quantum machine: it has to be *unitary*; that is, its time evolution operator has to be a unitary transformation. In this paper we give an efficient algorithm to decide if a linear quantum cellular automaton is unitary. The complexity of the algorithm is $O(n^{(3r-1)/(r+1)}) = O(n^3)$ in the algebraic computational model if the automaton has a continuous neighborhood of size r, where n is the size of the input.

Key words. quantum computation, reversible cellular automata

AMS subject classifications. 81V99, 68Q80

PII. S0097539797327702

1. Introduction.

The classical models of computation, such as Turing machines, random access machines, circuits, or cellular automata, are all universal in the sense that they can simulate each other with only polynomial overhead. These models are based on classical physics, whereas physicists believe that the universe is better described by quantum mechanics.

Feynman [8, 9] pointed out first that there might be a substantial gap between computational models based on classical physics and those based on quantum mechanics. The *quantum Turing machine* (QTM), the first model of quantum computation, was introduced by Benioff [1, 2]. Deutsch in [5] described a universal simulator for QTMs with exponential overhead. Bernstein and Vazirani [3] were able to construct a universal QTM with only polynomial overhead.

Other quantum computational models were also studied recently. Deutsch [6] has defined the model of *quantum circuits*, and later Yao [19] has shown that QTMs working in polynomial time can be simulated by polynomial size quantum circuits. Physicists were also interested in *quantum cellular automata*: Biafore [4] considered the problem of synchronization, Margolus [14] described space-periodic quantum cellular automata, and Lloyd [12, 13] discussed the possibility of realizing a special type of quantum cellular automaton. *Linear quantum cellular automata* (LQCAs) were formally defined by Watrous [18] and by Dürr, LêThanh, and Santha [7]. In the former paper it was shown that a subclass of LQCAs, *partitioned linear quantum cellular automata* (PLQCAs), can be simulated by QTMs with linear slowdown. Van Dam [17] defined space-periodic LQCAs and gave a universal instance of this model.

We should make clear at this point that most of these models are only theoretically motivated. *Real life* quantum computers as built today in laboratories are essentially small quantum circuits or partitioned cellular automata.

*Received by the editors September 22, 1997; accepted for publication (in revised form) December 1, 1999; published electronically March 13, 2002. This research was supported in part by the ESPRIT Working Group RAND2.

http://www.siam.org/journals/sicomp/31-4/32770.html

†Université Paris-Sud, LRI, bât 490, F-91405 Orsay Cedex, France (durr@lri.fr). This author's research was supported in part by the ISI Foundation.

‡CNRS, URA 410, Université Paris-Sud, LRI, bât 490, F-91405 Orsay Cedex, France (santha@ lri. fr). This author's work was done in part while at the Centre de Recerca Matemàtica, Institut d'Estudis Catalans, Bellaterra, Spain.

A quantum computational device is at any moment of its computation in a *superposition of configurations*, where each configuration has an associated complex amplitude. A superposition is *valid* if it has unit norm. If the device is *observed*, then a configuration will be chosen at random, where the probability of a configuration to be chosen is equal to the squared magnitude of its amplitude. Therefore it is essential that valid superpositions be transformed into valid superpositions, or, equivalently, that the *time evolution operator* of the device preserves the norm. This property is called *well-formedness*, and thus it is a natural problem to decide if a given quantum machine is well-formed. In the case of QTMs and PLQCAs there exist easily checkable constraints on the finite local transition function of the machine which are equivalent to its well-formedness. Such constraints were identified, respectively, by Bernstein and Vazirani [3] and by Watrous [18]. In the case of LQCAs no such local constraints are known, still Dürr, LêThanh, and Santha [7] gave a polynomial time algorithm to decide if an LQCA is well-formed. Part of this algorithm was improved by Høyer [10] using a different approach.

However, one of the basic postulates of quantum mechanics imposes an even stronger constraint than norm-preserving on the time evolution operator. It actually requires that this operator—as any other quantum operator—be a unitary transformation. We will call a machine which satisfies this constraint *unitary*. In [3] and [18] it was proven that norm-preserving already implies unitarity in the case of QTMs and PLQCAs. It is also trivially true for machines with finite configuration sets such as quantum circuits. However, this is not true for LQCAs; it is quite simple to construct a well-formed LQCA which is not unitary.[1]

In this paper we give an efficient algorithm to decide if an LQCA is unitary. The complexity of our algorithm is cubic if the input LQCA has continuous neighborhood. (Most papers in the literature about classical linear cellular automata deal only with such cases.) Our algorithm will use the procedure of [7] which in quadratic time decides if the LQCA is well-formed. The present paper actually gives an algorithm which decides if a well-formed LQCA is also unitary.

Well-formedness is equivalent to the orthonormality of the column vectors of the time evolution operator; unitarity also requires orthonormality from the row vectors. Deciding unitarity is much harder than deciding well-formedness. One way of seeing this is that whereas the column vectors have finite support, the row vectors can have an infinite number of nonzero components.

2. The computational model. Let us fix for the paper the following notation. If u and v are vectors in some inner-product space over the complex or the real numbers, then $\langle u|v\rangle$ will denote the inner product of u and v, and $\|u\|$ the norm of u. If M is a matrix in such a space, then M^* denotes its conjugate transpose.

We recall here the definition of an LQCA which is the quantum generalization of the classical one-dimensional cellular automaton. A more detailed description of this model can be found in [18] and in [7].

An LQCA is a 4-tuple $A = (\Sigma, q, N, \delta)$. The *cells* of the automaton are organized in a line and are indexed by the elements of $\mathbb{Z}$. The finite, nonempty set Σ is the set of *(cell-)states*, and $q \in \Sigma$ is a distinguished *quiescent state*. The *neighborhood* $N = (a_1, \ldots, a_r)$ is a strictly increasing sequence of integers, for some $r \geq 1$, giving the addresses of the neighbors relative to each cell. This means that the neighbors of

[1] A well-known example is the classical LCA XOR= $(\{0,1\}, 0, (0,1), \delta)$, where $\delta(x,y) = (x + y) \bmod 2$. This cellular automaton is injective for finite configurations but not surjective; thus the associated LQCA is well-formed but not unitary.

cell i are indexed by $i+a_1, \ldots, i+a_r$. An automaton is *simple* if its neighborhood is an interval of integers, that is $a_r = a_1 + r - 1$. In this paper we deal only with simple automata, and we will explain only briefly in the conclusion how our results apply to the general case.

The states of the cells are changing simultaneously at every time step according to the *local transition function*. This is the mapping $\delta : \Sigma^{|N|} \to \mathbb{C}^\Sigma$, which satisfies that, for every $(x_1, \ldots, x_r) \in \Sigma^r$, there exists $y \in \Sigma$ such that $[\delta(x_1 \ldots x_r)](y) \neq 0$. If at some time step the neighbors of a cell are in states $x_1, \ldots, x_r$, then at the next step the cell will change into state y with amplitude $[\delta(x_1 \ldots x_r)](y)$ which is denoted by $\langle \delta(x_1 \ldots x_r)|y\rangle$. The quiescent state q satisfies that

$$\langle \delta(q^r)|y\rangle = \begin{cases} 1 & \text{if } y = q, \\ 0 & \text{if } y \neq q. \end{cases}$$

The set of *configurations* is $\Sigma^{\mathbb{Z}}$, where for every configuration c, and for every integer i, the state of the cell indexed by i is c_i. A configuration c is *finite* if its *support* $\{i : c_i \neq q\}$ is finite. We are dealing only with LQCAs which evolve on finite configurations. We will denote the set of finite configurations by $\mathcal{C}_A$, and from now on we use the word *configuration* to mean a *finite configuration*. For a configuration c, let $\mathrm{idom}(c)$ be the *interval domain* of c, which is the smallest integer interval containing the support of c. For the sake of definiteness, we define the empty interval as $[0, -1]$ which is the interval domain of the everywhere quiescent configuration.

The local transition function induces the *time evolution operator* which we write in matrix form $U_A : \mathcal{C}_A \times \mathcal{C}_A \to \mathbb{C}$, where $U_A(d, c)$ is the transition amplitude of changing configuration c to configuration d in one step. It is defined by

$$U_A(d, c) = \prod_{i \in \mathbb{Z}} \langle \delta(c_{i+N})|d_i\rangle,$$

where $\delta(c_{i+N})$ is a short notation for $\delta(c_{i+a_1}, \ldots, c_{i+a_r})$. This product is well-defined since c has finite support.

The automaton evolves on *superpositions of configurations* which are elements of the Hilbert space $\ell_2(\mathcal{C}_A)$. If at some time step the automaton is in the superposition $u \in \mathbb{C}^{\mathcal{C}_A}$, then at the next time step it will be in the superposition $U_A u$. Therefore U_A is also an operator on $\ell_2(\mathcal{C}_A)$. A is *well-formed* if U_A is norm-preserving, and we say that it is *unitary* if U_A is a unitary transformation.

We will work in the algebraic computational model where complex numbers take unit space and arithmetic and logical operations take unit time. The description size of an automaton is clearly dominated by the local transition table δ. Therefore we define the *size* of the automaton to be $n = |\Sigma^{r+1}|$.

For the rest of the paper we will fix a well-formed simple LQCA $A = (\Sigma, q, N, \delta)$. Without loss of generality we assume $N = (0, 1, \ldots, r-1)$. Indeed, let $A' = (\Sigma, q, N', \delta)$ be the well-formed simple LQCA with the general neighborhood $N' = (j, \ldots, j + r - 1)$ for some integer j. We claim that A is unitary if and only if A' is unitary. Let $A'' = (\Sigma, q, (j), \delta')$ be the *shift cellular automaton*, where δ' is the identity. A'' is unitary since δ' is unitary. Moreover, $U_A = U_{A''} U_{A'}$ which proves the claim.

2.1. Example. The figures in this paper will illustrate our algorithm with the following LQCA: QFLIP= $(\{a, b\}, a, (0, 1), \delta)$ with $\langle \delta(x, y)|z\rangle$ defined for all $x, y, z \in$

$\{a, b\}$ by the following table:

$xy\backslash z$	a	b
aa	1	0
ba	0	1
ab	$1/\sqrt{2}$	$1/\sqrt{2}$
bb	$1/\sqrt{2}$	$-1/\sqrt{2}$

Using the algorithm in [7] it can be shown that QFLIP is well-formed, and in this article we show that its evolution operator U_{QFLIP} is even unitary. This LQCA has an interesting property. For $n \geq 0$, let c^n be the configuration which is b in all cells of index $i \in [-n, -1]$ and is a elsewhere. Then we have for all $n \geq 1$, $U_{\text{QFLIP}}(c^1, c^n) = (1/\sqrt{2})^n$. Thus an infinite number of configurations lead with nonzero amplitude to the single configuration c^1.

For the all quiescent configuration c^0, we have $U_{\text{QFLIP}}(c^0, c^0) = 1$; thus there is a unique configuration leading to it. Therefore when the LQCA QFLIP runs backwards in time, every cell with index $i \leq -1$ depends on cell -1. From this we conclude that there cannot be a LQCA with finite neighborhood whose evolution operator is exactly U^*_{QFLIP}.

This makes the model of LQCAs different from QTMs, since for every well-formed QTM M, there exists a QTM which runs M backward in time with a constant time overhead. It explains a bit why it seems difficult to simulate any LQCA by a QTM.

3. The main result. The main result of the paper is the following theorem.

THEOREM 1. *There exists an algorithm which takes a simple LQCA as input and decides in time* $O(n^{\frac{3r-1}{r+1}}) = O(n^3)$ *if it is unitary.*

Since in [7] a $O(n^2)$ algorithm is given to decide if an LQCA is well-formed, we will give only an algorithm which decides if a well-formed LQCA is unitary. The following lemma states that we have to verify only that the rows of the time evolution operator are of unit norm.

LEMMA 2. *Let* $U \in \mathbb{C}^{\mathcal{C}_A \times \mathcal{C}_A}$ *be a linear operator. If* U *is norm-preserving, then its rows have norm at most* 1. *If all the rows are of unit norm, then* U *is unitary.*

Proof. Let c be a configuration, and let $\mathbf{c}$ be the superposition which has amplitude 1 for c and 0 elsewhere. Then the norm of the row indexed by c in U is $\|U^*\mathbf{c}\|$. Since U is norm-preserving $\|U^*\mathbf{c}\| = \|UU^*\mathbf{c}\|$ and the projection of $UU^*\mathbf{c}$ on $\mathbf{c}$ has norm

$$|\langle \mathbf{c}|UU^*\mathbf{c}\rangle| = |\langle U^*\mathbf{c}|U^*\mathbf{c}\rangle| = \|U^*\mathbf{c}\|^2.$$

However, the projection of $UU^*\mathbf{c}$ on a unit vector has norm at most $\|U^*\mathbf{c}\|$, and therefore $\|U^*\mathbf{c}\| \leq 1$.

For the second part of the lemma observe that the projection of $UU^*\mathbf{c}$ on $\mathbf{c}$ has norm 1. Since U is norm-preserving the projection on any other basis vector $\mathbf{c}'$ must be 0. Thus $\langle \mathbf{c}'|UU^*\mathbf{c}\rangle$ is 1 if $c = c'$ and 0 otherwise, or in other words $UU^* = I$, which concludes the proof. □

The outline of the proof is the following. First we give a sequence of reduction steps in section 4 to a graph theoretical problem and to another one from linear algebra. We then give an algorithm to solve the problem. The different steps of the algorithm are presented in sections 5, 6, and 7. The proof of the main theorem is then summarized in section 8.

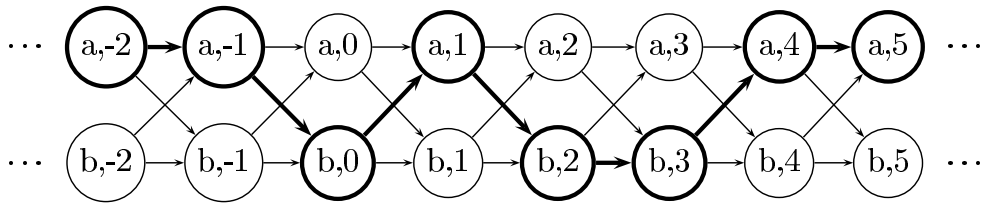


FIG. 1. *The configuration graph of the LQCA* QFLIP. *The bold path corresponds to the configuration* $\dots aababbaa \dots$.

4. The reduction. The different reduction steps are illustrated in Figures 1–3.

Our problem is the following. We have to decide if all row vectors of the evolution operator associated to a given LQCA have unit norm, under the assumption that the operator is norm-preserving. The naive method fails because one would have to compute the norm for an infinite number of rows. Moreover, for every row there can be an infinite number of nonzero entries and every entry is defined by a product on an unbounded number of terms. The purpose of this section is to reduce our problem to a finite one.

The *configuration graph* is the infinite directed graph $G_\infty(V, E)$ defined by $V = \Sigma^{r-1} \times \mathbb{Z}$ and $E = \{((xt, i), (ty, i+1)) : x, y \in \Sigma, t \in \Sigma^{r-2}, i \in \mathbb{Z}\}$. To our knowledge this type of graph was first used by Sutner and Maas [16] to show that a particular robot motion planning problem in the presence of moving obstacles is PSPACE-hard. It was used again by Sutner [15] to prove that the predecessor of every recursive configuration is also recursive.

A nonempty sequence (possibly infinite to the left, to the right, or in both directions) of vertices $(\dots, (w_i, i), \dots)$ in G_∞ is a *path* if and only if for at most a finite number of indices i we have $w_i \neq q^{r-1}$ and there is an edge between every two immediate vertices. Note that a sequence with a single vertex is already a path. We denote by F, L, R, and P, respectively, the set of paths which are finite, infinite to the left, infinite to the right, and infinite to both directions. Figure 1 illustrates a path of P.

We say that two paths p_1 and p_2 are *compatible* if the last vertex of p_1 and the first vertex of p_2 exist and they are the same. In that case the *composition* $p_1 \otimes p_2$ is the concatenation of the two sequences after identifying the extreme vertices. If P_1 and P_2 are sets of paths, then

$$P_1 \otimes P_2 = \left\{ p_1 \otimes p_2 : \begin{array}{l} p_1 \in P_1, p_2 \in P_2, \\ p_1 \text{ and } p_2 \text{ are compatible} \end{array} \right\}.$$

Let d be an arbitrary configuration. It induces a *weight* function g_d for the edges of G_∞, where $g_d((xt, i), (ty, i+1)) = |\langle \delta(xty) | d_i \rangle|^2$. We extend the weight function g_d to paths and to sets of paths. The weight of a path is the product of the respective edge weights, and the weight of a path set is the sum of the respective path weights. The weight of a path consisting of a single vertex is 1, and the weight of the empty path set is 0. We denote this *weighted configuration graph* by G_∞^d which is illustrated in Figure 2.

Although the weight of an infinite path is an infinite product, it is well-defined since all but a finite number of edges have weight 1. The following lemma establishes

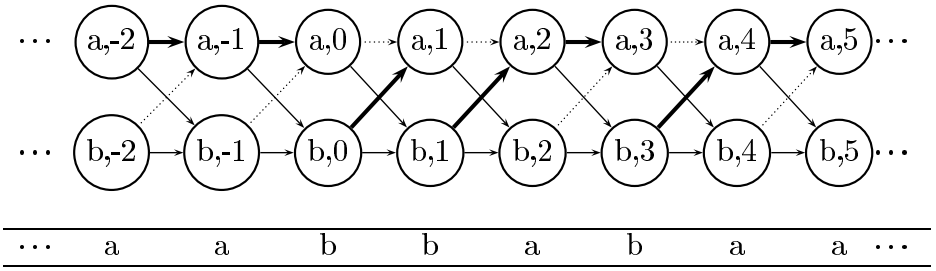


FIG. 2. *The configuration graph of the LQCA* QFLIP *weighted by the configuration* $d = \ldots aabbabaa \ldots$. *The bold edges have weight* 1, *the normal edges have weight* 1/2, *and the dotted-line edges have weight* 0.

a strong relationship between the weight of an infinite path in G^d_∞ and the entries of the time evolution matrix.

LEMMA 3. *For any configuration* d *the row indexed by* d *in* U_A *has norm* $\sqrt{g_d(P)}$.

Proof. We will show that there is a bijection h between the set of configurations $\mathcal{C}_A$ and the set of infinite paths P in G^d_∞ such that for every configuration c and d

$$g_d(h(c)) = |U_A(d,c)|^2.$$

Summing up over all configurations c will immediately conclude the lemma.

Let $h : \mathcal{C}_A \to P$ be defined for all configuration c by

$$h(c) = (\ldots, (c_i \ldots c_{i+r-2}, i), \ldots).$$

Then it is a bijection, and the following equalities conclude the proof:

$$\begin{aligned} g_d(h(c)) &= \prod_{i\in\mathbb{Z}} g_d((c_i \ldots c_{i+r-2}, i), (c_{i+1} \ldots c_{i+r-1}, i+1)) \\ &= \prod_{i\in\mathbb{Z}} |[\delta(c_{i+N})](d_i)|^2 \\ &= \left| \prod_{i\in\mathbb{Z}} [\delta(c_{i+N})](d_i) \right|^2 \\ &= |U_A(d,c)|^2. \qquad \square \end{aligned}$$

With Lemma 2 we got the following reduction of our problem.

COROLLARY 4. *In a well-formed LQCA, for every configuration* d, *we have* $g_d(P) \le 1$. *Moreover, the automaton is unitary if and only if for every* d, $g_d(P) = 1$.

Let us fix an interval $[j,k]$ and a configuration d with $\mathrm{idom}(d) \subseteq [j,k]$. This interval induces a subset of the path sets L, F, and R. For every $w, w' \in \Sigma^{r-1}$ we set

$$\begin{aligned} L^j_w &= \{p \in L : \text{the last vertex of } p \text{ is } (w,j)\}, \\ F^{j,k}_{w,w'} &= \left\{ p \in F : \begin{array}{l} \text{first vertex of } p \text{ is } (w,j), \\ \text{last vertex of } p \text{ is } (w',k+1) \end{array} \right\}, \\ R^k_{w'} &= \{p \in R : \text{the first vertex of } p \text{ is } (w',k+1)\}. \end{aligned}$$

Since the set of infinite paths can be decomposed as

$$P = \bigcup_{w,w' \in \Sigma^{r-1}} L_w^j \otimes F_{w,w'}^{j,k} \otimes R_{w'}^k,$$

we have

$$g_d(P) = \sum_{w,w' \in \Sigma^{r-1}} g_d(L_w^j) \cdot g_d(F_{w,w'}^{j,k}) \cdot g_d(R_{w'}^k).$$

The following lemma shows that $g_d(L_w^j)$ and $g_d(R_w^k)$ are independent from j, k, and d.

LEMMA 5. *For any intervals $[j,k]$, $[j',k']$ and configurations d, d' such that* $\mathrm{idom}(d) \subseteq [j,k]$, $\mathrm{idom}(d') \subseteq [j',k']$ *and for every $w \in \Sigma^{r-1}$, we have*

$$g_d(L_w^j) = g_{d'}(L_w^{j'}) \quad \text{and} \quad g_d(R_w^k) = g_{d'}(R_w^{k'}).$$

Proof. We prove only the first equation; the proof for the second one is analogous. Let $m = j' - j$. We define a bijection from L_w^j to $L_w^{j'}$ which preserves the weight. If $p = (\dots, (w_i, i), \dots, (w_j, j))$ is a path in L_w^j, then we define its image as $p' = (\dots, (w_i, i+m), \dots, (w_j, j+m))$. This is clearly a bijection and we also have $g_d(p) = g_{d'}(p')$ since $d_i = q$ for $i < j$ and $d'_i = q$ for $i < j'$. ∎

We define the *left* and *right border vectors*, respectively, $\vec{l} = (l_w)_{w \in \Sigma^{r-1}}$ and $\vec{r} = (r_w)_{w \in \Sigma^{r-1}}$ as follows: for $w \in \Sigma^{r-1}$, $l_w = g_d(L_w^j)$ and $r_w = g_d(R_w^k)$, where $[j,k]$ is an arbitrary interval and d is an arbitrary configuration satisfying $\mathrm{idom}(d) \subseteq [j,k]$. The next lemma states that $\vec{l}$ and $\vec{r}$ are in $\mathbb{R}^{\Sigma^{r-1}}$.

LEMMA 6. *For all $w \in \Sigma^{r-1}$, l_w and r_w are finite.*

Proof. Suppose there is a w such that $l_w = \infty$. (The case $r_w = \infty$ is symmetric.) We will prove that this implies the existence of a configuration such that the associated line vector has infinite norm, thus contradicting by Corollary 4 the hypothesis that A is well-formed.

Let w' be such that $r_{w'} > 0$. There exists such a w', since, for example, $r_{q^{r-1}} \geq 1$. Let $x_1, x_2, \dots, x_{2r-1} \in \Sigma$ such that $w = x_1 \dots x_{r-1}$ and $w' = x_r \dots x_{2r-2}$. We set $w_i = x_i x_{i+1} \dots x_{i+r-2}$ for $i = 1, \dots, r$ and $v_i = x_i x_{i+1} \dots x_{i+r-1}$ for $i = 1, \dots, r-1$. Note that $w_1 = w$ and $w_r = w'$. For $i = 1, \dots, r-1$ let $y_i \in \Sigma$ be such that $\langle \delta(v_i) | y_i \rangle \neq 0$. Let j be an arbitrary integer and set $k = j + r - 2$. We define the configuration d to be the quiescent state outside the interval $[j,k]$ and $d_{j+i-1} = y_i$ for $i = 1, \dots, r-1$. Then in G_∞^d the set of paths going through the vertices $(w_1, j), \dots, (w_r, k+1)$ already has infinite weight. Since each path has nonnegative weight, P also has infinite weight. ∎

The first part of our algorithm will be the computation of the border vectors $\vec{l}$ and $\vec{r}$. For the second part, we reduce now our problem to a question in linear algebra.

For every $a \in \Sigma$, let $M_a \in \mathbb{R}^{\Sigma^{r-1} \times \Sigma^{r-1}}$ be the linear operator whose matrix is defined for all $w, w' \in \Sigma^{r-1}$ as

$$M_a(w', w) = \begin{cases} |\langle \delta(xty) | a \rangle|^2 & \text{if } w = xt, w' = ty \text{ for some } x, y \in \Sigma \text{ and } t \in \Sigma^{r-2}, \\ 0 & \text{otherwise.} \end{cases}$$

We extend this definition to finite sequences over Σ. If ϵ denotes the empty word, then M_ϵ is the identity operator. Let $s > 1$ be an integer, and let $b = b_1 \dots b_s$ be an element of Σ^s. We define

$$M_b = M_{b_s} \cdots M_{b_1}.$$

LEMMA 7. *Let d be a configuration with* $\mathrm{idom}(d) = [j, k]$. *Then*

$$g_d(P) = \langle M_{d_j \dots d_k} \vec{l} | \vec{r} \rangle.$$

Proof. We have

$$\begin{aligned} g_d(P) &= \sum_{w,w' \in \Sigma^{r-1}} g_d(L^j_w) \cdot g_d(F^{j,k}_{w,w'}) \cdot g_d(R^k_{w'}) \\ &= \sum_{w,w'} l_w \cdot g_d(F^{j,k}_{w,w'}) \cdot r_{w'} \\ &= \sum_{w,w'} l_w \cdot M_{d_j \dots d_k}(w', w) \cdot r_{w'} \\ &= \sum_{w'} \left(\sum_{w} l_w \cdot M_{d_j \dots d_k}(w', w) \right) \cdot r_{w'} \\ &= \sum_{w'} (M_{d_j \dots d_k} \vec{l})(w') \cdot r_{w'} \\ &= \langle M_{d_j \dots d_k} \vec{l} | \vec{r} \rangle. \qquad \square \end{aligned}$$

Since for every $b \in \Sigma^*$ there exists a configuration d whose nonquiescent part is b, Corollary 4 and Lemma 7 imply the following reduction.

COROLLARY 8. *A well-formed LQCA is unitary if and only if for every $b \in \Sigma^*$ we have $\langle M_b \vec{l} | \vec{r} \rangle = 1$.*

We also have the following property, which simplifies our next reduction step.

LEMMA 9. *For every well-formed LQCA A we have $\langle \vec{l} | \vec{r} \rangle = 1$.*

Proof. Let d be the all quiescent configuration. Then in U_A the column indexed by d has the entry 1 at row d and 0 elsewhere. Since the column vectors of U_A are pairwise orthogonal, Lemma 2 implies that the row indexed by d has only zero entries besides column d. Therefore this row has norm 1, and the claim follows from Lemmas 3 and 7. $\square$

Let $m = |\Sigma^{r-1}|$. The border vectors can be seen as elements of $\mathbb{R}^m$, and the elements of the set $\mathcal{M} = \{M_a : a \in \Sigma\}$ can also be seen as linear transformations in $\mathbb{R}^m$. Let us fix a few notations for the inner-product space $\mathbb{R}^m$. Let $S \subseteq \mathbb{R}^m$ be a finite set of vectors. The *linear subspace* and the *affine subspace* generated by S, denoted here, respectively, by $\langle S \rangle$ and $[S]$, are defined as

$$\begin{aligned} \langle S \rangle &= \{\lambda_1 \vec{s}_1 + \cdots + \lambda_t \vec{s}_t | \; t \geq 0; \vec{s}_1, \dots, \vec{s}_t \in S; \lambda_1, \dots, \lambda_t \in \mathbb{R}\}, \\ [S] &= \{\lambda_1 \vec{s}_1 + \cdots + \lambda_t \vec{s}_t | \; t \geq 0; \vec{s}_1, \dots, \vec{s}_t \in S; \lambda_1, \dots, \lambda_t \in \mathbb{R}; \lambda_1 + \cdots + \lambda_t = 1\}. \end{aligned}$$

B is said to be a basis of $\langle S \rangle$ (respectively, of $[S]$) if $\langle B \rangle = \langle S \rangle$ (respectively, $[B] = [S]$) and has minimal cardinality for this property.

Let $\vec{u} \in \mathbb{R}^m$ be a vector and $\mathcal{F} \subseteq \mathbb{R}^{m \times m}$ be a finite family of linear transformations. We set $S + \vec{u} = \{\vec{v} + \vec{u} : \vec{v} \in S\}$, and $\mathcal{F}(S) = \{f(\vec{v}) : \vec{v} \in S, f \in \mathcal{F}\}$. Let $H_{\vec{u}}$ denote the linear subspace whose normal vector is $\vec{u}$, that is, $H_{\vec{u}} = \{\vec{v} : \langle \vec{v} | \vec{u} \rangle = 0\}$.

We define by induction on i, for $i \geq 0$, the sets $\mathcal{F}^i(S)$. Let $\mathcal{F}^0(S) = S$, and $\mathcal{F}^{i+1}(S) = \mathcal{F}^i(S) \cup \mathcal{F}(\mathcal{F}^i(S))$. We say that S is *closed* for $\vec{v}$ under $\mathcal{F}$ if $\bigcup_{i=0}^{\infty} \mathcal{F}^i(\{\vec{v}\}) \subseteq S$.

From Lemma 9, $H_{\vec{r}} + \vec{l}$ is the set of vectors which have unit inner product with $\vec{r}$, that is,

$$H_{\vec{r}} + \vec{l} = \{\vec{u} : \langle \vec{u} | \vec{r} \rangle = 1\}.$$

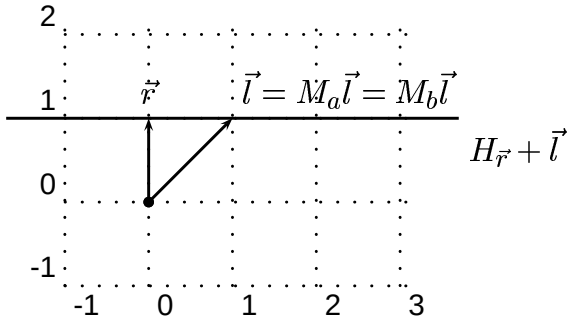


FIG. 3. *For the LQCA* QFLIP*, the border vectors* $\vec{l}$ *and* $\vec{r}$ *and the affine subspace* $H_{\vec{r}} + \vec{l}$*. In this example* $\vec{l}$ *is a fixpoint for the operators* M_a *and* M_b*, which shows that* U_{QFLIP} *is unitary.*

It is an affine subspace since $H_{\vec{r}} + \vec{l} = [H_{\vec{r}} + \vec{l}]$. Clearly for every $b \in \Sigma^*$, $\langle M_b\vec{l}|\vec{r}\rangle = 1$ if and only if $H_{\vec{r}} + \vec{l}$ is closed for $\vec{l}$ under $\mathcal{M}$. Therefore by Corollary 8 our reduction steps lead to the following theorem.

THEOREM 10. *A well-formed LQCA is unitary if and only if* $H_{\vec{r}} + \vec{l}$ *is closed for* $\vec{l}$ *under* $\mathcal{M}$*.*

This characterization is illustrated in Figure 3.

5. Computing the border vectors. In this section we will give an algorithm for computing the border vectors. By symmetry, it will be sufficient to give it only for the left vector. The main tool in the computation will be the weighted border graph. Its underlying graph can be seen as a slight modification of the finite version of the configuration graph. This graph was also used in [7] for checking that all the columns of U_A had unit norms. However, there the weights were defined as the norms of the transition state superpositions, whereas here they will be the squared magnitudes of the amplitude of the quiescent state in those superpositions.

The *(left) border graph* is the finite, directed, weighted graph $G_l = (V, E, g)$. The vertex set is $V = \Sigma^{r-1}$ and the edge set is

$$E = \{(xt, ty) : x, y \in \Sigma, t \in \Sigma^{r-2}\}.$$

The weight function is defined as

$$g((xt, ty)) = |\langle \delta(xty)|q\rangle|^2.$$

A *path* in G_l is a finite, nonempty sequence of at least two vertices such that there is an edge between every two consecutive vertices. Observe that a single vertex alone here does not form a path. As usual, the weight of a path is the product of the edge weights, and the weight of a set of paths is the sum of the individual path weights. The weight of the empty path set is 0.

For every $w \in \Sigma^{r-1}$, we define P_w as the set of paths in G_l whose first vertex is q^{r-1}, whose second vertex is different from q^{r-1}, and whose last vertex is w.

LEMMA 11. *For every* $w \in \Sigma^{r-1}$*, we have*

$$l_w = \begin{cases} g(P_w) & \text{if } w \neq q^{r-1}, \\ g(P_w) + 1 & \text{if } w = q^{r-1}. \end{cases}$$

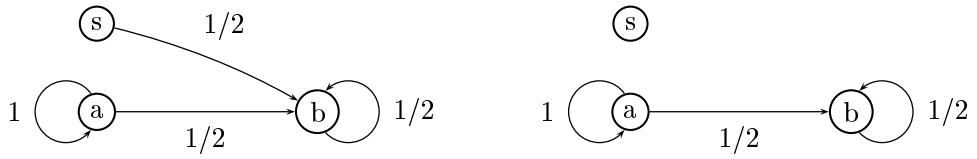


FIG. 4. *The graphs G'_l (left-hand) and G'_r (right-hand) associated to the LQCA* QFLIP. *From these graphs we can compute $\vec{l} = \binom{1}{1}$ and $\vec{r} = \binom{1}{0}$.*

Proof. Let d be a configuration with interval domain $[j, k]$, and let

$$p_q = (\dots, (q^{r-1}, i), \dots, (q^{r-1}, j)).$$

We set $L'_w = L^j_w - \{p_q\}$. We will give a weight-preserving bijection from L'_w to P_w which maps p to p'. Let $p = (\dots, (w_i, i), \dots, (w_j, j))$ be an element of L'_w, where $w_j = w$. Let $h \leq j$ be the greatest integer such that for every $i \leq h$ we have $w_i = q^{r-1}$. Then we set $p' = (q^{r-1}, w_{h+1}, \dots, w_j)$. This is clearly an injective mapping, and it is also surjective since by the choice of h, $w_{h+1} \neq q^{r-1}$. It is also weight-preserving since the edges in p until the vertex (w_h, h) all have weight 1. Since $g_d(p_q) = 1$ the lemma follows. ∎

THEOREM 12. *There exists an algorithm which computes the border vectors in time $O(n^{\frac{3(r-1)}{r+1}})$.*

Proof. According to Lemma 11 it is sufficient to compute $g(P_w)$ for $w \in \Sigma^{r-1}$. The main difficulty in this computation is that the paths of P_w are defined by a constraint which forces the second vertex to be different from q^{r-1}. The solution we propose codes this constraint directly in the graph, which we will augment by one vertex for this purpose. Then we compute the total path weight from i to j for all vertices i, j. To do this, we will adapt a standard algorithm which constructs the regular expression associated to a finite state automaton.

Let $G'_l = (V', E', g')$, where $V' = V \cup \{sq^{r-2}\}$ for a letter $s \notin \Sigma$,

$$E' = E \cup \{(sq^{r-2}, q^{r-2}y) : y \in \Sigma \backslash \{q\}\},$$

and $g'(e) = g(e)$ for all edges $e \in E$ and $g'((sq^{r-2}, q^{r-2}y)) = g((q^{r-1}, q^{r-2}y))$. This graph is illustrated in Figure 4. For every $w \in \Sigma^{r-1}$, let P'_w be the set of all paths in G'_l from sq^{r-2} to w. Clearly there is a weight-preserving bijection between P'_w and P_w.

The border vectors have only finite components; nevertheless, for their computation we have to extend the nonnegative real numbers with ∞. Let $\mathbb{R}^*$ be this set. We define the following computation rules with respect to ∞:

$$\infty + c = c + \infty = \infty,$$

for every $c \in \mathbb{R}^*$,

$$\infty \cdot c = c \cdot \infty = \infty \cdot \infty = \infty,$$

for every real number $c > 0$,

$$\infty \cdot 0 = \infty \cdot 0 = 0,$$

and $\infty^0 = 1$. We also define c^* for every $c \in \mathbb{R}^*$ as $\sum_{e'=0}^{\infty} c^{e'}$, that is,

$$c^* = \begin{cases} 1/(1-c) & \text{if } 0 \le c < 1, \\ \infty & \text{otherwise.} \end{cases}$$

Let $\{v_1, v_2, \ldots, v_{|V'|}\}$ be an arbitrary enumeration of the vertices of G'_l. For $1 \le i, j \le |V'|$ and for $0 \le k \le |V'|$, we define the path sets $P_k(i,j)$ as the set of paths which start in v_i, end in v_j, and all the other vertices in the path have indices less than or equal to k. Let $W_k(i,j)$ denote $g(P_k(i,j))$. Then we claim that $W_k(i,j)$ satisfies the following recursion for $1 \le i, j \le |V'|$ and $1 \le k \le |V'|$:

$$W_0(i,j) = \begin{cases} g'((v_i, v_j)) & \text{if } (v_i, v_j) \in E', \\ 0 & \text{otherwise,} \end{cases}$$
$$W_k(i,j) = W_{k-1}(i,j) + W_{k-1}(i,k) \cdot (W_{k-1}(k,k))^* \cdot W_{k-1}(k,j).$$

We prove our claim by induction on k. In $P_0(i,j)$ the only path is the edge between v_i and v_j if this edge exists.

Assume that this equation is true for $k-1$. We note that for every path of $P_k(i,j)$ there exists a unique integer e such that vertex v_k appears exactly e times the path. Thus we can write

$$P_k(i,j) = P_{k-1}(i,j) \cup \bigcup_{e=1}^{\infty} P_{k-1}(i,k) \otimes \underbrace{P_{k-1}(k,k) \otimes \cdots \otimes P_{k-1}(k,k)}_{e-1} \otimes P_{k-1}(k,j),$$

where the unions are disjoint and $\otimes$ is the path composition operator defined in section 4. By induction hypothesis we have

$$W_k(i,j) = W_{k-1}(i,j) + \sum_{e'=0}^{\infty} \left(W_{k-1}(i,k) \cdot (W_{k-1}(k,k))^{e'} \cdot W_{k-1}(k,j) \right),$$

which concludes the induction.

This proves the correction of the following algorithm: Let $m = |V'| = |\Sigma|^{r-1}$. Initialize W_0. For $k = 1, \ldots, m$ compute W_k using W_{k-1}. Finally, output the border vector $\vec{l}$ defined by $\vec{l}(w) = W_m(sq^{r-2}, w)$ for $w \neq q^{r-1}$ and $\vec{l}(q^{r-1}) = W_m(sq^{r-2}, q^{r-1})+1$. Proceed in similar fashion for $\vec{r}$. The complexity of the algorithm is $O(|\Sigma|^{3(r-1)}) = O(n^{\frac{3(r-1)}{r+1}})$. □

6. Closed affine subspace. In this section we will give a polynomial algorithm for the following problem.

CLOSED AFFINE SUBSPACE

Input: Two vectors $\vec{l}, \vec{r} \in \mathbb{R}^m$ such that $\langle \vec{l} | \vec{r} \rangle = 1$ and a set of linear transformations $\mathcal{M} = \{M_a : a \in \Sigma\}$ in $\mathbb{R}^m$, where $m = |\Sigma|^{r-1}$.

Question: Is $H_{\vec{r}} + \vec{l}$ closed for $\vec{l}$ under $\mathcal{M}$, i.e., for all $b \in \Sigma^*$ do we have $M_b \vec{l} \in H_{\vec{r}} + \vec{l}$?

We set $t = |\Sigma|$. For the simplicity of notation, let $H = H_{\vec{r}} + \vec{l}$ and let $E_i = \mathcal{M}^i(\{\vec{l}\})$. Since $H = [H]$, we have $E_i \subseteq H$ if and only if $[E_i] \subseteq H$. Therefore we have to decide if $[\bigcup_{i=0}^{\infty} E_i] \subseteq H$. Dimension arguments imply the existence of a fixpoint, a set E_j, such that $[E_j] = [\bigcup_{i=0}^{\infty} E_i]$. Moreover, we need only to keep track of a basis of $[E_i]$, that is, a set B_i of linearly independent vectors, with $[B_i] = [E_i]$.

THEOREM 13. *There exists an algorithm which decides if H is closed for $\vec{l}$ under $\mathcal{M}$ in time $O(n^{\frac{3r-2}{r+1}})$.*

Proof. We claim that this is realized by the following algorithm:

$B_0 := \{\vec{l}\}$
$i \quad := 1$
while $[B_i] \neq [B_i \cup \mathcal{M}(B_i)]$
$\quad B_{i+1} :=$ a basis of $[B_i \cup \mathcal{M}(B_i)]$
$\quad i := i + 1$
$B := B_i$
if $B \subseteq H$ accept
else reject

At every iteration $\dim([B_i])$ increases, and therefore the algorithm terminates in at most $m - 1$ iterations. We prove that it is correct.

We show $[B_i] = [E_i]$ for every $i \geq 0$ by induction. The statement holds by definition for $i = 0$. Suppose $[E_i] = [B_i]$ for some i. Since $\mathcal{M}$ contains only linear operators, we have for any set S, $[\mathcal{M}(S)] = [\mathcal{M}([S])]$. Therefore $[\mathcal{M}(E_i)] = [\mathcal{M}(B_i)]$ which implies $[E_{i+1}] = [B_{i+1}]$.

Since B is a fixpoint, that is, $[B] = [B \cup \mathcal{M}(B)]$, this implies that $[B] = [\bigcup_{i=0}^{\infty} E_i]$ and the correctness follows.

We now turn to the analysis of the complexity. We will build inductively the basis so that for all i, $B_i \subseteq B_{i+1}$. At the ith iteration, to build B_{i+1}, initially we set $B_{i+1} = B_i$. Then we compute every vector in $\mathcal{M}(B_i \backslash B_{i-1})$ and add it to B_{i+1} if it is not in the affine subspace generated by B_{i+1}. At the end of the algorithm, these steps were applied to all vectors $M\vec{b}$ for $M \in \mathcal{M}$ and $\vec{b} \in B$, thus at most $|\mathcal{M}| \cdot |B| = O(tm)$ times. Computing $M\vec{b}$ takes $O(m^2)$ with standard matrix multiplication, and checking affine independence also takes $O(m^2)$ with the algorithm described in the next section. Thus the overall complexity is $O(tm^3)$. The theorem follows since $t = n^{\frac{1}{r+1}}$ and $m = n^{\frac{r-1}{r+1}}$. ∎

7. Maintaining a basis. In this section we give a dynamic algorithm for the following problem. We want to maintain a basis B of a d-dimensional linear subspace in R^m such that the following requests for a given vector $\vec{u} \in \mathbb{R}^m$ can be treated efficiently:

Membership query: "Is $\vec{u} \in \langle B \rangle$?"

Add to basis: Replace B by $B \cup \{\vec{u}\}$.

We can define the problem for affine subspaces as well. Fortunately, the latter can easily be reduced to the former: Let $f : \mathbb{R}^m \to \mathbb{R}^{m+1}$ be the function which maps a vector $\vec{u}$ to $\vec{u}'$ with $u'_i = u_i$ for $i = 1, \ldots, m$ and $u'_{m+1} = 1$. Then every vector $\vec{v}$ satisfies $\vec{v} \in [B]$ if and only if $f(\vec{v}) \in \langle f(B) \rangle$.

A solution to this problem requires a tricky data structure to encode the basis. The naive way would be to represent B by a matrix such that its column vectors are exactly those of B and to apply the Gaussian elimination algorithm (see, for example, [11]) to check whether $\vec{u} \in \langle B \rangle$. This would require $O(md^2)$ time steps.

Transforming the matrix in an upper triangular form is the bottleneck of this approach. The representation we choose for B will improve this complexity.

THEOREM 14. *There is a dynamic algorithm for maintaining a basis of a d-dimensional subspace in $\mathbb{R}^m$ which treats each request in time $O(m(m-d))$.*

Proof. We represent a nonempty basis B by the couple (T, B), where $T \in \mathbb{R}^{m \times m}$ is an orthogonal matrix and $\langle T(B) \rangle = \mathbb{R}^d \times \{0\}^{m-d}$. The empty basis is represented by $(I, \emptyset)$, where I is the identity matrix.

Since $T(\langle B \rangle) = \langle T(B) \rangle$, $\vec{u} \in \langle B \rangle$ if and only if $T\vec{u} \in \mathbb{R}^d \times \{0\}^{m-d}$. Thus verifying $\vec{u} \in \langle B \rangle$ is reduced to checking if the last $m - d$ components of $T\vec{u}$ are all 0 which can

be done in time $O(m(m-d))$.

Suppose $\vec{u} \notin \langle B \rangle$. We will show that there is an orthogonal matrix M affecting only components from $d+1$ to m which satisfies $(MT\vec{u})_{d+1} \neq 0$ and $(MT\vec{u})_i = 0$ for $i = d+2, \ldots, m$. Thus $\langle MT(B) \rangle = \langle T(B) \rangle$ and $\langle MT(B \cup \{\vec{u}\}) \rangle = \mathbb{R}^{d+1} \times \{0\}^{m-(d+1)}$. Therefore $(MT, B \cup \{\vec{u}\})$ represents $B \cup \{\vec{u}\}$.

We define M as the composition of two operators M_1 and M_2 we describe now. By hypothesis $\vec{u} \notin \langle B \rangle$; therefore there exists an index $k \in \{d+1, \ldots, m\}$ such that $(T\vec{u})_k \neq 0$. Define M_1 to be the permutation matrix which exchanges k and $d+1$.

Let $\vec{u}' = M_1 T \vec{u}$. Note that $\vec{u}'_{d+1} \neq 0$. Then for an arbitrary vector $\vec{v}$ we define $(M_2\vec{v})_i = \vec{v}_i$ for $i = 1, \ldots, d+1$ and $(M_2\vec{v})_i = \vec{v}_i - \vec{v}_{d+1}\vec{u}'_i/\vec{u}'_{d+1}$ for $i = d+2, \ldots, m$. Clearly M_1 and M_2 are orthogonal linear operators, and since

$$M_2\vec{u}' = (u'_1, u'_2, \ldots, u'_{d+1}, 0, \ldots, 0)$$

M satisfies the required property. We can compute $M_2 M_1 T$ in time $O(m(m-d))$, which concludes the proof. □

8. Putting it all together. We are now able to prove Theorem 1, that is, to give an algorithm to decide if a given LQCA is unitary. Using Theorem 10 to solve this problem we have to compute the associated border vectors and decide the corresponding Closed Affine Subspace problem. According to Theorem 12 the border vectors can be computed in time $O(n^{\frac{3(r-1)}{r+1}})$, and due to Theorem 13 the last problem can be solved in time $O(n^{\frac{3r-2}{r+1}})$, which concludes the proof.

9. Conclusion. A not necessarily simple LQCA can be transformed into a simple one with the same time evolution operator. Let the original neighborhood be $N = (a_1, \ldots, a_r)$. The size of the new neighborhood will be $s = a_r - a_1 + 1$. If we define the *expansion factor* of an LQCA as $e = (s+1)/(r+1)$, then the algorithm works in the general case in time $O(n^{e\frac{3r-1}{r+1}}) = O(n^{3e})$.

In the case of space-periodic configurations, van Dam [17] has shown that LQCAs can be efficiently simulated by QTMs. Watrous [18] gave an equivalent result for partitioned LQCAs. This question still remains open for the model of this paper.

Acknowledgments. We are thankful to Laurent Rosaz, Huong LêThanh, and Umesh Vazirani for several helpful conversations. We also wish to thank the anonymous referees for helpful comments and, in particular, for simplifying section 6.

REFERENCES

[1] P. Benioff, *Quantum mechanical Hamiltonian models of Turing machines*, J. Statist. Phys., 29 (1982), pp. 515–546.
[2] P. Benioff, *Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy*, Phys. Rev. Lett., 48 (1982), pp. 1581–1585.
[3] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.
[4] M. Biafore, *Can computers have simple Hamiltonians?*, in Proceedings of the Second Workshop on Physics and Computation, Dallas, TX, 1994, pp. 63–69.
[5] D. Deutsch, *Quantum theory, the Church–Turing principle and the universal quantum computer*, Proc. Roy. Soc. London Ser. A, 400 (1985), pp. 97–117.
[6] D. Deutsch, *Quantum computational networks*, Proc. Roy. Soc. London Ser. A, 425 (1989), pp. 73–90.
[7] C. Dürr, H. LêThanh, and M. Santha, *A decision procedure for well-formed linear quantum cellular automata*, Random Structures & Algorithms, 11 (1997), pp. 381–394.

[8] R. Feynman, *Simulating physics with computers*, Internat. J. Theoret. Phys., 21 (1982), pp. 467–488.
[9] R. Feynman, *Quantum mechanical computers*, Found. Phys., 16 (1986), p. 507–531.
[10] P. Høyer, *Note on Linear Quantum Cellular Automata*, manuscript, 1996.
[11] A. Lenstra and H. W. Lenstra, Jr., *Algorithms in Number Theory*, in Handbook of Theoretical Computer Science, Vol. A, Elsevier, Amsterdam, 1990, pp. 683–685.
[12] S. Lloyd, *A potentially realizable quantum computer*, Science, 261 (1993), pp. 1569–1571.
[13] S. Lloyd, *Envisioning a quantum supercomputer*, Science, 263 (1994), p. 695.
[14] N. Margolus, *Parallel quantum computation,* in Complexity, Entropy and the Physics of Information, Addison-Wesley, Reading, MA, 1994, pp. 273–287.
[15] K. Sutner, *De Bruijn graphs and linear cellular automata*, Complex Systems, 5 (1991), pp. 19–30.
[16] K. Sutner and W. Maas, *Motion planning among time-dependent obstacles*, Acta Inform., 26 (1988), pp. 93–122.
[17] W. van Dam, *A universal quantum cellular automaton*, in Proceedings of the Fourth Workshop on Physics and Computation, Boston, MA, 1996, pp. 323–331.
[18] J. Watrous, *On one dimensional quantum cellular automata*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 528–537.
[19] A. Yao, *Quantum circuit complexity*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 352–361.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1090–1118

A POLYLOGARITHMIC APPROXIMATION OF THE MINIMUM BISECTION*

URIEL FEIGE† AND ROBERT KRAUTHGAMER†‡

Abstract. A bisection of a graph with n vertices is a partition of its vertices into two sets, each of size $n/2$. The bisection cost is the number of edges connecting the two sets. It is known that finding a bisection of minimum cost is NP-hard. We present an algorithm that finds a bisection whose cost is within ratio of $O(\log^2 n)$ from the minimum. For graphs excluding any fixed graph as a minor (e.g., planar graphs) we obtain an improved approximation ratio of $O(\log n)$. The previously known approximation ratio for bisection was roughly $\sqrt{n}$.

Key words. approximation algorithms, graph partitioning, graph separators, dynamic programming, divide and conquer

AMS subject classifications. 68W25, 05C85

PII. S0097539701387660

1. Introduction.

Let $G(V, E)$ be an undirected graph with n vertices and m edges, where n is even. For a subset S of the vertices (with $S \neq \emptyset, V$), the *cut* $(S, V \setminus S)$ is the set of all edges in G with one endpoint in S and one endpoint in $V \setminus S$; these edges are said to be cut by $(S, V \setminus S)$. The *cost* of a cut is the number of edges in it.

A cut $(S, V \setminus S)$ is called a *bisection* of G if its two sides, S and $V \setminus S$, are each of size $n/2$. We denote the minimum cost of a bisection of G by b. *Minimum bisection* is the problem of computing b for an input graph G. This problem is NP-hard (see [7]), and we address the problem of approximating it.

An algorithm is said to approximate a minimization problem within ratio $f \geq 1$ if it runs in polynomial time and outputs a solution whose cost is at most f times the cost of the optimal solution. The problem is said to have a polynomial time approximation scheme (PTAS) if for every fixed $f > 1$ there is an algorithm with approximation ratio f.

1.1. Previous work.

Leighton and Rao [10, 11] showed how to approximate within ratio $O(\log n)$ *minimum-quotient cuts*, which we shall call *min-ratio cuts*. In these cuts, one wishes to minimize the *cut ratio* (also called *edge expansion* or *flux*) $c/|S|$, where c is the number of edges cut and $|S|$ is the cardinality of the smaller of the two vertex sets.

A *β-balanced cut* is a cut that partitions the graph into two parts, each of size at most βn. Leighton and Rao [10] used the approximate min-ratio cuts to find a 2/3-balanced cut (also called *edge separator*) with at most $O(b \log n)$ edges; see also [11, 14]. Note that such a 2/3-balanced cut does not provide an $O(\log n)$ approximation for

*Received by the editors April 10, 2001; accepted for publication (in revised form) November 19, 2001; published electronically March 13, 2002. This research was supported in part by a Minerva grant. A preliminary version of this paper appears in *Proceedings of the 41st Symposium on Foundations of Computer Science*, 2000, pp. 105–115.

http://www.siam.org/journals/sicomp/31-4/38766.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (feige@wisdom.weizmann.ac.il, robi@wisdom.weizmann.ac.il). The first author is the incumbent of the Joseph and Celia Reskin Career Development Chair. The second author's research was supported in part by the Open University of Israel.

‡Current address: International Computer Science Institute (ICSI) and Computer Science Division, University of California, Berkeley, CA 94720 (robi@cs.berkeley.edu).

the value of b. For example, when the graph consists of three disjoint cliques of equal size, an optimal 2/3-balanced cut has no edges, whereas $b = \Omega(n^2)$.

A straightforward approach for obtaining an exact bisection is to first find an almost balanced cut (e.g., using approximate min-ratio cuts) and then move a few low degree vertices from one side to the other. Using this approach one can approximate bisection within a ratio of $\tilde{O}(\sqrt{m/b})$ (we use $\tilde{O}(f)$ to denote $O(f \cdot \text{polylog } n)$); see, e.g., [11, Footnote 10] and [6]. This is a dramatic improvement over the naive ratio of $O(m/b)$ (achieved by arbitrarily picking $n/2$ vertices) but might still be larger than n.

In terms of n, the best approximation ratio previously known is $\tilde{O}(\sqrt{n})$, due to [6]. Their approach follows, in part, a divide and conquer paradigm. Two of their main tools are (i) approximate min-ratio cuts, which are used to recursively break the graph, and (ii) dynamic programming, which is used to combine certain possible parts into an exact bisection. The current work also uses a divide and conquer approach but in a more sophisticated way.

Additional related work includes the following. In [13], Saran and Vazirani give an algorithm that approximates bisection within a ratio of $n/2$. In [1], Arora, Karger, and Karpinski show that bisection has a PTAS for everywhere-dense graphs, i.e., graphs with minimum degree $\Omega(n)$. In [8], Garg, Saran, and Vazirani give an approximation ratio of 2 for the problem of finding a 2/3-balanced cut of minimum cost in a planar graph. Their result extends to a β-balanced cut, for any $\beta \geq 2/3$, but does not extend to a bisection, which is a 1/2-balanced cut. In [4], Bui and Jones show that for any fixed $\epsilon > 0$, it is NP-hard to approximate the minimum bisection within an *additive* term of $n^{2-\epsilon}$. In terms of approximation ratio, however, there is no known hardness of approximation result which excludes the possibility that bisection has a PTAS.

1.2. Our results. Our main result is an algorithm for approximating the minimum bisection within a polylogarithmic ratio.

THEOREM 1.1. *A bisection of cost within ratio of $O(\log^2 n)$ of the minimum can be computed in polynomial time.*

In section 2 we give an overview of the algorithm. On a high level, the algorithm follows a divide and conquer approach. The input graph is recursively divided into parts, using a new cut notion which we call an *amortized cut*, and then the parts are combined into a bisection using dynamic programming.

In section 4 we describe our algorithm for approximating bisection, based on a subroutine for finding an amortized cut. If the subroutine is guaranteed to find a ρ-amortized cut in a graph, the algorithm computes a bisection whose cost is within a ratio of $1 + O(\rho \log n)$ of the minimum.

In section 3 we devise an algorithm for finding an $O(\log n)$-amortized cut in a general graph. By using this algorithm as a subroutine in the $1 + O(\rho \log n)$ approximation algorithm for bisection, we are guaranteed that $\rho = O(\log n)$, proving Theorem 1.1. The subroutine uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. The best known approximation algorithms for min-ratio cut in general graphs, due to Leighton and Rao [10, 11] and due to [2, 12], have approximation ratio $\tau = O(\log n)$.

In certain graph families, there is a better approximation ratio τ for the min-ratio cut problem. If these graph families are closed under taking induced subgraphs, then we can approximate bisection within an improved ratio of $O(\tau \log n)$. For example, it is shown in [9] that in graphs excluding any fixed graph as a minor (e.g., bounded-genus graphs) min-ratio cut can be approximated within a constant ratio, i.e., $\tau = O(1)$.

THEOREM 1.2. *In graphs excluding any fixed graph as a minor (e.g., planar graphs), a bisection of cost within a ratio of $O(\log n)$ of the minimum can be computed in polynomial time.*

In section 5 we show that our results extend to several natural generalizations of the bisection problem. These extensions include, for example, bisection of graphs with arbitrary nonnegative edge costs and graph partitioning into three parts of equal size.

1.3. Conventions and notation. We will often denote the two sides of a (not necessarily optimal) bisection as *white* W and *black* B. A graph may have several different bisections of minimum cost. For the analysis, let us fix one of them (arbitrarily) and call it *the fixed optimal bisection* (W^*, B^*).

For V_1, V_2, two disjoint subsets of vertices in a graph, let $e(V_1, V_2)$ denote the number of edges with one endpoint in V_1 and the other endpoint in V_2. Subsets $V_1, V_2 \subset V$ are called a *partition* of V if they are nonempty, disjoint, and their union is equal to V. In our context, V is the vertex set of a graph, and then a partition $V = V_1 \cup V_2$ is equivalent to the cut (V_1, V_2).

A subset of vertices $S \subset V$ with $0 < |S| < |V|$ corresponds to a cut $(S, \overline{S})$ in the graph, where $\overline{S} = V \setminus S$. We denote by $r(S)$ the ratio of this cut, i.e., $r(S) = \frac{e(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$, and by $r'(S)$ the ratio of this cut towards S, i.e., $r'(S) = \frac{e(S,\overline{S})}{|S|}$. We call S a *part* of the graph, referring either to the set of vertices S or to the subgraph induced on S, depending on the context.

2. Overview and techniques. Our approximation algorithm for minimum bisection has three stages, as outlined below.

Stage 1: Decomposition. This stage consists of a sequence of *divide steps*. The input to a divide step is a *part* of the input graph G, i.e., a vertex set and the subgraph induced on it, and the output is a partition of the vertex set into two nonempty subsets, giving two new parts of the graph. These divide steps are applied on the input graph G recursively until it is decomposed into individual vertices.

The output of the whole decomposition stage is a binary tree T that we call the *decomposition tree*. Each node i of the tree contains a part V_i obtained in a divide step, as follows. The root of the tree contains the input graph G, the leaves of the tree contain individual vertices of G, and the two direct descendants of a node i are the two subparts obtained in the divide step of its part V_i.

To complete the description of the decomposition stage, we need to explain how a divide step is performed. This is done using a new notion called an *amortized cut*, which we define later in this section. We devise an algorithm for finding amortized cuts in section 3. The decomposition stage is described in more detail in section 4.1.

Stage 2: Labeling. Consider a *labeling* of the decomposition tree T, which labels each (nonleaf) tree node as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is α-*consistent* with respect to a white-black bisection (W, B) of the input graph if every part V_i (at a tree node i) satisfies that $|W \cap V_i| \leq \alpha|V_i|$ if the label of node i is white and that $|B \cap V_i| \leq \alpha|V_i|$ if the label of node i is black.

The desired outcome of the labeling stage is a labeling which is α-consistent with the fixed optimal bisection (W^*, B^*), called in short an *opt-consistent labeling*. However, an optimal bisection is not known to the algorithm; so instead of finding an opt-consistent labeling, this stage produces a family of labelings such that at least one member of the family is opt-consistent. The description of how this is done is

deferred to section 4.2. For the purpose of this overview, it will be convenient to think of the labeling stage as if it produces only one labeling, which is opt-consistent.

Stage 3*: Combining.* Given a decomposition tree T and an arbitrary (not necessarily opt-consistent) labeling of it, the combining stage assigns to each vertex v of the input graph G a *white charge* and a *black charge*. The two charges are simple to compute based on the labels along the path from the root of T to the leaf that contains the vertex v.

The *charge of a bisection* (W, B) of the input graph G (with respect to the labeling) is defined as the sum of the white charges of the vertices of W and the black charges of the vertices of B. The functions white charge and black charge have the property that for every bisection, charge is an upper bound on cost (regardless of the labeling).

If the charge is defined with respect to an opt-consistent labeling of T, then our notion of amortized cut used in the decomposition stage guarantees, in addition, that the charge of the fixed optimal bisection is within a polylogarithmic factor of its cost b. Hence, using the opt-consistent labeling produced by the labeling stage ensures that the input graph G contains a bisection whose charge is within polylogarithmic ratio of b.

Finding a bisection of minimum charge in G is relatively straightforward. Associate with each vertex a *net-charge*, which is its white charge minus its black charge, and pick the $n/2$ vertices with smallest net-charge to form one side, W, leaving the remaining $n/2$ vertices in another side, B. The bisection (W, B) that we find has minimum charge, and its cost is thus within a polylogarithmic factor of b, the cost of the minimum bisection.

It is interesting to note that finding a minimum cost bisection is an optimization problem with a quadratic objective function (minimizing the number of edges, where edges are pairs of vertices). Finding a minimum charge bisection (given the decomposition tree and an opt-consistent labeling) is an optimization problem with a linear objective function (sum of net-charges over individual vertices). Hence, in a sense, our algorithm performs a linearization of a quadratic function and loses a polylogarithmic factor in the process.

The above presentation of the combining stage was oversimplified. The output of the labeling stage is not one labeling that is opt-consistent, but rather a large family of labelings, such that at least one of them is opt-consistent. Moreover, this family has exponential cardinality; so we cannot try the above net-charge approach on each labeling separately. Instead, we exploit the structure of this family of labelings and use dynamic programming to compute a labeling from the family and a bisection such that the charge of this bisection with respect to this labeling is minimum over all labeling-bisection pairs. Details appear in section 4.4.

In the rest of the overview we shall introduce and discuss the notion of *amortized cut*, which is of central importance in bounding the ratio between the charge and the cost of the fixed optimal bisection. To motivate this new notion we present our algorithm as a divide and conquer algorithm. We then suggest a kind of cut that is desirable for the algorithm's divide step and call this cut notion an amortized cut.

Divide and conquer approach. A possible divide and conquer approach for a graph problem is to divide the input graph G into two parts (using a cut), solve a subproblem for each part, and then combine the solutions of the two subproblems into a solution for G. This approach can be applied recursively, and then the input graph G is recursively divided into smaller and smaller parts, where each part is associated

with a subproblem. Note that the divide step cut is a tool of this approach and is not intended to be a solution to the subproblem.

In our context, the graph problem is minimum bisection, and we apply this divide and conquer approach for the more general problem of cutting away an arbitrary number of vertices that is given as part of the input. (Bisection is the special case where the given number is $n/2$.) Similarly, the subproblem of each part requires to cut away (from that part) an arbitrary number of vertices that is given in the subproblem. Note that minimum bisection is a cut problem, and therefore in addition to the *divide step cuts* we have here also *solution cuts* (later called *combined cuts*). Note that the solution cut of a part need not be the same as the divide step cut of this part.

Our three stage algorithm outlined above follows this divide and conquer approach. The task of breaking the input graph into smaller and smaller parts is performed by the decomposition stage, whose decomposition tree T represents the recursive structure of the divide steps.

For such a divide and conquer approach to be successful, it is desirable that (i) each of the two subproblems can be solved separately and (ii) the solutions of the two subproblems can be combined while incurring a relatively small additional cost. Below we provide an overview of how our algorithms handle these issues.

Consider the problem of cutting away k vertices from a part $U \subseteq V$ of the input graph. The corresponding divide step uses a cut (U_1, U_2) of U to break this problem into the two subproblems of cutting away k_1 vertices from U_1 and of cutting away k_2 vertices from U_2, with $k = k_1 + k_2$. (For the sake of exposition assume that k_1, k_2 can be guessed.) Let us assume that the subproblem associated with each subpart U_i is solved separately (by recursion) and the solution obtained for it is a cut (C_i, F_i) with $|C_i| = k_i$ (see also Figure 2.1). The two solution cuts are then combined into a cut of U that separates $k = k_1 + k_2$ vertices, namely $(C_1 \cup C_2, F_1 \cup F_2)$. Let $Cut(U', k')$ denote the cost of the cut of U' that separates k' vertices and is found by the algorithm. Then the cost of the combined cut is given by

$$Cut(U,k) = Cut(U_1,k_1) + Cut(U_2,k_2) + e(C_1,F_2) + e(C_2,F_1). \tag{2.1}$$

Previous accounting method. The approach of [6] is based on a straightforward upper bound on the cost (2.1) of the combined cut. The additional cost incurred by the divide step, i.e., $e(C_1, F_2) + e(C_2, F_1)$, is at most the cost of all the edges cut by the divide step, i.e., $e(U_1, U_2)$, yielding the upper bound

$$Cut(U,k) \le Cut(U_1,k_1) + Cut(U_2,k_2) + e(U_1,U_2). \tag{2.2}$$

We remark that a bound similar to (2.2) is used in divide and conquer algorithms for many other graph problems, such as minimum cut linear arrangement (a.k.a. cutwidth); see, e.g., [11].

The divide steps of [6] use an approximate min-ratio cut to break each part U. This cut appears to be suitable for the bound (2.2) because it minimizes the cost of the cut (U_1, U_2), and at the same time tries to cut the part U into parts of roughly equal size, so as to minimize the depth of the recursion.

It is particularly instructive to evaluate the quality of our upper bound in the case where the computed cut $(C_1 \cup C_2, F_1 \cup F_2)$ is just the cut induced on U by the optimal bisection (W^*, B^*). Intuitively, we analyze the case where the algorithm happens to find the optimal bisection. In fact, we will later use dynamic programming to find a bisection for which the upper bound is minimized; so such an analysis bounds from above the cost of the output bisection.

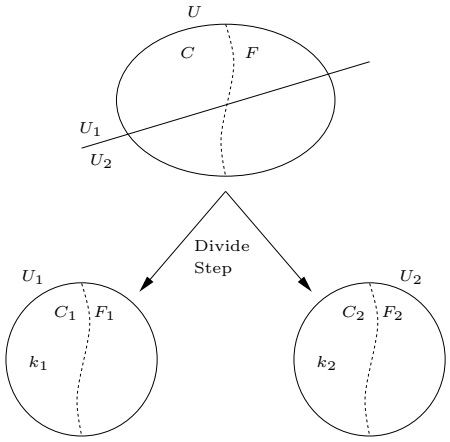


FIG. 2.1. *The divide and conquer paradigm.*

There are cases where the upper bound (2.2) is tight (i.e., holds with equality). Indeed, the cuts within each U_i are computed independently of each other, and so it might happen that all the edges between the two parts U_1, U_2 end up in the combined cut. However, this bound is insensitive to cases where only a few of the edges that are cut in the divide step end up in the combined cut, leading to a relatively poor approximation ratio.

New accounting method. We introduce a more sophisticated way of bounding the cost of the combined cut. Since $F_1 \subseteq U_1$ and $F_2 \subseteq U_2$ we can bound the cost of the combined cut by

$$Cut(U,k) \leq Cut(U_1,k_1) + Cut(U_2,k_2) + e(C_1,U_2) + e(C_2,U_1). \tag{2.3}$$

Unlike the actual cost (2.1), the upper bound (2.3) can be used in a divide and conquer approach, as follows. Let us call $e(C_1,U_2)+e(C_2,U_1)$ the *charge of the divide step* of U. This charge can be distributed into a charge $e(C_1,U_2)$ of the part U_1 and a charge $e(C_2,U_1)$ of the part U_2. The charge of a part U_i consists of the edges going from C_i to the other part U_{3-i} and thus depends on the cut (C_i,F_i) chosen in the part U_i but not on the cut chosen in the other part U_{3-i}. We obtain two separate subproblems (as in each part U_i we want to find a cut (C_i,F_i) for which the sum of the cost of this cut and the charge to this part is minimal), enabling a recursive divide and conquer approach. In contrast, the terms $e(C_1,F_2)$ and $e(C_2,F_1)$ of the actual cost of the combined cut depend on the cuts chosen in both parts and do not allow us to break the problem into two separate subproblems.

The new accounting method makes a distinction between the two sides C and F of the combined cut. Unlike, e.g., in (2.2), these two sides have different roles in the upper bound (2.3), and we will choose in a certain way which side is referred to as C (and which as F). Since we wish to minimize the charge, it makes sense to choose the smaller of the two sides to be C. In our analysis we have a somewhat relaxed condition, requiring that $|C| \leq \alpha|U|$, for a fixed $1/2 < \alpha < 1$. The task of identifying a side C as required above in each divide step (i.e., each node of the decomposition tree) is performed by the labeling stage, as explained in section 4.2.

The *charge of a bisection* is the upper bound that is obtained by applying the upper bound (2.3) recursively; i.e., it is the sum of the charges of all the divide steps.

In section 4.3 we discuss this notion in more detail, and in section 4.4 we show that its current formulation is equivalent to the one from Stage 3 of the algorithm outline (where the identification of a side C at each divide step corresponds to labeling of the decomposition tree T). From the current formulation it is straightforward that the charge of a bisection is always an upper bound on its cost (regardless of the identification of C at each divide step, i.e., the tree labeling).

We call the vertices of $C = C_1 \cup C_2$ *charged* and the vertices of $F = F_1 \cup F_2$ *free*. The edges in the part U can then be classified as charged-charged, charged-free, or free-free, according to their two endpoints.

Desired divide step. Rather than find a bisection of minimum cost, our approximation algorithm looks for a bisection of minimum charge. Our desired divide step is therefore one that guarantees that for the fixed optimal bisection, charge can be used to approximate cost. By the labeling stage, it suffices to refer here to charge with respect to an opt-consistent labeling; so from now on we assume that $|C| \leq \alpha|U|$ at each divide step.

Consider the charge of the fixed optimal bisection and recall that it is the sum of the charges of all the divide steps. The charge of a divide step of a part U is $e(C_1, U_2) + e(C_2, U_1)$ and can be written also as $e(C_1, F_2) + e(C_2, F_1) + 2e(C_1, C_2)$, i.e., the cost of the charged-free edges that the divide step cuts and twice the cost of the charged-charged edges that it cuts. Observe that a charged-free edge is always an edge of the fixed optimal bisection (and vice versa) and that each edge is cut exactly once in the decomposition stage. Hence, all the charged-free edges cut in all the divide steps are exactly all the edges of the fixed optimal bisection. So for the fixed optimal bisection, the difference between charge and cost is twice the cost of all the charged-charged edges cut in all the divide steps.

It is therefore desired that the divide step cuts relatively few charged-charged edges, where relative here is, with respect to b, the cost of the fixed optimal bisection. Since b is the total cost of the charged-free edges that are cut in all the divide steps, we seek an *amortization scheme* that amortizes the total cost of all charged-charged edges cut against the total cost of all charged-free edges cut. The partition of vertices to charged and free is not known to the divide step, and we therefore require that the amortization scheme holds for every possible partition of vertices to charged and free.

A simple amortization scheme can consider each divide step separately and amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges cut in the same divide step. Suppose that in every divide step the amortized cost in this method is at most ρ; i.e., at every part U we have that $e(C_1, C_2) \leq \rho[e(C_1, F_2) + e(C_2, F_1)]$. Then the total cost of charged-charged edges cut in all divide steps is clearly at most ρb, and the charge of the fixed optimal bisection is at most $(1 + 2\rho)b$.

The problem with this simple amortization scheme is that in order to guarantee that the scheme holds for all possible partitions of vertices to charged and free, ρ might be required to be at least n, a value that is too high for our intended application. For example, consider a graph that consists of two cliques of size $n/2$ connected by an edge e. If the divide step breaks any of the cliques, then letting this clique be C and the other clique be F, the amortization cost will be at least n. Otherwise, the divide step consists of the edge e and then, letting C consist of the two endpoints of e, the amortization cost will be infinite.

We employ a more complicated amortization scheme that allows a small amortization cost ρ but introduces an additional logarithmic factor. The reason for the

logarithmic factor is that this scheme amortizes against the same edge more than once (but, in a sense, not too many times). Another complication is that this scheme actually has two amortization methods, and it uses at each divide step the one that is better (for that divide step).

Amortized cut. We amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges *in the part being divided*, i.e. in the divide step of a part U we amortize $e(C_1, C_2)$ against $e(C, F)$. The edges that we amortize against are not cut in this divide step, and hence an edge may receive an amortized cost in many divide steps. However, our amortization scheme described below will guarantee that the total cost amortized against a single edge is at most $O(\rho \cdot \log n)$ for a suitable ρ. Since the edges that we amortize against are charged-free edges and hence edges of the fixed optimal bisection, it would follow that the total cost of the charged-charged edges cut in all the divide steps is at most $O(\rho \log n) \cdot b$, and so the charge of the fixed optimal bisection is $(1 + O(\rho \log n)) \cdot b$.

For motivation, consider the case where the divide steps recursion has depth $O(\log n)$, e.g., when all the divide steps are roughly balanced. In this case, an edge can receive an amortized cost in at most $O(\log n)$ divide steps. Suppose that in every divide step the amortized cost is at most ρ; i.e., in every part U we have that $e(C_1, C_2) \leq \rho \cdot e(C, F)$. Then the total cost amortized against a single edge is at most $O(\rho \log n)$.

We do not require that the divide steps are balanced but rather scale the amortization cost at a part U according to the imbalance of its divide step. Out of the several possible scaling factors we will use only the following two, where we assume, without loss of generality, that $|U_1| \leq |U_2|$. The first scaling factor is $e(C_1, F_1)/e(C, F)$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{e(C_1, F_1)}{e(C, F)} \cdot e(C, F). \tag{2.4}$$

The second scaling factor is $|C_1|/|C|$, and its corresponding amortization method requires that

$$e(C_1, C_2) \leq \rho \cdot \frac{|C_1|}{|C|} \cdot e(C, F). \tag{2.5}$$

Alternative formulations. The first amortization method (2.4) can also be written as $e(C_1, C_2) \leq \rho \cdot e(C_1, F_1)$. A convenient interpretation of this formulation is that we amortize against the charged-free edges inside U_1, the smaller side of the divide step cut (rather than inside U, the part being divided), and the amortized cost is required to be at most ρ.

The second amortization method (2.5) can be written also as $e(C_1, C_2) \leq \rho \cdot r'(C) \cdot |C_1|$, where $r'(C) = e(C, F)/|C|$. (See section 1.3 for the difference between $r'(C)$ and $r(C)$.) A convenient interpretation of this formulation is that we amortize against the vertices in C_1, the charged vertices inside the smaller side of the divide step cut, and the amortized cost is required to be at most $\rho \cdot r'(C)$.

Total amortized cost. The total cost amortized in the first method (2.4) is at most $O(\rho \log n) \cdot b$. Indeed, let us use the alternative formulation in which the amortization is only against edges inside U_1, the smaller side of the divide step cut. An edge can be inside U_1 in at most $\log n$ divide steps (since the size of the part it is contained in reduces at each such divide step by a factor of 2). Hence the total cost amortized in

this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$, and the claim follows.

The total cost amortized in the second method (2.5) is also at most $O(\rho \log n) \cdot b$. Indeed, we show in section 4.3 that the total cost amortized in this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$ (essentially by careful summation of the relevant terms of the form $|C_1|/|C|$), and the claim follows.

Our amortization scheme. Our amortization scheme chooses at each divide step the scaling factor that is better for this divide step, and so it suffices to have that at each part U at least one of (2.4) and (2.5) holds. It follows from the above discussion (see section 4.3 for a full proof) that the total cost amortized in both methods together is at most $O(\rho \log n) \cdot b$.

We can now formally define our desired divide step according to the (alternative formulations of) the two amortization methods described above. We call this cut an amortized cut.

DEFINITION. *Let (U_1, U_2) be a cut with $|U_1| \le |U_2|$ in a graph $G'(U, E')$, and let $U = C \cup F$ be a partition of the graph vertices U to charged vertices C and free vertices F. Let us denote $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$, as in Figure* 2.1. *Let*

$$\rho_e = \frac{e(C_1, C_2)}{e(C_1, F_1)} \qquad \text{and} \qquad \rho_v = \frac{e(C_1, C_2)}{|C_1| \cdot r'(C)}, \tag{2.6}$$

where $r'(C) = e(C, F)/|C|$. We call ρ_e the amortized cost for the edges*, and ρ_v the* amortized cost for the vertices*. (Note that ρ_e, ρ_v depend on C, F.)*

The amortized cost of the cut (U_1, U_2) *is the maximum of* $\min\{\rho_e, \rho_v\}$*, where the maximum is taken over all partitions $U = C \cup F$ with $0 < |C| \le \alpha|U|$ for a fixed $\frac{1}{2} \le \alpha < 1$. We say that the cut (U_1, U_2) is* ρ-amortized *if its amortized cost is at most ρ.*

In order for us to correctly handle cases where there is no cost to amortize against, we use the convention that $\frac{0}{0}$ is defined to be 0 and that $\frac{t}{0}$ for $t > 0$ is defined to be ∞. In particular, we may extend (2.6) to the case where $C = \emptyset$ and then ρ_e, ρ_v are defined to be 0.

Convenient characterizations. A convenient characterization of an amortized cut is given in the following proposition, whose proof is straightforward. (We will use this characterization in section 4.)

PROPOSITION 2.1. *A cut (U_1, U_2) with $|U_1| \le |U_2|$ is ρ-amortized if and only if for every $C \subset U$ with $|C| \le \alpha|U|$ and $F = U \setminus C$,*

$$e(C_1, C_2) \le \rho \cdot \max\left\{ e(C_1, F_1),\ \tfrac{|C_1|}{|C|} \cdot e(C, F) \right\},$$

where $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$.

The restriction $|C| \le \alpha|U|$ implies that the two terms $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}}$ and $r'(C) = \frac{e(C,F)}{|C|}$ differ by no more than a constant factor. Indeed, $\min\{|C|, |F|\} = \Theta(|C|)$ and hence $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}} = \frac{e(C,F)}{\Theta(|C|)} = \Theta(r'(C))$.

We can therefore characterize the amortized cost of a cut (up to constant factors) in terms of $r(C)$ rather than $r'(C)$. (We will use this characterization in section 3.)

PROPOSITION 2.2. *A cut (U_1, U_2) with $|U_1| \leq |U_2|$ is $O(\rho)$-amortized if for every partition $U = C \cup F$ with $0 < |C| \leq \alpha|U|$,*

$$\min\left\{\frac{e(C_1,C_2)}{e(C_1,F_1)}\ ,\ \frac{e(C_1,C_2)}{|C_1|\cdot r(C)}\right\} \leq \rho, \tag{2.7}$$

where $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$.

Remarks. Observe that without the restriction $|C| \leq \alpha|U|$, the amortized cost ρ might be required to be $\Omega(|U|)$, a value that is too high for our intended application. For example, consider a clique on n vertices and a cut (U_1, U_2) in it with $|U_1| \leq |U_2|$. Let one vertex of U_2 be the only free vertex and the rest of the vertices be charged. The number of charged-charged edges cut is $|U_1| \cdot \Theta(n)$. There are no charged-free edges in U_1; so the amortized cost for the edges is $\rho_e = \infty$. The number of charged vertices in the smaller side is $|U_1|$ and $r'(C) = \frac{n-1}{n-1} = 1$; so the amortized cost for the vertices is $\rho_v = \frac{|U_1|\Theta(n)}{|U_1|\cdot 1} = \Theta(n)$. Therefore, the amortized cost of any cut would be $\rho = \Omega(n)$.

In contrast, we show that the restriction $|C| \leq \alpha|U|$ allows us to obtain relatively small values of ρ. Namely, there always exists a cut whose amortized cost is $\rho = O(1)$, and a cut whose amortized cost is $O(\log|U|)$ can be computed efficiently. We remark that our constructions are stronger than those required by Proposition 2.2, as they satisfy (2.7) with no restriction on $|C|$. (The point is that we use $r(C)$ rather than $r'(C)$, which makes a significant difference when $|C| \gg |F|$, as in the above clique example.)

Note that the amortized cost ρ is not an approximation ratio. On the one hand, it is not clear from the definition that every graph has an $O(1)$-amortized cut. On the other hand, the amortized cost of a cut may be smaller than 1, as demonstrated by a graph that consists of two cliques of size $n/2$ connected by an edge. The cut that separates the two cliques can be seen to have amortized cost $O(1/n)$.

3. Finding an amortized cut. In this section we devise an algorithm for finding $O(\log n)$-amortized cuts in general graphs and $O(1)$-amortized cuts in graphs excluding any fixed minor (e.g., planar graphs). The input graph for this algorithm is denoted by G (though it may be just a part of the input graph for bisection). We assume that G is connected, as otherwise we can separate a connected component while cutting no edges at all.

Section 3.1 shows that every optimal min-ratio cut is an $O(1)$-amortized cut. It follows that in every graph there exists an $O(1)$-amortized cut. An optimal min-ratio cut is NP-hard to find in general graphs, and we thus consider approximate min-ratio cuts.

Section 3.2 demonstrates an approximate min-ratio cut which would be a poor divide step for our accounting method. In particular, its amortized cost is high, showing that the arguments of section 3.1 do not immediately extend from optimal min-ratio cuts to approximate ones.

Section 3.3 presents an algorithm that uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. Known algorithms for the min-ratio cut problem in general graphs [11, 2, 12] have approximation ratio $\tau = O(\log n)$, and we can thus find an $O(\log n)$-amortized cut. For certain graph families a better approximation ratio is possible. For example, in graphs excluding any fixed minor, a ratio of $\tau = O(1)$ is known due to [9], and we can thus find an $O(1)$-amortized cut.

3.1. Min-ratio cuts are $O(1)$-amortized. We give an $O(1)$ upper bound on the amortized cost of optimal min-ratio cuts. The proof is based on the characterization given in Proposition 2.2 for an amortized cut. We remark that our proof satisfies (2.7) with no restriction on $|C|$.

LEMMA 3.1. *An optimal min-ratio cut in a graph is $O(1)$-amortized.*

Proof. Let (V_1, V_2) be an optimal min-ratio cut in a graph G and assume, without loss of generality, that $|V_1| \le |V_2|$. Let $V = C \cup F$ be an arbitrary partition of the graph vertices to charged vertices C and free vertices F, with $0 < |C| < |V|$, and denote $C_i = V_i \cap C$ and $F_i = V_i \cap F$ for $i = 1, 2$ (see also Figure 3.1). We show below that

$$\min\left\{\frac{e(C_1,C_2)}{e(C_1,F_1)},\ \frac{e(C_1,C_2)}{|C_1|\cdot r(C)}\right\} \le 2, \tag{3.1}$$

and then by Proposition 2.2 we will have that (V_1, V_2) is $O(1)$-amortized, which proves the lemma. Note that we can assume that $|C_1| > 0$, as otherwise there is nothing to prove.

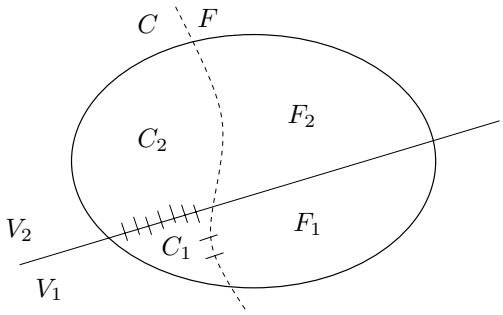


FIG. 3.1. *The amortized cost of an optimal min-ratio cut* (V_1, V_2).

One easy case is when $\frac{e(C_1,C_2)}{e(C_1,F_1)}$ (i.e., the amortized cost for the edges ρ_e) is at most 2, which clearly implies (3.1).

Another easy case is when $\frac{e(C_1,C_2)}{|C_1|} \le 2r(V_1)$. Since (V_1, V_2) is an optimal min-ratio cut, we also have that $r(V_1) \le r(C)$. We obtain that $\frac{e(C_1,C_2)}{|C_1|\cdot r(C)} \le 2\frac{r(V_1)}{r(C)} \le 2$, and therefore (3.1) holds.

We next prove that one of the two easy cases above must hold, as otherwise we must have that $r(F_1) < r(V_1)$, in contradiction with (V_1, V_2) being an optimal min-ratio cut. Indeed, assume that $e(C_1, C_2)/e(C_1, F_1) > 2$ and $\frac{e(C_1,C_2)}{|C_1|} > 2r(V_1)$. Since $r(V_1) = \frac{e(V_1,V_2)}{|V_1|}$ is the average degree from V_1 to V_2, it can be represented as the following convex combination of the average degree from C_1 to V_2 and the average degree from F_1 to V_2, namely

$$r(V_1) = \frac{|F_1|}{|V_1|}\cdot\frac{e(F_1,V_2)}{|F_1|} + \frac{|C_1|}{|V_1|}\cdot\frac{e(C_1,V_2)}{|C_1|}.$$

Since $r(F_1) = \frac{e(F_1,V_2)+e(F_1,C_1)}{|F_1|}$ (note that $|F_1| \le |V_1| \le \frac{1}{2}|V|$), we can represent $r(V_1)$ also as

$$r(V_1) = \frac{|F_1|}{|V_1|}\cdot r(F_1) + \frac{|C_1|}{|V_1|}\cdot\left[\frac{e(C_1,V_2)-e(F_1,C_1)}{|C_1|}\right].$$

By the above two assumptions (that exclude the easy cases) we have that

$$\frac{e(C_1,V_2)-e(F_1,C_1)}{|C_1|} \geq \frac{e(C_1,C_2)-e(F_1,C_1)}{|C_1|} \geq \frac{\frac{1}{2}e(C_1,C_2)}{|C_1|} > r(V_1).$$

The last two inequalities imply that

$$r(V_1) > \frac{|F_1|}{|V_1|}\cdot r(F_1) + \frac{|C_1|}{|V_1|}\cdot r(V_1).$$

We obtained that some convex combination of $r(F_1)$ and $r(V_1)$ is smaller than $r(V_1)$, and we can therefore conclude that $r(F_1) < r(V_1)$. This contradicts the fact that (V_1, V_2) is an optimal min-ratio cut and completes the proof of Lemma 3.1. ∎

The converse of Lemma 3.1 is not true, and an $O(1)$-amortized cut can be an $\Omega(n)$-approximate min-ratio cut, as follows from the next proposition with $t = O(1)$.

PROPOSITION 3.2. *Fix a constant $1/2 < \alpha < 1$ for the definition of an amortized cut. Then for every $t = o(n)$, there is an $O(1/t)$-amortized cut which is an $\Omega(n/t)$-approximate min-ratio cut.*

Proof. Consider a graph on n vertices, for a sufficiently large n, that consists of three cliques as follows. V_1 is a clique on t vertices, V_2 is a clique on αn vertices, and V_3 is a clique on the remaining $\Omega(n)$ vertices. In addition, the graph contains one edge connecting V_1 to V_2 and one edge connecting V_2 to V_3.

The cut $(V_1, V_2 \cup V_3)$ has amortized cost $O(1/t)$. Indeed, let $C \cup F$ be a partition of the vertices with $|C| \leq \alpha n$. We may assume that C contains both endpoints of the edge between V_1 and V_2, as otherwise the cut contains no charged-charged edges and its amortized cost is 0. So we have that the cost of the charged-charged edges cut is 1 and that both V_1 and V_2 contain at least one charged vertex. If V_1 contains also at least one free vertex, then the number of charged-free edges in V_1 is at least $t-1$ and hence $\rho_e = \frac{e(C_1,C_2)}{e(C_1,F_1)} \leq 1/(t-1)$. Otherwise, we have $C_1 = V_1$; since there are at most αn charged vertices, and at least one of them is in V_1, we have that V_2 contains also free vertices and thus $e(C,F) \geq \Omega(n)$; it follows that $\rho_v = \frac{e(C_1,C_2)}{e(C,F)} \cdot \frac{|C|}{|C_1|} \leq O(1/t)$.

The cut $(V_1, V_2 \cup V_3)$ is an $\Omega(n/t)$-approximate min-ratio cut. Indeed, the ratio of this cut is $r(V_1) = 1/t$, while the cut $(V_3, V_1 \cup V_2)$ is an optimal min-ratio cut and has ratio $r(V_3) = O(1/n)$. ∎

The next corollary follows from Lemma 3.1.

COROLLARY 3.3. *In every graph there exists an $O(1)$-amortized cut.*

Corollary 3.3 is optimal up to constant factors, and there are graphs for which any cut has amortized cost $\Omega(1)$. For example, consider a clique on n vertices. Given a cut (V_1, V_2) with $|V_1| \leq |V_2|$, let α be the constant in the amortized cut definition and take $(\alpha - 1/2)n$ vertices of V_2 and all of V_1 to be the charged vertices. It can be seen that $\rho_e = \infty$ and $\rho_v = \Theta(1)$, and so the amortized cost of the cut (V_1, V_2) is $\Omega(1)$, as claimed.

3.2. Approximate min-ratio cuts might be poor amortized cuts. We demonstrate that an approximate min-ratio cut of a graph might be a poor divide step and, in particular, a poor amortized cut. Consider, for example, the following graph G on $2n + 2\sqrt{\epsilon n}$ vertices for a fixed $0 < \epsilon < 1$ (see also Figure 3.2). The vertex set of the graph is $F_1 \cup F_2 \cup C_1 \cup C_2$, where each of F_1, F_2 are of size n, each of C_1, C_2 are of size $\sqrt{\epsilon n}$, and each of the four subsets forms a clique. These four cliques are connected as follows. Between F_1 and F_2 there are n edges that form a matching (i.e., have no common endpoint). Between C_1 and C_2 there are all possible ϵn edges; thus

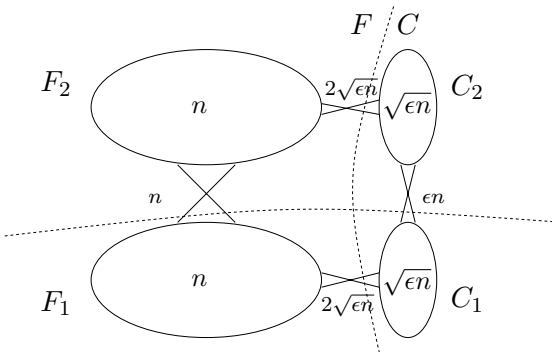


FIG. 3.2. *A poor divide step by an approximate min-ratio cut.*

$C_1 \cup C_2$ forms a clique. There are also $2\sqrt{\epsilon n}$ edges between F_i and C_i (for $i = 1, 2$) so that their endpoints at F_i are distinct and each vertex of C_i is an endpoint of exactly two of these edges.

Let $C = C_1 \cup C_2$ be the charged vertices and $F = F_1 \cup F_2$ the free vertices. Such a partition to charged and free may reflect the "right" cut of $2\sqrt{\epsilon n}$ vertices from the graph G (if, e.g., the input graph for bisection consists of this graph G and a clique on $2n - 2\sqrt{\epsilon n}$ vertices).

Consider a divide step based on the cut $(F_1 \cup C_1, F_2 \cup C_2)$, whose ratio is nearly optimal. Indeed, an optimal min-ratio cut in this graph is $(F_1, C_1 \cup F_2 \cup C_2)$ and its ratio is $1 + 2\sqrt{\epsilon}/\sqrt{n}$. The cut $(F_1 \cup C_1, F_2 \cup C_2)$ has a slightly higher ratio of $(1 + \epsilon)(1 - o(1))$, and so it is a $1 + \epsilon$ approximate min-ratio cut.

Observe that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step. It cuts ϵn charged-charged edges while the total number of charged-free edges in G (and the bisection cost in the input graph) is only $4\sqrt{\epsilon n}$. According to the new accounting method, such a divide step does not give an approximation ratio better than $\Omega(\sqrt{\epsilon n})$.

The observation that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step is supported by its high amortized cost. The amortized cost for the edges is $\rho_e = \epsilon n / 2\sqrt{\epsilon n} = \sqrt{\epsilon n}/2$. The ratio of the cut (C, F) is $r(C) = r'(C) = 2$; so the amortized cost for the vertices is $\rho_v = \epsilon n / (\sqrt{\epsilon n} r'(C)) = \sqrt{\epsilon n}/2$. We conclude that a $1 + o(1)$ approximate min-ratio cut might have amortized cost $\rho \geq \min\{\rho_e, \rho_v\} = \sqrt{\epsilon n}/2$.

3.3. Finding $O(\tau)$-amortized cut. We present an algorithm that finds an $O(\tau)$-amortized cut, given a subroutine for computing a τ-approximate min-ratio cut. The algorithm is motivated by the $O(1)$ upper bound on the amortized cost of a min-ratio cut shown in section 3.1. In particular, we examine what additional properties are required in order to extend the analysis of Lemma 3.1 from optimal min-ratio cuts to approximate ones.

The proof of Lemma 3.1 uses *twice* the fact that (V_1, V_2) is an optimal min-ratio cut. In the first usage we had that $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2$, which extends to the case where (V_1, V_2) is an approximate min-ratio cut with the approximation ratio carried over to the amortized cost; i.e., if (V_1, V_2) is a τ-approximate min-ratio cut, then we have $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2\tau$.

The second time we used the fact that (V_1, V_2) is an optimal min-ratio cut was to say that $r(F_1) < r(V_1)$ cannot hold and gives a contradiction. In general, this usage does not extend to an approximate min-ratio cut, as demonstrated by the example

in section 3.2. However, the proof does extend to an approximate min-ratio cut if we have the additional property that the ratio of V_1 is minimal over all its subsets F_1; i.e., $r(V_1) \leq r(F_1)$ for all $F_1 \subset V_1$. We therefore obtain that the proof of Lemma 3.1 extends to approximate min-ratio cuts as follows.

LEMMA 3.4. *Let (V_1, V_2) be a τ-approximate min-ratio cut in a graph, with $|V_1| \leq |V_2|$. If $r(V_1) \leq r(F_1)$ for every $F_1 \subset V_1$, then (V_1, V_2) is an $O(\tau)$-amortized cut.*

Note that the proof of Lemma 3.4 is not symmetric with respect to the two amortization methods. It guarantees that either $e(C_1, C_2)/e(C_1, F_1) \leq 2$ (i.e., the amortized cost for the edges ρ_e is at most 2) or $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\tau$ (i.e., the amortized cost for the vertices ρ_v is $O(\tau)$). In contrast, in the proof of Lemma 3.1 for optimal min-ratio both amortization costs are $O(1)$.

The amortized cut algorithm. We use Lemma 3.4 to devise an algorithm that finds an $O(\tau)$-amortized cut based on a τ-approximate min-ratio cut. The algorithm, described in Figure 3.3, starts with a τ-approximate min-ratio cut (V_1, V_2) and then "fixes" it so that it would also be "minimal" with respect to containment, as required by Lemma 3.4. It then follows that the output cut is $O(\tau)$-amortized.

In order to "fix" the cut (V_1, V_2), the algorithm uses minimum (s, t)-cuts in a related graph G', which is defined in step 2. The related graph G' contains edges of the input graph G, as well as new edges. The edges from G have unit capacity, while the capacity of the new edges is some parameter $p > 0$. Step 3 then finds the optimal value of p with respect to the minimum (s, t)-cut. Before discussing implementation issues of step 3, let us analyze the algorithm's correctness.

Algorithm FINDAMORTIZED**.**

1. Find in the input graph $G = (V, E)$ a τ approximate min-ratio cut (V_1, V_2) with $|V_1| \leq |V_2|$.
2. Create a related graph G':
   – Merge all vertices of V_2 into a single vertex t, removing self-loops at t, and keeping all edges to V_1, including parallel edges.
   – Add a new vertex s which is connected to each vertex of V_1 by an edge whose capacity (weight) is a parameter $p > 0$.
3. Let S denote the vertices of V_1 which are on the same side with s in a minimum (s, t)-cut of G'.
   – Find (e.g., by binary search) the minimum $p > 0$ for which $S \neq \emptyset$. (Possibly, $S = V_1$).
4. Output the cut $(S, V \setminus S)$ of the input graph.

FIG. 3.3. *Algorithm for amortized cuts.*

LEMMA 3.5. *The cut $(S, V \setminus S)$ output by algorithm* FINDAMORTIZED *is a τ-approximate min-ratio cut. In addition, every nonempty subset of V_1 has ratio at least as large as S, i.e., $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$.*

Proof. Consider an arbitrary value p and an arbitrary (s, t)-cut in the related graph G' with the corresponding set $S \subset V_1$ (see Figure 3.4). The cut consists of (i) edges between s and $V_1 \setminus S$ (each of capacity p), (ii) edges between S and $V_1 \setminus S$ (these are edges from the input graph G), and (iii) edges between S and t (these are the edges between S and V_2 in the input graph G). The capacity of this (s, t)-cut is thus

$$\text{cap}(S) = p \cdot |V_1 \setminus S| + e(S, V \setminus S),$$

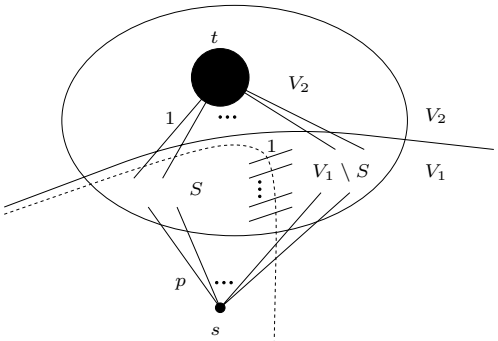


FIG. 3.4. *An (s,t)-cut in the related graph G'.*

where, as usual, $e(\cdot,\cdot)$ denotes the number of corresponding edges in the input graph G. In the special case of the empty set $S = \emptyset$, the capacity of the (s,t)-cut is

$$\mathrm{cap}(\emptyset) = p \cdot |V_1|.$$

Fixing the value of p, let us compare the capacity of the cut defined by the empty set $\emptyset$ with that of an arbitrary set $S \neq \emptyset$, i.e., $\mathrm{cap}(\emptyset)$ vs. $\mathrm{cap}(S)$. The empty set $\emptyset$ yields a smaller capacity whenever

$$\begin{aligned} p \cdot |V_1| &< p \cdot |V_1 \setminus S| + e(S, V \setminus S) \\ &\Updownarrow \\ p &< \frac{e(S, V \setminus S)}{|S|} = r(S), \end{aligned}$$

where $r(S)$ is the ratio of the cut $(S, V \setminus S)$ in the input graph G. (Note that $|S| \leq |V_1| \leq \frac{1}{2}|V|$ and that $r(S) > 0$ if G is connected.)

We claim that the value of p found at step 3 is essentially $p^* = \min\{r(S) : \emptyset \neq S \subseteq V_1\}$. Indeed, when $p < p^*$, a minimum (s,t)-cut in G' corresponds to $S = \emptyset$, and, when $p > p^*$, a minimum (s,t)-cut yields a set $S \neq \emptyset$. When $p = p^*$, a minimum (s,t)-cut can be obtained either by $S = \emptyset$ or by (one or more) $S \neq \emptyset$ with $r(S) = p^*$.

When $p = p^* + \epsilon$ for a very small $\epsilon > 0$, only the sets $S \neq \emptyset$ with $r(S) = p^*$ give smaller capacity than the empty set, and thus a minimum (s,t)-cut is obtained by one of these sets S. By the definition of p^*, this set $\emptyset \neq S \subset V_1$ has minimal ratio $r(S)$ over all nonempty subsets of V_1, i.e., $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$, as claimed. Furthermore, since $S = V_1$ is included in this range, we get that $r(S) \leq r(V_1)$ and hence $(S, V \setminus S)$ is a τ-approximate min-ratio cut, finishing the proof. We remark that a slightly modified algorithm can guarantee in addition that $r(S) < r(S')$ for every $S' \subset S$ with $S' \neq \emptyset, S$. Details are omitted. □

THEOREM 3.6. *Given a subroutine for computing a τ-approximate min-ratio cut, algorithm* FINDAMORTIZED *finds an $O(\tau)$-amortized cut.*

Proof. Lemma 3.5 guarantees that the cut found by the algorithm satisfies the requirements of Lemma 3.4, from which it follows that the cut is $O(\tau)$-amortized. □

We now address the issue of implementing step 3. Observe that p^* is the maximum value p for which the empty set $\emptyset$ gives a minimum (s,t)-cut. Since, by definition, p^* is the ratio $r(S)$ of a set S, it has only n^3 possible values, which can be exhaustively

searched. Alternatively, p^* can be found in $O(\log n)$ iterations of binary search, since as an exact multiple of $1/|S|$ it is bounded between 0 and n, and the difference between any two of its possible values is more than $1/n^2$.

Once we find p^*, we need to find a set $S \neq \emptyset$ that gives a minimum (s,t)-cut for p^*. We can either guess a vertex of V_1 and merge it with s before computing the minimum (s,t)-cut for p^* or, alternatively, compute a minimum (s,t)-cut for $p = p^* + \epsilon$ with, e.g., $\epsilon = 1/n^2$.

4. The bisection algorithm. In this section we describe our approximation algorithm for bisection and prove the following theorem. (See section 2 for the definition of an amortized cut.)

THEOREM 4.1. *Given a subroutine that finds a ρ-amortized cut, a bisection within a ratio of $1 + O(\rho \log n)$ of the minimum can be found in polynomial time.*

4.1. Decomposition stage. The decomposition stage recursively divides the input graph $G = (V, E)$ into smaller and smaller parts using a ρ-amortized cut subroutine (e.g., the one devised in section 3). Each part is further divided unless it consists of a single vertex.

The decomposition stage builds a rooted binary tree T, called the *decomposition tree*, which corresponds to the recursive decomposition of the input graph G in a natural way, as follows. (Throughout, we call the vertices of T *nodes* to avoid confusion with the vertices of the input graph G.) Each tree node i contains a part $V_i \subseteq V$ that was found during the recursive decomposition. The root node of T contains V, i.e., the whole input graph G. Let us denote the two children of a nonleaf node i by $L(i)$ and $R(i)$. Then their two parts $V_{L(i)}, V_{R(i)}$ are the result of dividing V_i; i.e., the ρ-amortized cut found in V_i is $(V_{L(i)}, V_{R(i)})$. A leaf of the tree T contains a part that consists of a single vertex of G. Therefore T contains exactly n leaves and $n-1$ nonleaf nodes.

4.2. Labeling stage. Recall the following definitions from section 2. A *labeling* of the decomposition tree T labels each nonleaf node of the tree as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is α-*consistent* with respect to a white-black bisection (W, B) of G if every tree node i satisfies the following: If the label of node i is white, then $|W \cap V_i| \leq \alpha |V_i|$, and if the label of node i is black, then $|B \cap V_i| \leq \alpha |V_i|$ (where V_i is the part contained in node i). A labeling is called *opt-consistent* if it is α-consistent with the fixed optimal bisection (W^*, B^*).

The labeling stage produces a family $\mathcal{F}$ of labelings. The cardinality of $\mathcal{F}$ is exponential in n; so rather than listing its members explicitly, the labeling stage produces an implicit representation of $\mathcal{F}$. The actual work of the labeling stage is to *mark* certain nodes of T, and these nodes implicitly define the family $\mathcal{F}$, as described below.

The labeling stage marks some of the nodes of T in a process that goes from the root of T towards its leaves, as follows. The root of T is always marked, and any other node i in the tree is marked in this process if its closest marked ancestor j satisfies $|V_i| \leq \frac{1}{2\alpha}|V_j|$. (As before, V_i and V_j are the parts contained in the nodes i and j, respectively.) Note that the constant α is chosen so that $\frac{1}{2} < \alpha < 1$, implying $\frac{1}{2} < \frac{1}{2\alpha} < 1$.

A labeling of T is said to be *derived* from the marked nodes if the label of every unmarked node is the same as the label of its closest marked ancestor. (There is no

restriction on the labels of the marked nodes.) Note that in this case the labels of the marked nodes uniquely define the labels of all the internal tree nodes.

The family $\mathcal{F}$ produced by the labeling stage consists of all the labelings that can be derived from the marked nodes. Since each of the $\Omega(n)$ marked nodes can be labeled arbitrarily by one of two colors, the resulting family of labelings has exponentially large cardinality, and we cannot explicitly list all the family members. Instead, the algorithm implicitly represents this family $\mathcal{F}$ by identifying which are the marked nodes.

LEMMA 4.2. *The family of labelings $\mathcal{F}$ contains at least one opt-consistent labeling.*

Proof. Let the white-black cut (W, B) be the fixed optimal bisection. Consider the labeling that is derived from the marked nodes, with the label of each marked node i being the color in minority among the vertices of V_i.

This labeling is clearly in the family $\mathcal{F}$, and we claim that it is also opt-consistent. Indeed, the label of a marked node i is by definition the minority color in V_i. The label of an unmarked node i is the same as the label of its closest marked ancestor j. Suppose, without loss of generality, that this label (of i and j) is white. Then at most half the vertices of V_j are white, i.e., $|W \cap V_j| \leq \frac{1}{2}|V_j|$. Observe that $V_i \subset V_j$ and $|V_i| > \frac{1}{2\alpha}|V_j|$ and hence $|W \cap V_i| \leq |W \cap V_j| \leq \frac{1}{2}|V_j| < \alpha|V_i|$. Hence, this labeling of $\mathcal{F}$ is opt-consistent. ∎

4.3. The charge of a bisection. We now formally define the *charge of a bisection* (W, B) with respect to the decomposition tree T and a labeling of it. The reference to T will later be omitted, as we always refer to the tree computed in the decomposition stage.

DEFINITION. *Let (W, B) be a bisection of the input graph and assume we are given a decomposition tree T and a labeling of it. For each (nonleaf) node i of T, if i is labeled white, then we let (see Figure 4.1) $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if i is labeled black, then we let $C_i = B \cap V_i$ and $F_i = W \cap V_i$. We obtain a cut (C_i, F_i) of the part V_i, and say that C_i is* charged *and F_i is* free. *The* charge of the divide step *of a (nonleaf) node i is defined as*

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

The charge of the bisection (W, B) *is defined as the sum of all the divide steps charges, i.e.,*

$$\sum_{i \in T} e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

(These charges are defined with respect to T and a labeling of it.)

Bisection charge vs. cost. In certain conditions, a bisection charge can approximate its cost. As shown below, the charge of a bisection upper bounds its cost, and the gap between them is not too large if the charge is taken with respect to an α-consistent labeling (as in the case of the fixed optimal bisection and an opt-consistent labeling).

LEMMA 4.3. *The charge of a bisection (W, B) with respect to any labeling is at least as large as its cost.*

Proof. As we have seen in section 2, the true cost of the (W, B) edges cut in a divide step i is $e(C_i \cap V_{L(i)}, F_i \cap V_{R(i)}) + e(C_i \cap V_{R(i)}, F_i \cap V_{L(i)})$ and is therefore not larger than the charge of this step. The proof follows by summing over all divide steps,

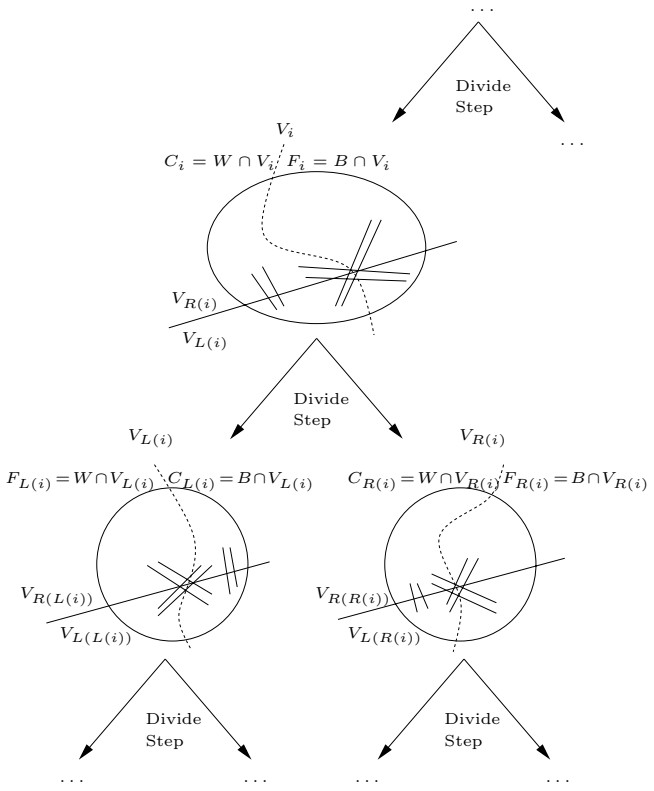


FIG. 4.1. *The charge of a bisection* (W, B) *throughout the decomposition tree.*

since the decomposition stage eventually divides the graph into individual vertices, and so every edge of the bisection (W, B) is cut at some divide step. ∎

LEMMA 4.4. *The charge of a bisection* (W, B) *with respect to a labeling that is* α*-consistent with it is at most* $e(W, B) \cdot (1 + O(\rho \log n))$.

Proof. Consider a bisection (W, B) and a labeling of T that is α-consistent with it. As we have seen in section 2 and in Lemma 4.3 the charge of a divide step is larger than the true cost of the (W, B) edges cut in that step by the cost of the charged-charged edges cut in that divide step. Summing over the divide steps we get that the charge of (W, B), the fixed optimal bisection, is larger than its cost by $2\sum_i e(C_i \cap V_{L(i)}, C_i \cap V_{R(i)})$, where i ranges over all (nonleaf) nodes i in T. We use the shorter notation $C_L = C_i \cap V_{L(i)}$ and $C_R = C_i \cap V_{R(i)}$, where i is clear from the context.

To upper bound $2\sum_i e(C_L, C_R)$, observe that each part V_i is divided using a ρ-amortized cut and that the α-consistent labeling guarantees that $|C_i| \leq \alpha |V_i|$ for all nodes i; so we can use the amortization scheme of section 2. Namely, let us assume, without loss of generality, that the decomposition stage places in the left child of a node i the smaller of the two subparts of V_i; i.e., $|V_{L(i)}| \leq |V_{R(i)}|$ for every nonleaf node i. Then by Proposition 2.1 we can upper bound

$$e(C_L, C_R) \leq \rho \cdot \max\left\{ e(C_L, F_L) \ , \ \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i) \right\}$$

and obtain

$$2\sum_i e(C_L, C_R) \le 2\rho \cdot \left\{ \sum_i e(C_L, F_L) + \sum_i \tfrac{|C_L|}{|C_i|} \cdot e(C_i, F_i) \right\}. \tag{4.1}$$

Therefore, to complete the proof of Lemma 4.4 it suffices to upper bound the sums in the curly brackets (i.e., the total cost amortized in each of the two methods) by $e(W,B) \cdot O(\log n)$.

Consider first $\sum_i e(C_L, F_L)$. The edges that contribute to this sum are charged-free edges and hence edges of the bisection (W,B). An edge in the cut (C_L, F_L) must be inside $V_{L(i)}$, the smaller side of the cut of V_i, and any single edge can be inside $V_{L(i)}$ in at most $\log n$ divide steps i throughout the tree T. Hence, $\sum_i e(C_L, F_L)$ consists of at most $\log n$ times the cost of every edge of the bisection (W,B), and therefore this sum is at most $e(W,B) \cdot \log n$.

Next consider $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$ and recall our convention that $\frac{0}{0}$ is defined to be 0. The edges of $e(C_i, F_i)$ contribute to the sum their cost scaled by a factor of $\frac{|C_L|}{|C_i|}$. Each edge of $e(C_i, F_i)$ is a charged-free edge and hence an edge of the bisection (W,B). However, an edge of the bisection (W,B) belongs to $e(C_i, F_i)$ if and only if this edge is inside V_i. The nodes i for which this edge is inside V_i are all on a path from the root to a leaf of the decomposition tree T, and therefore the total contribution of this edge is at most its cost scaled by the sum of $\frac{|C_L|}{|C_i|}$ over that path in T.

We claim that the sum of $\frac{|C_L|}{|C_i|}$ over any path from the root to a leaf is bounded by $O(\log n)$. It follows from this claim that $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$ can be described as the cost of every edge of the bisection (W,B) scaled by at most $O(\log n)$, and therefore this sum is at most $e(W,B) \cdot O(\log n)$.

To prove the claim, consider an arbitrary path from the root to a leaf and denote the path nodes by $1, 2, \ldots, p+1$. At each node i the charged side (i.e., C_i) may be either W or B, depending on the label of the node; so denoting $w_j = |W \cap V_j|$ and $b_j = |B \cap V_j|$, we have that $\frac{|C_L|}{|C_i|}$ is either $\frac{w_{L(i)}}{w_i}$ or $\frac{b_{L(i)}}{b_i}$ and clearly at most their sum. Hence,

$$\sum_{i=1}^{p} \frac{|C_L|}{|C_i|} \le \sum_{i=1}^{p} \frac{w_{L(i)}}{w_i} + \sum_{i=1}^{p} \frac{b_{L(i)}}{b_i}.$$

First consider $\sum_1^p \frac{w_{L(i)}}{w_i}$, and observe that w_i is a nonincreasing sequence, since, in the tree, node i is a parent of node $i+1$. If node $i+1$ is a left child (of its parent node i), then $w_{L(i)} = w_{i+1}$ and hence $\frac{w_{L(i)}}{w_i} = \frac{w_{i+1}}{w_i} \le 1$. The number of such nodes i is at most $\log n$, since the path from the root to a leaf can contain at most $\log n$ left children i. (Recall that $|V_{L(i)}| \le |V_{R(i)}|$.) The contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is therefore at most $\log n$.

If node $i+1$ is a right child (of its parent i), then $w_{L(i)} = w_i - w_{i+1}$, and the contribution of all such nodes i is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i}$. Clearly, $\frac{w_i - w_{i+1}}{w_i} \le \frac{1}{w_i} + \cdots + \frac{1}{w_{i+1}+1}$ and hence the contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i} \le \frac{1}{w_1} + \cdots + \frac{1}{2} + 1 = H(w_1) \le H(n)$, where $H(k) = \sum_1^k \frac{1}{j}$ is the kth harmonic number.

We conclude that $\sum_1^p \frac{w_{L(i)}}{w_i} \le \log n + H(n) \le O(\log n)$. Similarly, $\sum_1^p \frac{b_{L(i)}}{b_i} = O(\log n)$, and together we get that $\sum_1^p \frac{|C_L|}{|C_i|} \le O(\log n)$, proving the claim and the lemma. ∎

COROLLARY 4.5. *The charge of the fixed optimal bisection* (W^*, B^*) *with respect to an opt-consistent labeling is at most* $b(1 + O(\rho \log n))$.

Distributing charge to vertices. It will be convenient (algorithmically) to distribute the charge of a bisection (W, B) (with respect to T and a labeling) to the vertices of the input graph, as follows. For each vertex $v \in V_i$ let the *cross-degree* of v at node i, denoted $cross_i(v)$, be the cost of the edges that are incident at v and are cut in divide step i. We define the *charge of a vertex* $v \in V$ as the sum of the cross-degree of v at all nodes i for which v belongs to the charged side, i.e., $\sum_{i:v \in C_i} cross_i(v)$. The next lemma proves that distributing the charge of a bisection to the graph vertices is indeed correct.

LEMMA 4.6. *The charge of a bisection* (W, B) *is the sum of the charges of all vertices in* G.

Proof. The charge of a divide step of node i is equal to the sum of the cross-degrees at node i of all vertices $v \in V_i$, i.e.,

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}) = \sum_{v \in C_i} cross_i(v) \ .$$

Summing over all nodes i in the tree T, the left-hand side is, by definition, the bisection charge, and the right-hand side is the sum of the charges of all vertices in G. The proof follows. ∎

Distributing the charge to the vertices of G is important algorithmically. The charge of a vertex depends on (and can be easily computed from) the side of this vertex in the bisection (W, B), the decomposition tree T, and the labeling of T, but it does not depend on the side of the cut (W, B) that other vertices of the graph belong to. It follows that the charge of a bisection (W, B), with respect to a given decomposition tree T and a labeling of it, depends *linearly* on the placement of vertices into W and B. This formulation of charge will be exploited by (the dynamic programming in) the combining stage.

4.4. Combining stage. The combining stage computes a bisection of the input graph G and a labeling of the decomposition tree T such that the bisection charge with respect to the labeling is at most $b \cdot (1 + O(\rho \log n))$. It then follows from Lemma 4.3 that the cost of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$, as desired.

First consider the case where an opt-consistent labeling is known. Then it suffices to compute a bisection of G whose charge with respect to this opt-consistent labeling is minimal because Corollary 4.5 guarantees that the charge of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$. Below we describe a simple procedure for finding a bisection of G with minimal charge with respect to a given labeling.

However, we do not know how to efficiently find an opt-consistent labeling, and therefore we go over all the labelings in the family $\mathcal{F}$. Specifically, using a more complicated procedure described below the combining stage finds a bisection of G and a labeling from $\mathcal{F}$ such that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs. Lemma 4.2 guarantees that at least one of these labelings is opt-consistent, in which case Corollary 4.5 applies. Hence, the bisection-labeling pair computed by this procedure satisfies that the charge of the bisection with respect to the labeling is indeed at most $b \cdot (1 + O(\rho \log n))$.

Minimizing charge over a given labeling. Finding a bisection of minimum charge with respect to a given labeling is relatively straightforward. By Lemma 4.6, the charge of a bisection (W, B) is the sum of the vertex charges. Since the decomposition tree T and the labeling are fixed, the charge of a vertex depends only on its side

in the bisection (W, B). We can therefore compute for each vertex v what its charge is when it belongs to W, called the *white charge* of v, and what its charge is when it belongs to B, called the *black charge* of v. (Note that summing the white charge and the black charge of a vertex gives the degree of that vertex in G.)

The charge of a bisection (W, B) is then the sum of the white charges of W and the black charges of B. To find a bisection (W, B) with minimum charge with respect to the given labeling, we can thus compute for each vertex its net-charge (white charge minus black charge) and take W to be the $n/2$ vertices with smallest net-charge. (This algorithm for the case where a labeling is given was used in the algorithm outline in section 2, where we assumed that the labeling stage produces an opt-consistent labeling.)

Minimizing charge over the family $\mathcal{F}$. The combining stage uses dynamic programming to find a bisection and a labeling from the family $\mathcal{F}$ so that the charge of the bisection with respect to this labeling is minimum over all such bisection-labeling pairs.

The dynamic programming table Q has entries of the form $Q(i, k, g)$, where i is a node of the decomposition tree T, k is an integer between 0 and $|V_i|$, and g is a *guess list* that contains the labels of the marked ancestors of node i. Throughout, i is considered an ancestor of itself.

An entry $Q(i, k, g)$ in the table contains the optimal solution to the following problem: Choose k vertices of V_i and a labeling from $\mathcal{F}$ that agrees with g so that when these k vertices are placed in the side W and the remaining vertices of V_i are placed in the side B, the sum of the charges of all the vertices of V_i with respect to the chosen labeling is minimal over all such choices. Note that when we consider only labelings from the family $\mathcal{F}$ that agree with g, the labels of all the ancestors of i are uniquely defined from g, while the marked descendants of i can have arbitrary labels.

For a leaf node i, the table entry $Q(i, k, g)$ can be computed directly, as follows. Since i is a leaf node, the part V_i consists of a single vertex, say v, and k can be either 0 or 1. If $k = 0$, then v is necessarily in B, and if $k = 1$, then v is necessarily in W. The guess list g gives the labels of all the nodes on the path from the leaf i to the root and hence all the labels that can possibly affect the charge of v. Since k and g uniquely define all the data that the charge of v depends on, $Q(i, k, g)$ is just the charge of v and can be computed directly as $\sum_j cross_j(v)$, where j ranges over all ancestors of i whose label (according to g) agrees with the side of v (as follows from k).

For a nonleaf node i, the table entry $Q(i, k, g)$ can be efficiently computed from table entries of its children nodes $L(i), R(i)$. Indeed, choosing k vertices from V_i is equivalent to choosing j vertices from one child part $V_{L(i)}$ and $k - j$ vertices from the other child part $V_{R(i)}$; so we need to add up two entries, each corresponding to one child node. The optimal value of j is not known, but it can be exhaustively searched. The guess list g can be extended into lists g_L, g_R for the children nodes in possibly more than one way. Therefore,

$$Q(i, k, g) = \min_{0 \le j \le k} \min_{g_L, g_R} \{Q(L(i), j, g_L) + Q(R(i), k - j, g_R)\},$$

where g_L, g_R range over all possible extensions of g, as described below. If a child node $L(i)$ is a marked node, then there are two possible ways to extend the list g into a list g_L (by adding a label for $V_{L(i)}$), and the optimum $Q(i, k, g)$ is achieved by taking the one which is better. If a child node $L(i)$ is not a marked node, then the only extension is $g_L = g$ because i and $L(i)$ have the same marked ancestors. The

possible extensions of the child node $R(i)$ are similar. It follows that each table entry of a nonleaf node i can be computed from table entries of its children $L(i), R(i)$ in time $O(|V_i|) = O(n)$.

To fill all the table entries, start from the entries that correspond to leaf nodes i and go upwards on the decomposition tree T. In particular, the entries $Q(i_{root}, n/2, g)$ will be computed for the root node i_{root}. At the root node, the guess list g contains the label of the root and thus has only two possible values. (In fact, the two entries must be the same due to symmetry.) The combining stage outputs $\min_g Q(i_{root}, n/2, g)$, which, by definition, is the minimum charge of all bisections of the input graph with respect to any labelings from $\mathcal{F}$, as desired. A bisection that achieves this minimum charge can also be computed. Simply go over the table entries in the reverse order of computation and recover at each entry the values of j, g_L, g_R that gave the optimum. Alternatively, associate with each entry $Q(i, k, g)$ a set of k vertices of V_i which is optimal for it and its corresponding labels.

LEMMA 4.7. *The combining stage finds in polynomial time a bisection of the input graph G and a labeling from the family $\mathcal{F}$ so that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs.*

Proof. The above discussion shows that the algorithm correctly computes every entry $Q(i, k, g)$, and a bisection-labeling pair as desired.

The size of the table Q is polynomial in n. Indeed, there are only $O(n)$ tree nodes i. For each tree node i, the range of k contains $O(|V_i|) = O(n)$ possible values. In addition, at each tree node i the guess list g contains labels of at most $O(\log n)$ ancestor nodes, and thus g assumes polynomially many values. The polynomial bound on the size of the table Q follows.

An entry for a leaf node i is computed efficiently. An entry for a nonleaf node is efficiently computed from previously computed entries. By the upper bound on the table size we conclude that all the table entries are computed in polynomial time, and in particular $Q(i_{root}, n/2, g)$. ∎

COROLLARY 4.8. *The combining stage finds bisection of the input graph (and a labeling of T) such that bisection charge (with respect to the labeling) is at most $b(1 + O(\rho \log n))$.*

Proof. By Lemma 4.7 and Corollary 4.5 there exists a bisection of G and a labeling of $\mathcal{F}$ such that the bisection charge with respect to the labeling is at most $b(1 + O(\rho \log n))$. The proof then follows by applying Lemma 4.2. ∎

This corollary completes the proof of Theorem 4.1, since by Lemma 4.3 the charge of a bisection is an upper bound on its actual cost.

5. Extensions. Our results extend to several variants (and generalizations) of the minimum bisection problem, including the case of edges with arbitrary nonnegative costs (section 5.1), the case of vertices with polynomially bounded nonnegative integer weights (section 5.2), the variant that requires, in addition, to separate a given pair of vertices s and t (section 5.3), the case of cutting away from the graph an arbitrary number of vertices (instead of $n/2$) that is given as part of the input (section 5.4), the case of cutting the input graph into a fixed number of equal-size parts (section 5.5), and the case of finding a 2/3-balanced cut whose cost is small relative to the minimum bisection cost b (section 5.6).

In what follows, the *basic bisection problem* refers to the minimum bisection problem that was defined in section 1. In contrast, the *extended bisection problems* refer to the variants of the problem specified above. We discuss each extended problem separately, but it is straightforward to combine together several extensions (e.g., to

allow both edge costs and vertex weights as described above and require that the total weight of the vertices cut away is a number k that is given in the input).

We consider two approaches for extending our approximation algorithm from the basic bisection problem to an extended problem. One approach is to *reduce* the extended problem to the basic one. Another approach is to *modify the algorithm* that we devised for the basic bisection problem so that it handles also the extended variant. As we discuss below, each approach has its own advantages and so it is valuable to show both approaches for each extended problem. We indeed show that for almost all the extended problems specified above both approaches can be applied, although for a few problems we provide only a modified algorithm.

A major advantage of the reduction approach is that it is self contained and not restricted to the particular algorithm that we devise; so future improvement in the approximation ratio for the basic problem may lead to an immediate improvement also for the extended problem. Most of our reductions transform an approximation ratio $f(n)$ for the basic problem into an approximation ratio $f(n^{O(1)})$ for the extended problem (because they increase the number of vertices n by a polynomial), and so for the current approximation ratio $f(n)$, which is polylogarithmic, these reductions increase the approximation ratio by at most a constant factor. The techniques used in our reductions are similar to those devised in [4, 3] for the (different) purpose of proving NP-hardness results.

The advantages of the algorithm modification approach are that it preserves aspects that are specific to our algorithm, such as an improved $O(\log n)$ approximation ratio for planar graphs, and that it is usually more efficient (and therefore practical) than the reduction approach. A drawback of the algorithm modification approach is that it requires going again through the algorithm's analysis. In particular, we might be required to verify that the approximate min-ratio cut algorithm (that we use as a black-box) can be extended accordingly. However, the necessary changes in the algorithm and its proof are usually straightforward.

5.1. Edge costs. Suppose that the edges of the input graph G have arbitrary nonnegative costs, and that the cost of a bisection is the total cost (i.e., sum of the costs) of its edges, and we wish to find a bisection of G of (approximately) minimum cost.

Reduction. We reduce the extended problem of bisection with edge costs (described above) to the basic bisection problem, as follows. Given a graph G with edge costs as an input, we first guess the most costly edge in a minimum cost bisection of G, by exhaustively trying all $O(n^2)$ edges in the input graph. By scaling all edge costs, we can assume, without loss of generality, that the cost of the guessed edge is n^2. It follows that the cost b of the optimum bisection is at least n^2 but smaller than n^4. We then round down all edge costs to their closest integer, which can decrease the cost of any bisection by at most $\binom{n}{2} \le b/2$ and therefore by a factor of at most 2. We next change to n^5 every edge cost that is larger than n^5, which does not affect the cost of nearly optimal bisections (i.e., whose original cost was within a ratio of roughly n from the minimum). Finally, we replace each vertex of the graph by a clique of size n^5 and each edge (u, v) of cost t by t unit cost edges placed arbitrarily between the clique of u and the clique of v. (Since $t < n^{10}$ we can do that with no parallel edges.)

The bisection of minimum cost b in G corresponds to a bisection of cost $\Theta(b)$ in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has n^6 vertices) yields a bisection whose cost is $O(b(\log n^6)^2) = O(b \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise

its cost will be at least $n^5 - 1 \gg b \log^2 n$, and it therefore must correspond to a bisection of G, whose cost is roughly the same, namely $O(b \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with edge costs, as follows. Rather than considering the number of edges we always consider their cost; e.g., $e(V_1, V_2)$ denotes the sum of the costs of the edges with one endpoint in V_1 and one endpoint in V_2. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Figure 3.3) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the edge costs, but known algorithms (e.g., due to [11]) provide this subroutine. Note also this algorithm's binary search (step 3) takes $O(M \log n)$ iterations, where M is the number of bits used to represent an edge cost, and so the running time is polynomial in the input size. The resulting approximation ratio is the same as for the basic problem, i.e., $O(\log^2 n)$.

5.2. Polynomial vertex weights. Suppose that the vertices of the input graph G have nonnegative integer weights that are bounded by a polynomial n^c (where n is the number of vertices in G), and let a bisection be a cut that separates half of the total weight (i.e., the sum of the weights) of the vertices of V. We wish to find a bisection of G of (approximately) minimum cost. Note that if the weights are allowed to be exponential in n, finding any bisection of the graph is equivalent to the partition (or subset-sum) problem and therefore NP-hard.

Reduction. We reduce the extended problem of bisection with vertex weights (described above) to the basic bisection problem, as follows. Given a graph G with vertex weights as an input, we replace each vertex of cost w in G by a clique of $\max\{1, w \cdot n^3\}$ unit weight vertices and replace each edge (u, v) in G by one edge placed arbitrarily between the clique of u and the clique of v. In addition, for each vertex of weight 0 in G we place in the graph a new isolated vertex of unit weight.

A bisection of minimum cost b in G corresponds to a bisection of the same cost b in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has at most n^{c+4} vertices) yields a bisection whose cost is $O(b(c+4)^2 \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise its cost will be at least $n^3 - 1 \gg b \cdot (c+4)^2 \log^2 n$. Furthermore, the vertices of the created cliques of size at least n^3 must be partitioned evenly by this bisection, as otherwise their partition deviates from an even one by at least n^3 (these clique sizes are multiples of n^3) which is much more than the total number of remaining vertices, $2n^2$. (Recall that we added isolated vertices for vertices of weight 0 in G.) The computed bisection of the resulting graph therefore corresponds to a bisection of G, whose cost is the same, namely $O(b(c+3)^2 \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with vertex weights, as follows. Rather than considering the number of vertices in a part we always consider their total weight; e.g., $r(S)$ denotes the cost of the cut $(S, V \setminus S)$ divided by the minimum between the weight of S and the weight of $V \setminus S$. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Figure 3.3) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the vertex weights, but known algorithms (e.g., due to [11]) provide this subroutine. Note also that in this algorithm's related graph G' (step 2) the capacity of an edge between a vertex $v \in V_1$ and the new vertex s is p times the weight of v_1. The resulting approximation ratio is the same as for the basic problem, i.e., $O(\log^2 n)$.

5.3. Separating two vertices from each other ($s-t$ cut). Suppose that the input graph G contains two special vertices, s and t, and we wish to find a bisection that separates s from t and has minimum cost. (Note that the converse restriction, namely that s, t will not be separated, is equivalent to merging them into one vertex of weight 2 and therefore follows from section 5.2.)

Reduction. We reduce the extended problem of a bisection that separates s from t to the extended problem of bisection with vertex weights (described in section 5.2), as follows. Given an input graph G with special vertices s, t as above, we let the vertices s, t have weights n and let all other vertices of G have weight 1. The total weight of s and t together is $2n$, while the total weight of all other vertices is $n-2$ (and thus smaller); so every bisection of the resulting graph must separate s from t. It follows that every bisection of the resulting graph corresponds to a bisection of G that separates s from t and has the same cost, and vice versa. We can therefore find a bisection of G that separates s from t and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of a bisection that separates s from t, as follows. We change the dynamic programming table Q of the combining stage so that every entry $Q(i,k,g)$ contains two solutions (if they exist); one solution with the k chosen vertices containing s but not t and the other solution with the k chosen vertices not containing any of s and t. Computing the table entries is straightforward, and the output of the algorithm is $\min_g Q(i_{root}, n/2, g)$, where the minimum is taken only over solutions that contain s and not t. The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e., $O(\log^2 n)$.

5.4. Cutting an arbitrary given number of vertices. Suppose that the input consists of a graph G and a number k, and we wish to find a minimum cost cut that separates exactly k vertices.

Reduction. We reduce the problem of cutting away a given number k of vertices to the problem of bisection with vertex weights (described in section 5.2), as follows. Given an input graph G and a number k (assume, without loss of generality, that $k \le n/2$), we let the vertices of G have weight 1 and add to the graph an isolated vertex of weight $n-2k$. It is clear that every bisection of the resulting graph corresponds to a cut of G that separates k vertices and has the same cost, and vice versa. We can therefore find a cut of G that separates k vertices and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of cutting a given number of vertices, as follows. The only change in the algorithm is in the combining stage, that now outputs $\min_g Q(i_{root}, k, g)$, where Q is the dynamic programming table (see section 4.4). The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e., $O(\log^2 n)$.

5.5. Cutting into a fixed number of parts. Suppose that we wish to find a cut that separates the input graph G into a fixed number p of parts of equal size.

We do not know of a reduction from this extended problem to the basic bisection problem. A recursive bisection approach has a poor performance in general, although it may be useful in some special cases and if some requirements are relaxed; see [15] and the references therein.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the problem of cutting the graph into p parts of equal size, as follows. The cost of a cut that partitions V into p parts $V^1, \ldots, V^p$ is

$$\sum_{j<l} e(V^j, V^l) = \frac{1}{2} \sum_j e(V^j, V \setminus V^j).$$

Therefore, by scaling the value of every possible solution by a factor of 2 (which clearly does not affect any approximation ratio issues), we obtain that the objective function of the extended problem has the convenient form $\sum_j e(V^j, V \setminus V^j)$. Observe that each cut $(V^j, V \setminus V^j)$ corresponds to separating V^j from the other parts, which are grouped into one part $V \setminus V^j$. Thus, each summand $e(V^j, V \setminus V^j)$ in the objective function is similar to the basic bisection problem (with the minor exception that the two sides are not of the equal sizes). Below we describe the modifications to the three stages of the algorithm, which works simultaneously on all p cuts $(V^j, V \setminus V^j)$. Its analysis is based on applying the new accounting method of section 2 separately to each of these p cuts.

The decomposition stage computes a decomposition tree T exactly as in the algorithm for the basic problem (see section 4.1). Observe that the amortized cut notion does not depend on the cut that we seek, and so the obtained decomposition (and its tree T) can be used for all cuts $(V^j, V \setminus V^j)$.

We extend the notion of a labeling of the decomposition tree, as follows. An extended labeling of T assigns to every tree node a vector of p "basic" labels, one label for each cut $(V^j, V \setminus V^j)$. An extended labeling corresponds to deciding at each tree node i and for each j, which of V^j and $V \setminus V^j$ is considered charged (and which is considered free) in the part V_i. Note that an extended labeling can be viewed as a vector, whose coordinate j forms a basic labeling for $(V^j, V \setminus V^j)$.

The labeling stage marks some nodes of the tree T exactly as in the algorithm for the basic problem (see section 4.2). This stage implicitly defines a family $\overline{\mathcal{F}}$ that consists of all extended labelings in which every unmarked node has the same label as its closest marked ancestor. (There is no restriction on the labels of the marked nodes.) It is straightforward that $\overline{\mathcal{F}}$ contains at least one extended labeling for which every coordinate j (forms a basic labeling that) is α-consistent with the cut $(V^j, V \setminus V^j)$. We can restrict the number of possible labels at the marked (and hence also unmarked) nodes from 2^p to $p+1$ values, as follows. Similar to the proof of Lemma 4.2 it is sufficient for our purposes that $\overline{\mathcal{F}}$ contains the labeling where V^j is considered free at a marked node i if more than half the vertices of the part V_i are from V^j. At any part V_i, the latter can happen for at most one value of j, and so it suffices to consider only labelings where at most one V^j is free.

We extend the notion of a charge of a vertex, as follows. The extended charge of a vertex v with respect to an extended labeling is the sum of the basic charges of v with respect to each of the p coordinates of this extended labeling.

The combining stage uses dynamic programming on a table $\overline{Q}$, whose entries are of the form $\overline{Q}(i, \overline{k}, \overline{g})$, as follows. i is a tree node. $\overline{k} = (k_1, \ldots, k_p)$, where k_j is the desired size of the jth part and $\sum_j k_j = |V_i|$. $\overline{g} = (g_1, \ldots, g_p)$, where g_j is a guess list that contains the jth label of every marked ancestor of i. An entry $\overline{Q}(i, \overline{k}, \overline{g})$ contains the optimal solution to the following problem: Choose a partition of V_i into subsets with sizes according to $\overline{k}$, and choose a labeling from $\overline{\mathcal{F}}$ that agrees with g so that the sum of the extended charges of all the vertices of V_i with respect to the chosen labeling is minimal over all such choices. Note that this problem requires

some correlation between p cuts, and therefore $\overline{Q}(i,\overline{k},\overline{g})$ is generally not equal to $\sum_j Q(i,k_j,g_j)$ (where Q is the basic table).

The rules for computing the entries of the table $\overline{Q}$ are a straightforward extension of those for the table Q (see section 4.4). The algorithm computes all the table entries and then outputs $\min_{\overline{g}} \overline{Q}(i_{root},\overline{k},\overline{g})$, where $\overline{k}=(n/p,\ldots,n/p)$.

The running time of this modified algorithm is polynomial in n (for fixed p). Indeed, the decomposition stage and the labeling stage are exactly as in the algorithm for the basic bisection problem; so let us consider the dynamic programming table $\overline{Q}$ of the combining stage. The number of tree nodes i is $O(n)$, and the range of $\overline{k}$ contains at most n^p possible values. The vector $\overline{g}$ contains one of $p+1$ possible values for each of the $O(\log n)$ marked ancestors (of the relevant tree node i); so $\overline{g}$ assumes one of $n^{O(\log p)}$ values. It follows that the size of the table $\overline{Q}$ is $n^{p+O(\log p)}$. Each table entry is computed efficiently from previously computed entries, and hence the combining stage takes polynomial time.

To analyze the approximation ratio, let $V^1,\ldots,V^p$ be the optimal partition of the input graph into p parts of equal size. Recall that the extended charge of a vertex is the sum of its basic charges with respect to each cut $(V^j, V\setminus V^j)$, and we can therefore apply the analysis of the basic algorithm for each cut $(V^j, V\setminus V^j)$ separately. It follows that the output value is guaranteed to be at most $O(\log^2 n)\cdot\sum_j e(V^j, V\setminus V^j)$. Furthermore, one can obtain from the table $\overline{Q}$ a cut (into p parts of equal size) whose cost is at most (half) this value, i.e., within a ratio of $O(\log^2 n)$ from the minimum.

5.6. Bicriteria approximation and balanced cuts. Suppose that we wish to find a 2/3-balanced cut (recall that a cut is called β-balanced if it partitions the graph into two parts, each of size at most βn) whose cost is guaranteed to be small relative to the minimum cost b of a bisection (i.e., a 1/2-balanced cut). Here, the minimum bisection problem is relaxed in two respects, as the solution cut is allowed to have cost larger than b and also to deviate from the cardinality constraints (for its two sides). Algorithms for such problems are sometimes referred to as bicriteria approximation and sometimes as pseudoapproximation.

Known bicriteria approximation algorithms find a 2/3-balanced cut whose cost is at most $O(b\log n)$. Leighton and Rao [10, 11] show how an algorithm that finds a τ approximate min-ratio cut can be used to find a 2/3-balanced cut of cost $O(b\tau)$; the approximation ratio $\tau=O(\log n)$ that they achieve is the best currently known; see also [14]. Even et al. [5] devise a different algorithm that also finds a 2/3-balanced cut of cost $O(b\log n)$.

We show below that amortized cuts can be used to obtain also bicriteria approximation algorithms (in addition to approximation algorithms) for minimum bisection. In fact, our algorithm is similar to the one of [10, 11], except that we use amortized cuts instead of approximate min-ratio cuts.

LEMMA 5.1. *An algorithm that finds a ρ-amortized cut can be used to find a 2/3-balanced cut of cost $b(1+O(\rho))$.*

Proof. Given an input graph $G(V,E)$ on n vertices, use the algorithm that finds a ρ-amortized cut, as follows. Repeatedly find (in the graph) a ρ-amortized cut and remove (from the graph) the smaller of its two sides until the graph contains no more than $2n/3$ vertices. Denoting by S the set of vertices that remain in the graph after the last iteration, output the cut $(S, V\setminus S)$.

It is straightforward to see that $n/3<|S|\le 2n/3$, and hence the output cut $(S,V\setminus S)$ is a 2/3-balanced cut. We prove below that the total cost of all edges cut

by the amortized cuts (throughout the iterations) is at most $b(1 + O(\rho))$. It would then follow immediately that $e(S, V \setminus S) \leq b(1 + O(\rho))$, as required.

We now upper bound the total cost of all edges cut in the amortized cuts. Let (W, B) be a fixed optimal bisection of cost b and call the vertices of W white and the vertices of B black. The total cost of white-black edges cut is clearly at most b. We show below that the total cost of all white-white edges cut is $O(b\rho)$. By the symmetry between W and B, we will then have a similar upper bound on the total cost of the black-black edges cut and obtain the desired upper bound of $b(1 + O(\rho))$ on the total cost of all edges cut.

To show that the total cost of white-white edges cut in the amortized cuts is $O(b\rho)$, we consider the white vertices W as charged in all the amortized cuts, and then white-white edges are charged-charged edges. The algorithm applies a ρ-amortized cut in parts of G that contain at least $2n/3$ vertices. At least $n/2 - n/3 = n/6$ of the vertices in such a part are black, while at most $n/2$ of them are white, and hence at most $3/4$ of the vertices in this part are considered charged. Taking a constant $\alpha \geq 3/4$ in the definition of an amortized cut, we have that the cost of the charged-charged edges cut can be amortized in one of two amortization methods (see section 2).

In one amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the smaller side of the cut, with amortized cost at most ρ. Observe that an edge can be in the smaller side of the amortized cut (the side that is removed) in at most one iteration; so the total cost amortized in this method (in all the iterations) against one charged-free edge is at most ρ. Hence, the total cost amortized in this method (in all the iterations) is at most $b\rho$.

In the other amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the part being divided, with amortized cost at most $\rho|C_1|/|C|$, where C denotes the charged vertices in the part being divided and C_1 denotes the charged vertices in the smaller side of the cut. The total cost amortized in this method (in all the iterations) against one charged-free edge is then upper bounded by ρ times the sum of $|C_1|/|C|$ over all iterations. Recall that the charged vertices are the white vertices, and so $|C| \geq n/6$ in all amortized cuts (i.e., iterations). Furthermore, each vertex is in the smaller side of the cut (the side that is removed) in at most one iteration, and so the sum of $|C_1|$ over all iterations is at most $n/2$. It follows that the total cost amortized in this method (in all the iterations) against one charged-free edge is at most 3ρ, and hence the total cost amortized in this method is at most $b \cdot 3\rho$.

We conclude that the total cost of all charged-charged (i.e., white-white) edges cut in all the iterations is at most $b \cdot 4\rho$. As described above, this proves that the total cost of all edges cut in all the iterations is at most $b(1 + 8\rho) = b(1 + O(\rho))$, and the lemma follows. □

We remark that a 2/3-balanced cut of cost $b(1 + O(\rho))$ can be found also by modifying the algorithm we devised for the basic bisection problem so that its combining stage outputs $\min_{g, n/3 \leq k \leq n/2} Q(i_{root}, k, g)$ (and its corresponding cut). Indeed, the proof of Lemma 5.1 shows a 2/3-balanced cut whose charge (with respect to a certain labeling in $\mathcal{F}$) is at most $b(1 + O(\rho))$. Details are omitted.

Concluding remarks. Designing an algorithm that finds a cut of amortized cost better than $O(\log n)$ remains an important open question. An efficient algorithm that accomplishes that will not only improve the approximation ratio for minimum bisection (by Theorem 4.1), but also the bicriteria approximation ratio for minimum bisection (by Lemma 5.1), which will lead, in turn, to improved approximation ratios

for many other problems; see [11, section 3].

Finding a cut whose amortized cost is better than $O(\log n)$ is, in a sense, no harder (and possibly easier) than approximating min-ratio cuts within a ratio better than $O(\log n)$, as the former problem is reducible (by Theorem 3.6) to the latter. Furthermore, an $O(1)$-amortized cut always exists (by Corollary 3.3), and we know of no hardness result for the problem of finding such a cut.

Acknowledgments. We thank Kobbi Nissim for his part in bootstrapping this research and the anonymous referees for comments that improved the presentation.

REFERENCES

[1] S. Arora, D. Karger, and M. Karpinski, *Polynomial time approximation schemes for dense instances of NP-hard problems*, J. Comput. System Sci., 58 (1999), pp. 193–210.

[2] Y. Aumann and Y. Rabani, *An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm*, SIAM J. Comput., 27 (1998), pp. 291–301.

[3] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser, *Graph bisection algorithms with good average case behavior*, Combinatorica, 7 (1987), pp. 171–191.

[4] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Inform. Process. Lett., 42 (1992), pp. 153–159.

[5] G. Even, J. Naor, S. Rao, and B. Schieber, *Fast approximate graph partitioning algorithms*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 1997, pp. 639–648.

[6] U. Feige, R. Krauthgamer, and K. Nissim, *Approximating the minimum bisection size*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 530–536.

[7] M. R. Garey, D. S. Johnson, and L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.

[8] N. Garg, H. Saran, and V. V. Vazirani, *Finding separator cuts in planar graphs within twice the optimal*, SIAM J. Comput., 29 (1999), pp. 159–179.

[9] P. Klein, S. A. Plotkin, and S. Rao, *Excluded minors, network decomposition, and multicommodity flow*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 682–690.

[10] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 422–431.

[11] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[12] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.

[13] H. Saran and V. V. Vazirani, *Finding k cuts within twice the optimal*, SIAM J. Comput., 24 (1995), pp. 101–108.

[14] D. Shmoys, *Cut problems and their applications to divide-and-conquer*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS, Boston, 1997.

[15] H. D. Simon and S.-H. Teng, *How good is recursive bisection?*, SIAM J. Sci. Comput., 18 (1997), pp. 1436–1445.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1119–1155

LATTICE EMBEDDINGS FOR ABSTRACT BOUNDED REDUCIBILITIES*

WOLFGANG MERKLE†

Abstract. We give an abstract account of resource-bounded reducibilities as exemplified by the polynomially time- or logarithmically space-bounded reducibilities of Turing, truth-table, and many-one type. We introduce a small set of axioms that are satisfied for most of the specific resource-bounded reducibilities appearing in the literature. Some of the axioms are of a more algebraic nature, such as the requirement that the reducibility under consideration is a reflexive relation, while others are formulated in terms of recursion theory and, for example, are related to delayed computations of arbitrary recursive sets.

The main technical result shown is that for any reducibility that satisfies these axioms, every countable distributive lattice can be embedded into any proper interval of the structure induced on the recursive sets. This extends a corresponding result for polynomially time-bounded reducibilities due to Ambos-Spies [*Inform. and Control*, 65 (1985), pp. 63–84], as well as a result on embeddings of partial orderings for axiomatically described reducibilities due to Mehlhorn [*J. Comput. System Sci.*, 12 (1976), pp. 147–178].

Key words. resource-bounded reducibilities, embeddings of partial orderings, embeddings of distributive lattices, abstract reducibilities, axiomatization of reducibilities

AMS subject classifications. 03D15, 03D30, 03D45, 03D75, 68Q15

PII. S0097539701385958

1. Introduction.

1.1. Overview and related work. Resource-bounded reducibilities such as the standard time- or space-bounded reducibilities are usually defined as appropriate restrictions of Turing reducibility; i.e., a set A is reducible to a set B iff there is some index e with $A = \Phi_e(B)$ such that e and B satisfy certain conditions. (Here Φ_e denotes the partial recursive functional computed by the eth oracle Turing machine M_e.) Standard examples for such conditions are given by restrictions on the way M_e might access its oracle, such as for reducibilities of many-one type, and by bounds on the amount of time, space, or other resources M_e might use. In most cases such restrictions can be conveniently expressed by specifying a set E of admissible indices as in Definition 1.

DEFINITION 1. *A binary relation $\leq_r$ on 2^ω is a* BOUNDED REDUCIBILITY *iff there is a recursive set E that contains only indices of total recursive functionals such that for all sets A and B*

$$A \leq_r B \quad \textit{iff} \quad \exists e \in E\ A = \Phi_e(B). \tag{1}$$

The concept of bounded reducibility indeed comprises most of the resource-bounded reducibilities that can be found in the literature, including the usual time- or space-bounded reducibilities. The notation bounded reducibility apparently has been introduced by Book, Lutz, and Wagner [8], while the concept has been used before by several authors.

*Received by the editors March 7, 2001; accepted for publication (in revised form) November 22, 2001; published electronically April 12, 2002.
http://www.siam.org/journals/sicomp/31-4/38595.html
†Universität Heidelberg, Mathematisches Institut, Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany (merkle@math.uni-heidelberg.de).

When investigating the structures induced on the class REC of recursive sets by the various bounded reducibilities [2, 25], one typically asks whether such a structure is dense or whether it contains minimal pairs. Recall that the structure is dense iff every proper interval contains a set different from the interval's endpoints and, assuming that $\leq_r$ is a partial preordering with least element, that the structure $(\mathrm{REC}, \leq_r)$ contains a minimal pair if there are two incomparable recursive sets such that every set that is reducible to both sets is a least set.

Early results obtained for the structures $(\mathrm{REC}, \leq_{\mathrm{T}}^{\mathcal{P}})$ and $(\mathrm{REC}, \leq_{\mathrm{m}}^{\mathcal{P}})$ evolving from the polynomially time-bounded Turing and many-one reducibility,[1] respectively, are the density of the structures and the existence of minimal pairs, both due to Ladner [13]. Subsequently, Machtey [16] constructed a minimal pair of sets computable in exponential time, and Landweber, Lipton, and Robertson [15] further improved on this: actually any recursive set strictly above $\emptyset$ bounds a minimal pair. In what follows these results were extended, and the presentation of their proofs was substantially improved by several authors, including Mehlhorn [17, 18], Chew and Machtey [11], Balcázar and Díaz [3], and Schöning [30, 31]. Then Ambos-Spies [1] showed a general embedding theorem that comprises several preceding results as special cases: if $\leq_r$ is equal to $\leq_{\mathrm{T}}^{\mathcal{P}}$, to $\leq_{\mathrm{m}}^{\mathcal{P}}$, or to one of several variants of polynomially time-bounded truth table reducibility $\leq_{\mathrm{tt}}^{\mathcal{P}}$, then

(2)

> every countable distributive lattice can be embedded (as a lattice) into any proper interval of $(\mathrm{REC}, \leq_r)$ with least or greatest element preserved.

From (2) we obtain, for example, Ladner's density result by embedding the three-element total ordering into a given proper interval, and the existence of minimal pairs below a recursive set B that is strictly above $\emptyset$ follows by embedding the four-element Boolean algebra below B with least element preserved.

Ladner [13] already pointed out that his results go through for rather general types of time- or space-bounded reducibilities. Subsequently, also the stronger results mentioned were transferred to bounded reducibilities that are not defined in terms of deterministic polynomial time.

(i) Serna [32] proves that $(\mathrm{REC}, \leq_{\mathrm{NC}_1})$ is dense and possesses minimal pairs below any set strictly above $\emptyset$.

(ii) Copestake [12] demonstrates for polynomially time-bounded nondeterministic reducibility of Turing type that the corresponding structure induced on the recursive sets is dense and possesses minimal pairs.

(iii) Vollmer [34] extends the results on lattice embeddings due to Ambos-Spies to the reducibility $\leq_{\mathrm{m}}^{\log}$.

The fact that several bounded reducibilities bear similar structural properties and, what is more, that the same proof techniques apply in the different cases suggest that to some extent nontrivial structural properties of bounded reducibilities might be developed within an abstract or axiomatic approach to bounded reducibilities. In general, however, even for a reflexive and transitive bounded reducibility the structure induced on the recursive sets will not be dense; see Example 7 below. This indicates that there is no hope for deriving interesting structural results about bounded reducibilities without adding further conditions or axioms that, for example, provide,

[1] We introduce the specific reducibilities used as examples by informal descriptions. Full definitions can be found in the textbooks by Balcázar, Díaz, and Gabarró [4, 5] and by Odifreddi [24, 25].

intuitively speaking, a minimal amount of computational power that can be used in reducing one set to another.

We will pursue an abstract approach to resource-bounded reducibilities that is based on the concept of standard reducibility introduced in Definition 17 below. As the main technical result we show in section 4 that the mentioned result on lattice embedding extends to all standard reducibilities.

Archetypal examples of standard reducibilities on 2^ω are polynomially time-bounded Turing reducibility $\leq_{\mathrm{T}}^{\mathcal{P}}$ and logarithmically space-bounded many-one reducibility $\leq_{\mathrm{m}}^{\log}$. On the other hand, some bounded reducibilities that arise from more restricted models of computations are not standard reducibilities. For examples of such reducibilities see the discussion subsequent to the introduction of the concept of standard reducibility in Definition 17.

Axiomatic approaches to structural properties of bounded reducibilities have been presented previously by, among others, Basu [6], Mehlhorn [17, 18], Mueller [23], and Schmidt [29]. Our generalized approach is closely related to the work of the latter three authors, while Basu's axiom system is designed to be used in connection with reducibilities between functions and is too general if applied to reducibilities between sets. In particular, the work of Mehlhorn has been most influential to our approach via his concept of delayed simulation. Mehlhorn proves from a set of axioms, which apparently in several respects is more restrictive than ours, that the recursive sets form a dense structure, and he states that in fact every countable partial ordering can be embedded into every proper interval of the recursive sets.

Within the axiomatic approach based on the concept of standard reducibility, one cannot derive only the result on lattice embeddings shown below but also results on minimal and exact pairs or on distributivity and decidability of the structure induced on the recursive sets [19, 20]. In the context of separations by random oracles, almost classes, and bounded error probabilistic classes, results can be derived from assumptions that are more general than the ones used to define standard reducibilities; see Book, Lutz, and Wagner [8], Book, Vollmer, and Wagner [9], Merkle and Wang [22], and Regan and Royer [27].

1.2. Notation. The notation introduced in the following is mostly standard. For notation not explained here or below in the text, see Balcázar, Díaz, and Gabarró [4, 5], Odifreddi [24, 25], and Soare [33].

Natural numbers and strings. We identify the set $\omega = \{0, 1, \dots\}$ of natural numbers and the set $\{\lambda, 0, 1, 00, 01, \dots\}$ of (finite, binary) strings via the unique order isomorphism that takes the standard ordering on ω to the length-lexicographical ordering on strings, and we denote both orderings by the symbol $\leq$. We extend the identification in the canonical way to the powerset 2^ω of ω and the powerset of the set of all strings. Recall that resource-bounded reducibilities are usually defined in terms of Turing machine models where strings are used as inputs and for querying the oracle, and consequently these reducibilities are binary relations between sets of strings; by the above identification, we will view such reducibilities as binary relations on 2^ω. We refer to subsets of ω and 2^ω by the terms SETS and CLASSES, respectively. We denote sets by uppercase letters $A, B, \dots$, and classes by uppercase calligraphic letters $\mathcal{A}, \mathcal{B}, \dots$.

Functions and functionals are meant to be total, if not explicitly attributed as being partial. We denote the class of functions from ω to ω by ω^ω. We identify subsets of ω with their characteristic functions; i.e., we view 2^ω as a subclass of ω^ω.

Reducibilities. The symbol $\leq_r$ denotes any binary relation between sets of natural numbers, which is meant as reducibility. We will define the usual concepts arising in connection with reducibilities such as the LOWER and UPPER $\boldsymbol{\leq_r}$-CONE,

$$\leq_r (A) := \{X : X \leq_r A\} \quad \text{and} \quad \geq_r(A) := \{X : A \leq_r X\},$$

respectively, of a set A. Frequently, we will use the term r-cone in place of $\leq_r$-cone, and we will proceed similarly for other terms and reducibilities.

Two sets A and B are $\boldsymbol{r}$-EQUIVALENT, $A \equiv_r B$ for short, iff their upper and lower r-cones coincide, respectively. Hence two sets are r-equivalent iff, intuitively speaking, they can be substituted for each other *salva veritate* in all contexts involving only the relation $\leq_r$. Two sets A and B are $\boldsymbol{r}$-INTERREDUCIBLE, $A =_r B$ for short, iff A is reducible to B and vice versa. For a reflexive relation $\leq_r$, any pair of equivalent sets is also interreducible, and, likewise, for a transitive relation, interreducibility implies equivalence. As a consequence, interreducibility and equivalence coincide for partial preorderings, i.e., for relations that are reflexive and transitive.

Partial recursive functions and functionals. Unless we explicitly refer to some other domain, say, to ω^ω, functionals are functions from 2^ω to 2^ω. We denote functionals by uppercase Greek letters $\Gamma, \Delta, \ldots$. We identify a functional Γ with a function from $2^\omega \otimes \omega$ to $\{0,1\}$ via the equation

$$\Gamma(X, x) \;=\; (\Gamma(X))(x).$$

We use the notation M_i in order to refer to the ith Turing machine in the standard enumeration of all Turing machines of some given type, and we assume that it is always understood from the context which type of Turing machine is meant, that is, for example, whether we consider Turing machines with or without oracle access. We refer to the partial recursive function or functional computed by Turing machine M_i by

φ_i in the case of $\{0,1\}$-valued Turing machines without oracle access,
ϕ_i in the case of ω-valued Turing machines without oracle access,
Φ_i in the case of $\{0,1\}$-valued oracle Turing machines.

We refer to the class of recursive sets and functions by REC and FREC, respectively. We assume that there is some recursive function s that translates indices with respect to the enumeration $\varphi_0, \varphi_1, \ldots$ into indices with respect to the enumeration $\phi_0, \phi_1, \ldots$, that is, such that for all e in ω holds $\varphi_e = \phi_{s(e)}$.

Partial characteristic functions. Lowercase Greek letters $\alpha, \beta, \gamma, \ldots$ denote PARTIAL CHARACTERISTIC FUNCTIONS, i.e., (total) functions from some subset I of ω to $\{0,1\}$. The DOMAIN of a partial characteristic function α is denoted by $\mathrm{dom}(\alpha)$; i.e., for example, $\mathrm{dom}(\alpha)$ is equal to ω iff α is a set. A partial characteristic function is FINITE iff its domain is finite. The partial characteristic functions are partially ordered by the relation $\sqsubseteq$ where $\alpha \sqsubseteq \beta$ iff the graph of α is contained in the graph of β (i.e., iff the domain of α is contained in the domain of β and α agrees there with β).

For a partial characteristic function α and a set I, we denote by $\alpha | I$ the RESTRICTION OF $\boldsymbol{\alpha}$ TO $\boldsymbol{I}$, that is, the uniquely determined partial characteristic function $\gamma \sqsubseteq \alpha$ with domain $I \cap \mathrm{dom}(\alpha)$. In particular, $A|I$ is the partial characteristic function with domain I that agrees there with the set A.

Definition by cases and patching. For partial characteristic functions α, β, and for a set M, we let

$$\langle \alpha, \beta \rangle^M(x) \;:=\; \begin{cases} \alpha(x), & x \in M, \\ \beta(x), & x \notin M; \end{cases} \tag{3}$$

that is, the partial characteristic function $\langle \alpha, \beta \rangle^M$ agrees with α on $\mathrm{dom}(\alpha) \cap M$, with β on $\mathrm{dom}(\beta) \cap \overline{M}$, and is undefined otherwise. We denote the partial characteristic function

$$\langle \alpha, \beta \rangle \ := \ \langle \alpha, \beta \rangle^{\omega \setminus \mathrm{dom}(\beta)}$$

as $\boldsymbol{\beta}$-PATCH of α; i.e., for example, $\langle A, \beta \rangle$ is the unique set that agrees with β for all arguments in $\mathrm{dom}(\beta)$ and with A otherwise.

Joins. We define the join of two sets A and B by

$$A \oplus B(x) := \begin{cases} 0 & \text{if } x = \lambda, \\ A(y) & \text{if } x = 0y, \\ B(y) & \text{if } x = 1y. \end{cases}$$

Lattices. A partial ordering (p.o.) is a pair of a set U and a binary relation $\leq$ on U that is reflexive, transitive, and antisymmetric. A p.o. $(U, \leq)$ is a LATTICE iff for every pair a and b of elements in U there exists a least upper bound (l.u.b.) and a greatest lower bound (g.l.b.) of a and b in U. A lattice L is distributive iff for all a, b, and c in L we have $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.

Closure under finite variation. A set A is a FINITE VARIATION of a set B, $A =^* B$ for short, iff A and B differ at most at finitely many places. A subclass $\mathcal{A}$ of 2^ω is CLOSED UNDER FINITE VARIATION (C.F.V.) iff for every set A in $\mathcal{A}$, all finite variations of A are in $\mathcal{A}$, too. A binary relation $\leq_r$ on 2^ω is c.f.v. iff for all sets A, the lower and the upper r-cone of A are both c.f.v.

We let $\langle .,. \rangle : \omega^2 \to \omega$ be the standard effective and effectively invertible pairing function from $\omega \otimes \omega$ onto ω [33]. A subclass $\mathcal{A}$ of 2^ω is RECURSIVELY PRESENTABLE iff either $\mathcal{A}$ is empty or there is a recursive set E such that $\mathcal{A}$ coincides with the "rows" of E; i.e., $\mathcal{A} = \{A_0, A_1, \dots\}$ where $A_i = \{x : \langle x, i \rangle \text{ in } E\}$.

2. Standard reducibilities.

2.1. Faithful relations. In order to be able to mimic the usual proof techniques employed in connection with resource-bounded reducibilities, we will require that standard reducibilities are bounded reducibilities that are sort of "faithful" to the information contained in their set arguments and, intuitively speaking, can use a nontrivial amount of computational power in reducing one set to another. In the remainder of this section, we introduce notation related to the concept of faithfulness, and in the following sections we develop concepts related to computational power.

In the case of a transitive relation $\leq_r$, the concept of faithfulness is equivalent to the natural conditions that $\leq_r$ is reflexive, that $\emptyset$ and ω are r-reducible to all sets, and that the join of two sets is a l.u.b. for them (see Proposition 4). However, as we want to comprise reducibilities that are not necessarily transitive, we use a more involved definition of faithfulness stated in Definition 3.

DEFINITION 2. *Let $\leq_r$ be a binary relation on 2^ω, let A and B be sets, and let $\mathcal{A}$ be a subclass of 2^ω.*

(i) *The relation $\leq_r$ is* LOCALLY TRANSITIVE *at A and B iff A is not r-reducible to B or we have for all sets X and Y*

$$X \leq_r A \ \textit{implies} \ X \leq_r B, \tag{4}$$

$$B \leq_r Y \ \textit{implies} \ A \leq_r Y. \tag{5}$$

(ii) *A set U is a* LOCALLY TRANSITIVE UPPER BOUND *for (the elements of) $\mathcal{A}$ iff U is an upper bound for $\mathcal{A}$ and for all sets A in $\mathcal{A}$, the relation $\leq_r$ is locally transitive at A and U. Likewise, U is a* LOCALLY TRANSITIVE L.U.B. *for $\mathcal{A}$ iff U is a l.u.b. for $\mathcal{A}$ and for all sets A in $\mathcal{A}$ the relation $\leq_r$ is locally transitive at A and U.*

A transitive relation is locally transitive at every pair of sets; hence for a transitive relation the standard and the locally transitive variant of the concept of upper bound coincide, and a similar remark holds for l.u.b.'s.

DEFINITION 3. *A binary relation $\leq_r$ on 2^ω is* FAITHFUL *iff for all sets A, B, and X, first, the set $A \oplus B$ is a locally transitive l.u.b. for A and B in $(2^\omega, \leq_r)$ and, second,*

$$X \leq_r A \oplus B \text{ implies } X \leq_r B \oplus A, \tag{6}$$

$$X \leq_r A \oplus \emptyset \text{ implies } X \leq_r A, \tag{7}$$

$$X \leq_r A \oplus \omega \text{ implies } X \leq_r A. \tag{8}$$

Most resource-bounded reducibilities that can be found in the literature are faithful. There are nontransitive faithful reducibilities such as $\leq_{k-\mathrm{tt}}$ for every fixed $k \geq 2$. The following proposition shows that faithful relations are reflexive and for them $\emptyset$ and ω are reducible to all other sets.

PROPOSITION 4. *Let $\leq_r$ be a faithful relation on 2^ω.*

(i) *For all sets A, the sets $\emptyset$, ω, and A are r-reducible to A.*

(ii) *For all sets A and B, the set $A \oplus B$ is a l.u.b. for A and B in $(2^\omega, \leq_r)$.*

Conversely, if the relation $\leq_r$ is transitive and satisfies both of these conditions, then $\leq_r$ is faithful.

Proof. For a faithful relation $\leq_r$, condition (i) is immediate; so it remains to show (ii). Now, given a set A, then the sets A and $\emptyset$ are both r-reducible to $A \oplus \emptyset$; hence by (7) both sets are also r-reducible to A. Likewise, by (8) we obtain $\omega \leq_r A$. Conversely, assume that the relation $\leq_r$ is transitive and satisfies (i) and (ii) . By transitivity and (ii) it is immediate that the join operator provides locally transitive l.u.b.'s for every pair of sets. Furthermore, we obtain (6), (7), and (8) by transitivity of $\leq_r$ and because by assumption on the join operator we have

$$A \oplus B \leq_r B \oplus A, \qquad A \oplus \emptyset \leq_r A, \qquad \text{and} \qquad A \oplus \omega \leq_r A. \qquad \square$$

PROPOSITION 5. *Let $\leq_r$ be a faithful relation on 2^ω and let A and B be sets.*

(i) $\geq_r(A \oplus B) = \geq_r(A) \cap \geq_r(B)$.

(ii) $A \oplus B \equiv_r B \oplus A$.

(iii) $A \equiv_r A \oplus \emptyset \equiv_r A \oplus \omega$.

Proof. For a faithful relation $\leq_r$, the join of two sets is a locally transitive l.u.b. for the sets joined. Thus, concerning (i), the inclusion from left to right is immediate from the definition of locally transitive upper bound, and the reverse inclusion follows by definition of l.u.b. Using (i), we then infer that the two sets denoted by the join expressions on both sides of the equation in (ii) have identical upper cones, while for the lower cones this is immediate from (6). Concerning (iii), it is sufficient to show that the lower (respectively, upper) cones of A, $A \oplus \emptyset$, and $A \oplus \omega$ are identical. We show this for the first two sets and omit the almost identical considerations for the third set. By (i), the upper cone of $A \oplus \emptyset$ is equal to the intersection of the upper cones of A and $\emptyset$; hence it is equal to the upper cone of A. On the other hand, by (7) the lower cone of $A \oplus \emptyset$ is contained in $\leq_r (A)$, and the reverse containment holds because $A \oplus \emptyset$ is a locally transitive upper bound for A. $\square$

REMARK 6. *Fix an arbitrary faithful relation $\leq_r$. Then the fact $A \leq_r B$ implies that B and $A \oplus B$ are r-interreducible but in general does not imply that these two sets are r-equivalent. In fact, the second implication holds iff $\leq_r$ is transitive.*

For a proof, observe that B is always reducible to $A \oplus B$, while in case $A \leq_r B$, the l.u.b. $A \oplus B$ of A and B must be reducible to their upper bound B. Thus the sets B and $A \oplus B$ are interreducible and, by the discussion in section 1.2, *are thus in fact equivalent in case $\leq_r$ is indeed transitive. Conversely, assume that B and $A \oplus B$ are always equivalent if A is reducible to B. Then for given sets A, B, and Y such that $A \leq_r B \leq_r Y$, we infer that $A \oplus B$ is reducible to Y and hence, because the former set is a locally transitive upper bound for A, A is also reducible to Y. As a consequence, the relation $\leq_r$ is transitive.*

In Example 7, we construct a faithful bounded reducibility that is in addition transitive and c.f.v. but where the corresponding structure induced on the recursive sets is not dense. The example shows that in order to be able to derive the density of the recursive sets within our axiomatic framework, we have to add further assumptions on the reducibilities under consideration.

EXAMPLE 7. *Given a set A and a string w, for the scope of this example let $A^{<w>}$ be equal to the set $\{x : wx \in A\}$ and call a set A β-reducible to a set B, for short, $A \leq_\beta B$ iff for some n in ω there is a mapping*

$$r : \{0,1\}^n \to \{0,1\}^* \tag{9}$$

such that for all strings w of length n,

$$A^{<w>} \text{ is a finite variation of } \emptyset,\ \omega,\ B^{<r(w)>},\ \text{or } \overline{B^{<r(w)>}}. \tag{10}$$

So β-reducibility is of a rather restricted one-question truth-table type. For each string w of length n and for almost all places wx that extend w, the same one-place evaluation function (constant 0 *or* 1, *identity, or negation) is applied to the answer received on querying the oracle at place $r(w)x$. We omit the routine proof that β-reducibility is a faithful bounded reducibility that is transitive and c.f.v. Next we argue that the set*

$$T := \{1^k : k \in \omega\}$$

is minimal in $(2^\omega, \leq_\beta)$; hence, in particular, the structure $(\mathrm{REC}, \leq_\beta)$ is not dense.

By definition, the set T is minimal if it is not a least set, and every set A that is β-reducible to T either is a least set or T is β-reducible to A. First, the set T is not β-reducible to $\emptyset$. Given a function r that witnesses that some set X is β-reducible to $\emptyset$ where r is defined on strings of length n, then the set X contains either almost all or only finitely many strings of the form $1^n y$. Second, assume $A \leq_\beta T$ for a set A, and let this fact be witnessed by a function r as in (9) *that is defined on strings of length n. If there is some string w of length n where $r(w)$ is in T and the evaluation function is nonconstant, then, by choice of T, $A^{<w>}$ is some finite variation of either the set T or of its complement, depending on the evaluation function being identity or negation. So in this case, T is β-reducible to A. On the other hand, if for all strings w of length n the evaluation function is constant or $r(w)$ is not in T, then A is β-reducible to $\emptyset$. Hence for every set A that is β-reducible to T, we have either $T \leq_\beta A$ or $A \leq_\beta \emptyset$.*

2.2. Delayed simulations and delayed patching. In this section, we describe an abstract account of the ability of resource-bounded oracle Turing machines to overwrite or "patch" their oracle according to the results of resource-bounded subcomputations. This account is based on the concept of *delayed patching*. By patching a functional we refer to evaluating the functional not with respect to its set argument B but with respect to a patched version $\langle B, \sigma \rangle$ of B. Furthermore, the attribute delayed refers to the fact that the patching is done with respect to arbitrary effective enumerations $\alpha_0, \alpha_1, \ldots$ of finite partial characteristic functions where, however, in general the ith partial characteristic function will not be used while computing the value $\Gamma(B, i)$, but delayed, that is, for number arguments larger than i. Note again that this kind of delayed access to effectively given information is common in connection with resource-bounded oracle Turing machines. Given an effective enumeration $\alpha_0, \alpha_1, \ldots$ as above, a resource-bounded oracle Turing machine can eventually compute and access α_i for arbitrarily large values of i; however, intuitively speaking, the Turing machine has to wait until its number input and hence its resource-bounds become large enough. The ability to perform such delayed computations is modelled by the concepts delayed simulation and simulation class introduced in Definition 8.

DEFINITION 8. *Let h, s, and l be (not necessarily recursive) functions from ω to ω.*

(i) *The function s is* MANY-ONE REDUCIBLE *to h via l iff for all x in ω, we have $s(x) = h(l(x))$.*

(ii) *The function s is a* DELAYED SIMULATION *of the function h iff s is many-one reducible to h via some nondecreasing function l with range ω.*

(iii) *A subclass $\mathcal{F}$ of ω^ω is a* FUNCTIONAL SIMULATION CLASS *iff there is a recursive function* sim *from ω to ω where for all e in ω*

- *$\phi_{\mathrm{sim}(e)}$ is a function in $\mathcal{F}$;*
- *if ϕ_e is total and $\phi_e(0)$ is equal to 0, then $\phi_{\mathrm{sim}(e)}$ is a delayed simulation of ϕ_e.*

(iv) *A subclass $\mathcal{S}$ of 2^ω is a* SIMULATION CLASS *iff there is a recursive function* sim *such that $\varphi_{\mathrm{sim}(e)}$ is a set in $\mathcal{S}$ for all e in ω and in addition $\varphi_{\mathrm{sim}(e)}$ is a delayed simulation of φ_e whenever φ_e is a set that does not contain* 0.

(That is, $\mathcal{S}$ is a simulation class if it satisfies the definition of functional simulation class with ω^ω and $\phi_0, \phi_1, \ldots$ replaced by 2^ω and $\varphi_0, \varphi_1, \ldots$.)

The concept delayed simulation was introduced by Mehlhorn [18, Axiom 6]. The relation between Mehlhorn's axiomatic approach and our own one in terms of standard reducibilities is discussed in Remark 20 below.

In the definition of the concept simulation class (and likewise for functional simulation classes) it is indeed reasonable to require that the function sim yields delayed simulations only in case $\varphi_e(0) = 0$. Remark 9 shows that there cannot be a recursive function sim where $\varphi_{\mathrm{sim}(e)}$ is total for all e in ω and is a delayed simulation of φ_e for *all* sets φ_e.

REMARK 9. *Recall from recursion theory that the sets A_0 and A_1 defined by*

$$A_i := \{e \in \omega : \varphi_e(e) = i\}, \qquad i = 0, 1,$$

are recursively inseparable; that is, there is no recursive set D such that A_0 is contained in D and A_1 is contained in $\overline{D}$. For a proof, it suffices to observe that any such set D must differ from every recursive set φ_e at place e. Now, by the smn-theorem, there is a recursive function g such that for all e, x, and y in ω we have

$$\varphi_{g(e,x)}(y) = \varphi_e(x).$$

However, if we assume that there is a recursive function sim *such that* $\varphi_{\mathrm{sim}(e)}$ *is a set for all* e *in* ω *and is a delayed simulation of* φ_e *for* all *sets* φ_e*, then we obtain a contradiction because the set*

$$D := \{e \in \omega : \varphi_{\mathrm{sim}(g(e,e))}(0) = 0\}$$

is recursive and separates A_0 *from* A_1 *by definition of* g*.*

Example 10 shows that the class of functions computable in polynomial time is a functional simulation class.

EXAMPLE 10. *For every* e *in* ω*, there is some Turing machine* $M_{\mathrm{sim}(e)}$ *that operates as follows.*

> *Given an input of length* n*, the Turing machine* $M_{\mathrm{sim}(e)}$ *tries to compute successively the values* $\phi_e(0), \phi_e(1), \ldots$ *by simulating the corresponding computations of* M_e*. The simulating machine* $M_{\mathrm{sim}(e)}$ *runs for a total of* n *steps and then outputs the last successfully computed value or, if even* $\phi_e(0)$ *could not be computed, the value* 0*.*

By definition of $\mathrm{sim}(e)$*, it should be clear that the function* sim *can be chosen to be recursive and that for every* e *in* ω*,*

> *the Turing machine* $M_{\mathrm{sim}(e)}$ *is total and runs in polynomial time and* $\phi_{\mathrm{sim}(e)}$ *is a delayed simulation of* ϕ_e *whenever* ϕ_e *is total and* $\phi_e(0)$ *is equal to* 0*.*

In connection with the latter property, we can safely assume that the simulation of M_e *for a single number argument requires at least one step and that consequently the simulating computation cannot skip function values of* φ_e *in the sense that, for example, the function* ϕ_e *has value* 1 *at some place* x *and has value* 0 *everywhere else, but* $\phi_{\mathrm{sim}(e)}$ *is the constant function with value* 0*.*

REMARK 11. *The concept of simulation class introduced in Definition* 8 *is robust under various changes of its definition. For example, we obtain the same concept if we do not require* $\varphi_{\mathrm{sim}(e)}$ *to be a delayed simulation of* φ_e *for all* e *where* φ_e *is a set that does not contain* 0*, but just for all such* e *where* φ_e *is finite or cofinite. Likewise, we can equivalently define the concept of simulation classes if we replace in its definition the concept of delayed simulations by a variant where the range of the witnessing reduction* l *as in Definition* 8 *need not be equal to* ω*, i.e., where the range of* l *is unbounded but might have gaps. For proofs and for further discussion we refer to Merkle* [20, section 3.6]*.*

Let $\sigma_0, \sigma_1, \ldots$ be an appropriate effective enumeration of all partial characteristic functions where by convention we let σ_0 be equal to the empty string. Recall from the introduction that $\langle A, \alpha\rangle$, the α-patch of A, is the set that agrees with α on $\mathrm{dom}(\alpha)$ and agrees with A otherwise.

DEFINITION 12.

(i) *Given a functional* Γ *and a function* $g : \omega \to \omega$*, the* $\boldsymbol{g}$-PATCH OF $\boldsymbol{\Gamma}$ *is the functional* $\Gamma \otimes g$ *defined by*

$$(\Gamma \otimes g)(A, x) := \Gamma(\langle A, \sigma_{g(x)}\rangle, x).$$

(ii) *Let* $\mathcal{R}$ *be a class of functionals and let* $\mathcal{F}$ *be a subclass of* ω^ω*. Then the* CLASS OF $\boldsymbol{\mathcal{F}}$-PATCHES OF $\boldsymbol{\mathcal{R}}$ *is*

$$\mathcal{R} \otimes \mathcal{F} := \{\Gamma \otimes g : \Gamma \in \mathcal{R} \text{ and } g \in \mathcal{F}\}.$$

(iii) *A class* $\mathcal{R}$ *of functionals is closed under* DELAYED PATCHING *iff there is a functional simulation class* $\mathcal{F}$ *where* $\mathcal{R} \otimes \mathcal{F}$ *is contained in* $\mathcal{R}$*.*

Proposition 13, which has some interest in its own, is used in Example 14 in order to show that the standard enumeration of polynomially time-bounded oracle Turing machines yields a reduction cover for the reducibility $\leq_{\mathrm{T}}^{\mathcal{P}}$ that is closed under delayed patching.

PROPOSITION 13. *Let the recursive function b from ω to ω be nondecreasing and unbounded and let $\mathcal{F}$ be a functional simulation class. Then the class*

$$\{f \in \mathcal{F} : f(x) \leq b(x) \text{ for all } x \in \omega\} \tag{11}$$

is again a functional simulation class.

Proof. The idea of the proof is quite simple. Given some recursive function h in class $\mathcal{F}$, we construct a recursive delayed simulation g of h where $g(x) \leq b(x)$ holds for all x in ω. Then for any delayed simulation f of g, where this fact is witnessed by some nondecreasing function l with range ω, we have for all x in ω

$$f(x) \;=\; g(l(x)) \;\leq\; b(l(x)) \;\leq\; b(x).$$

The relations hold, from left to right, by choice of l, because g is bounded by b, and finally because b is nondecreasing and by $l(x) \leq x$. Thus in order to show that the class defined in (11) is a functional simulation class, we map a recursive function not just to a delayed simulation h in $\mathcal{F}$, but in addition we construct a recursive delayed simulation g of h that is bounded by b, and then we pick a delayed simulation f of g in $\mathcal{F}$. An index for such a function g can be obtained effectively from an index for h; i.e., there is a recursive function r that maps any index for a function h in $\mathcal{F}$ to an index of a delayed simulation g of h as above. Then, given a function sim that witnesses that $\mathcal{F}$ is a functional simulation class, by the preceding discussion the function

$$\mathrm{sim}' := \mathrm{sim} \circ r \circ \mathrm{sim}$$

witnesses that the class defined in (11) is a functional simulation class.

Formally, given some function h in $\mathcal{F}$, we define a corresponding function g as above by $g(x) := h(l(x))$ where $l(0)$ is equal to 0 and for all x we let

$$l(x+1) := \begin{cases} l(x)+1 & \text{in case } h(l(x)+1) \leq b(x+1), \\ l(x) & \text{otherwise.} \end{cases}$$

□

EXAMPLE 14. *We fix an effective enumeration of polynomially time-bounded oracle Turing machines where machine $\langle e, j\rangle$ in the enumeration corresponds to Turing machine M_e in the standard enumeration of Turing machines with an additional restriction on the running time that is given by polynomial j in some appropriate enumeration of all polynomials. Then the class $\mathcal{R}_{\mathrm{T}}^{\mathcal{P}}$ of all functionals computed by some oracle Turing machine in this enumeration is an effective reduction cover for $\leq_{\mathrm{T}}^{\mathcal{P}}$. Now, first, adding polynomially time-bounded subcomputations to a polynomially time-bounded oracle Turing machine will not put us outside of the class $\mathcal{R}_{\mathrm{T}}^{\mathcal{P}}$, and, second, the class $\mathcal{FP}$ of functions computable in polynomial is a functional simulation class according to Example* 10. *However, this does not imply directly that the standard reduction cover $\mathcal{R}_{\mathrm{T}}^{\mathcal{P}}$ is closed under delayed patching because given some function g that is computable in polynomial time, the corresponding function values might be so large that for more reasonable effective enumerations of the finite partial characteristic functions it will be impossible to access the values of the partial characteristic*

function $\sigma_{g(x)}$ in time polynomially bounded in the length of x. Thus if we patch a functional Γ in $\mathcal{R}^{\mathcal{P}}_{\mathrm{T}}$ by some function g computable in polynomial time, then in general the new functional $\Gamma \otimes g$ will not be computable in polynomial time unless we choose an effective enumeration $\sigma_0, \sigma_1, \ldots$ of the finite partial characteristic functions where the coding is rather inefficient.

In order to handle this problem, let b be a nondecreasing and unbounded recursive function that grows so slowly that for all x and for all $j \leq b(x)$, we can in time $|x|$ answer all relevant questions about the domain and the values of σ_j. Then by Example 10 *and Proposition* 13, *the class*

$$\mathcal{FP}_{\leq b} := \{g \in \mathcal{FP} : g(x) \leq b(x) \text{ for all } x \in \omega\}$$

is a functional simulation class. Furthermore, by our choice of b, the reduction cover $\mathcal{R}^{\mathcal{P}}_{\mathrm{T}}$ is closed under patching with functions in $\mathcal{FP}_{\leq b}$. More precisely, for any functional Γ computed by a polynomially time-bounded oracle Turing machine M and for any function g in $\mathcal{FP}_{\leq b}$, the following oracle Turing machine M' computes $\Gamma \otimes g$.

> *M' works essentially as M but every query state of M is replaced by a sequence of new states corresponding to a subcomputation that takes care of the g-patching. In the subcomputation, first, the value of $g(x)$ for the current number input x is computed and, second, it is checked whether the value y written currently on the oracle tape is in the domain of $\sigma_{g(x)}$. This check can be done in polynomial time by definition of b and by choice of g in $\mathcal{FP}_{\leq b}$. After this subcomputation, the execution of M' resumes as follows. In case y is not in the domain of $\sigma_{g(x)}$, then the next state is determined in exactly the same way as for M by the value of the oracle at place y while, otherwise, the value $\sigma_{g(x)}(y)$ is used in place of the actual value of the oracle in order to determine the next state.*

In connection with the description of the oracle Turing machine M' recall that once an oracle Turing machine enters a query state, the next state is determined by the value of the oracle for the number currently written on the oracle tape. Observe further that for Turing machine models with write-only access to the oracle tape, we have to keep a copy of the current oracle question on some work tape in order to be able to check whether the current query is in the domain of the patch we want to apply.

2.3. Definition by Oracle-dependent cases and effective reduction covers.

DEFINITION 15.

(i) *A* TT-CONDITION *is a subclass $\mathcal{T}$ of 2^ω such that for some finite set I and for all sets A, membership of A in $\mathcal{T}$ depends only on the restriction of A to I.*

(ii) *Let $\mathcal{T}$ be a tt-condition and let Γ_0 and Γ_1 be functionals. The $\boldsymbol{\mathcal{T}}$-*MIX *of Γ_0 and Γ_1 is the functional $\langle \Gamma_0, \Gamma_1 \rangle^{\mathcal{T}}$ defined by*

$$\langle \Gamma_0, \Gamma_1 \rangle^{\mathcal{T}}(A) := \begin{cases} \Gamma_0(A) & \text{if } A \in \mathcal{T}, \\ \Gamma_1(A) & \text{if } A \notin \mathcal{T}. \end{cases} \tag{12}$$

(iii) *For a tt-condition $\mathcal{T}$, we denote the transition from two functionals to their $\mathcal{T}$-mix as* DEFINITION BY ORACLE-DEPENDENT CASES *with respect to $\mathcal{T}$.*

(iv) *A class $\mathcal{R}$ of functionals is* CLOSED UNDER DEFINITION BY ORACLE-DEPENDENT CASES *iff for every tt-condition $\mathcal{T}$ the $\mathcal{T}$-mix of two functionals in $\mathcal{R}$ is again in $\mathcal{R}$.*

DEFINITION 16. *A list $\Delta_0, \Delta_1, \dots$ of functionals is an* EFFECTIVE ENUMERATION *of a set $\mathcal{R}$ of functionals iff there is a recursive function e such that $\Delta_i = \Phi_{e(i)}$ for all i and the set $\mathcal{R}$ coincides with $\{\Delta_0, \Delta_1, \dots\}$.*

A set $\mathcal{R}$ of functionals is an EFFECTIVE REDUCTION COVER *for a binary relation $\leq_r$ on 2^ω iff $\mathcal{R}$ has some effective enumeration and for all sets A and B, $A \leq_r B$ iff $A = \Gamma(B)$ for some functional Γ in $\mathcal{R}$.*

For any class $\mathcal{R}$ that has an effective enumeration, by the padding lemma this fact is always witnessed by a function e that is strictly monotonous and hence has recursive range. Using the latter observation, it is immediate from Definitions 1 and 16 that a binary relation on 2^ω is a bounded reducibility iff it has some effective reduction cover. For standard reducibilities as introduced in Definition 17, we require in addition that there is an effective reduction cover that satisfies certain closure properties.

DEFINITION 17. *A binary relation $\leq_r$ on 2^ω is a* STANDARD REDUCIBILITY *iff*

(i) *the relation $\leq_r$ is faithful and c.f.v.;*

(ii) *there is some effective reduction cover for $\leq_r$ that is closed under delayed patching and under definition by oracle-dependent cases.*

In the definition of standard reducibilities we can drop the condition that the relation is c.f.v. because this follows already from the remaining requirements. We omit the simple but lengthy proof and refer to Merkle [20, Proposition 35].

EXAMPLE 18. *Standard reducibilities comprise not only the usual polynomially time- or logarithmically space-bounded reducibilities of many-one and Turing type but also less common reducibilities such as the honest variant $\leq^{\mathcal{P}}_{\text{h-T}}$ of $\leq^{\mathcal{P}}_{\text{T}}$, the reducibility $\leq_{\text{NC}_1}$ defined in terms of circuits, as well as the nontransitive relations $\leq^{\mathcal{P}}_{k-\text{tt}}$ for any fixed $k \geq 2$* [20].

Examples of bounded reducibilities that are not standard reducibilities are given by Turing reducibility confined to constant space or by the reducibility $\leq^{\mathcal{P}}_1$, i.e., the variant of polynomially time-bounded many-one reducibility where the reduction functions are required to be one-to-one. For a proof, recall that the latter reducibility is not c.f.v. and observe that for the former reducibility the class of admissible cases is not a simulation class.

REMARK 19. *Given an effective reduction cover $\mathcal{R}$ for a bounded reducibility $\leq_r$, the closure of $\mathcal{R}$ under definition by oracle-dependent cases is again an effective reduction cover for $\leq_r$. This follows from the simple observation that for each tt-condition $\mathcal{T}$, the $\mathcal{T}$-mix $\langle \Gamma_0, \Gamma_1 \rangle^{\mathcal{T}}$ of two functionals in $\mathcal{R}$ is again a recursive functional and maps every set B to $\Gamma_0(B)$ or to $\Gamma_1(B)$, i.e., to a set that by assumption is reducible to B.*

As a consequence, a bounded reducibility always has an effective reduction cover that is closed under definition by oracle-dependent cases. The point in requiring the existence of such a reduction cover in the definition of standard reducibility is that this reduction cover at the same time is required to be closed under delayed patching. When working with such reduction covers, we are able to mimic switching between functionals according to oracle-independent subcomputations, as is done with the gap language technique. *If we first put two functionals together via a definition by cases with respect to some tt-condition, then afterwards, by finitely patching the set argument of the new compound functional at all the places the selecting tt-condition depends on, we can alternate between the two initial functionals; for details of this construction see the proof of Proposition* 39.

In Remark 20, we briefly discuss the relation of Mehlhorn's axiomatic approach to our own one in terms of the concept of standard reducibility. For a more detailed

account we refer to Merkle [20, section 3.7].

REMARK 20. *Mehlhorn's axiom system* [18] *apparently is meant to be applied to bounded reducibilities of Turing type. In accordance with this intended range of application, his axioms do not just involve delayed simulations of recursive sets but of all sets that can be computed relative to the given oracle. Furthermore, every lower cone is required to be closed under definition by number-dependent cases with respect to all sets in this lower cone. As a consequence, Mehlhorn's axiom system is satisfied for the usual reducibilities of Turing type but not for standard reducibilities such as many-one or bounded truth-table reducibility restricted to polynomial time* [20].

On the other hand, the definition of standard reducibility involves a condition on delayed patching, which has no counterpart in Mehlhorn's axiom system. The rationale for adding such a condition is that we aim at embedding distributive lattices, not just partial orderings. In connection with embeddings of lattices, the condition on delayed patching is then used for constructing pairs of sets with a specific g.l.b. Embeddings of partial orderings can indeed be obtained from assumptions that are more general than the concept of standard reducibility; see Corollary 53 *below.*

3. Three lemmas on standard reducibilities.

3.1. Gap languages and generalized use. In this section we will show three lemmas that will then be used in section 4 in the proof of our main result about lattice embeddings for standard reducibilities. Embeddings of partial orderings and of lattices have been constructed before for several specific bounded reducibilities and in corresponding proofs the use of gap languages has become a standard technique [1, 4, 30, 31].

DEFINITION 21. *A set is a* GAP LANGUAGE *iff the set and its complement are both infinite.*

A gap language A can be conceived as a partition of ω into infinitely many finite *blocks*, where each block corresponds to a maximal set of consecutive natural numbers that either all are in A or all are in the complement of A. Indeed we will use gap languages almost exclusively as a convenient tool to specify such partitions. We will number the blocks of a gap language G in the natural way, and we will assign to each x in ω the number of its block with respect to G. These concepts are formally introduced in Definition 22. For further use, however, they are defined for arbitrary partial characteristic functions and not just for gap languages. Intuitively speaking, given a partial characteristic function α where x is the least place such that $\alpha(x)$ is undefined, then the blocks of α are defined in the natural way up to but not including x, and these are all blocks of α.

DEFINITION 22. *Let $\alpha : \omega \to \{0,1\}$ be a partial characteristic function. For all x in ω such that $\alpha(y)$ is defined for all $y \leq x$, we let*

$$\mathrm{bn}(\alpha, x) \;:=\; |\{y \in \omega : y < x \;\&\; \alpha(y) \neq \alpha(y+1)\}|\,,$$

and we let $\mathrm{bn}(\alpha, x)$ be undefined for all other x. We call $\mathrm{bn}(\alpha, x)$ the BLOCK NUMBER *of x with respect to α.*

For all j in ω, we let BLOCK $\boldsymbol{j}$ OF $\boldsymbol{\alpha}$ *be equal to the set $\{x \in \omega : \mathrm{bn}(\alpha, x) = j\}$, and we say block j of α* EXISTS *iff block j of α is nonempty.*

The blocks of a partial characteristic function α are "numbered" by the function $\mathrm{bn}(\alpha, .)$, starting with block 0. Obviously, the function $\mathrm{bn}(\alpha, .)$ is total iff α is total, and α has infinitely many blocks iff α is a gap language.

DEFINITION 23. *Let A and B be gap languages. Then B is a* GAP COVER *for A iff every block of B contains some block of A.*

It is immediate from Definition 23 that the gap cover relation is transitive; that is, if B is a gap cover for A and so is C for B, then C is also a gap cover for A.

The values of a recursive functional Γ can be computed by an oracle Turing machine that on input x and oracle A can read only a finite part of A before it outputs $\Gamma(A,x)$. Consequently, there is a finite set I such that $\Gamma(A,x)$ is equal to $\Gamma(B,x)$ for all B that agree with A on I. By the theorem of Trakhtenbrot and Nerode [24, Proposition III.3.2], the set I can in fact be chosen independently of A; i.e., for all x there is a finite set I such that

$$X|I = Y|I \quad \text{implies} \quad \Gamma(X,x) = \Gamma(Y,x) \qquad \text{for all sets } X,Y. \tag{13}$$

The class of sets I that satisfy (13) are closed under intersection and thus in particular there is a least such set, which obviously is uniquely determined and finite [20, 22].

DEFINITION 24. *Let Γ be a recursive functional. For every natural number x, we call the least set I that satisfies* (13) *the* GENERALIZED USE *of Γ at x, and we denote this set by* $\mathrm{u}(\Gamma,x)$.

REMARK 25. *There is an effective procedure that on inputs x and e eventually outputs* $\mathrm{u}(\Phi_e,x)$ *whenever $\Phi_e(X,x)$ is defined for all oracles X (and that otherwise might fail to terminate). The proof of the theorem of Trakhtenbrot and Nerode shows that in the given situation a finite set I as in* (13) *can be obtained effectively. In order to determine* $\mathrm{u}(\Phi_e,x)$ *it then suffices to search through all subsets of I.*

In principle, all the material presented in what follows could be formulated in terms of the standard concept of use for oracle Turing machines, where for given oracle Turing machine, input, and oracle the use contains exactly the numbers at which the oracle Turing machine queries its oracle. However, we consider it to be convenient to work with the generalized use because the latter is determined just by the functional and does not require to specify a specific oracle Turing machine computing the functional. In fact, the concept of generalized use extends canonically to arbitrary continuous, not necessarily recursive functionals [20, 22].

3.2. The diagonalization lemma. Suppose that we are given an effective reduction cover $\{\Delta_0,\Delta_1,\dots\}$ for a bounded reducibility $\leq_r$. Then a set E is r-reducible to a set F iff there is *some* functional Δ_j where E is equal to $\Delta_j(F)$. As a consequence, we can construct sets E and F where E is not reducible to F by *diagonalizing* against all functionals Δ_j; that is, it is sufficient to ensure that for all j in ω there is some x_j in ω such that we have

$$E(x_j) \neq \Delta_j(F,x_j). \tag{14}$$

Now the generalized use of the functionals Δ_j is always finite and thus for any given j we can enforce (14) by specifying $E(x_j)$ and a finite part of F; that is, we are led to a finite extension construction. Before we employ such a construction in the proof of Lemma 28, we introduce some notation.

DEFINITION 26. *Let G be a gap language and let $k \geq 1$ be a natural number. Then two n-tuples $(X_1,\dots,X_k)$ and $(Y_1\dots,Y_k)$ of sets are* $\boldsymbol{G}$-SIMILAR, *written*

$$(X_1,\dots,X_k) \simeq^G (Y_1,\dots,Y_k)$$

iff there are infinitely many blocks I of G such that for all $i=1,\dots,k$, the sets X_i and Y_i agree on I. Two sets are G-similar iff the corresponding 1-tuples are G-similar.

For further use, we state the following easy fact, which is immediate from the definitions of the concepts involved.

REMARK 27. *If two tuples of sets are G-similar for some gap language G, then they are also H-similar for every gap cover H of G.*

LEMMA 28 (diagonalization lemma). *Let $\leq_r$ be a bounded reducibility such that $\leq_r$ is c.f.v. and let E and F be recursive sets where $E \not\leq_r F$. Then there is a recursive gap language G such that we have for all sets E' and F'*

$$(E, F) \simeq^G (E', F') \quad \textit{implies} \quad E' \not\leq_r F'.$$

The diagonalization lemma and its proof are similar to Schöning's uniform diagonalization theorem [4, 30]. We will use the diagonalization lemma in connection with results about partial order embeddings in order to ensure that the constructed embeddings preserve nonorder. More precisely, given sets E and F where E is not reducible to F, then by the diagonalization lemma in order to construct a set E' that is not reducible to F' it is sufficient to ensure that E' and F' agree with E and F, respectively, on some set that contains infinitely many blocks of the gap language G obtained from the diagonalization lemma.

Proof of Lemma 28. We construct in stages a gap language G as required in the lemma. We write I_s for block s of G and during stage s we specify which numbers are in I_s. This then determines G by letting $G(0)$ be equal to 0.

Let $\Delta_0, \Delta_1, \ldots$ be an effective enumeration of an effective reduction cover for $\leq_r$. At stage 0, we let $I_0 = \{0\}$. At stage $s > 0$, for all $j \leq s$ and for all pairs α and β of finite partial characteristic functions with domain equal to the union of the blocks $I_0, \ldots, I_{s-1}$, we let $z_{\alpha,\beta,j}$ be the least number that satisfies

$$\langle E, \alpha\rangle(z_{\alpha,\beta,j}) \neq \Delta_j(\langle F, \beta\rangle, z_{\alpha,\beta,j}). \tag{15}$$

There is indeed such a number as $\langle E, \alpha\rangle = \Delta_j(\langle F, \beta\rangle)$ would imply $E \leq_r F$ by $\leq_r$ being c.f.v. Furthermore, the least such number can be found effectively in α, β, and j because the Δ_j are uniformly recursive. Next we choose I_s so large that for all such α, β, and j we have

$$\{z_{\alpha,\beta,j}\} \cup \mathrm{u}(\Delta_j, z_{\alpha,\beta,j}) \subseteq I_0 \cup \cdots \cup I_s. \tag{16}$$

In order to show that the gap language G has the required properties, assume for a proof by contradiction that there are sets E' and F' where, first, (E', F') and (E, F) are G-similar, and, second, the set E' is r-reducible to F', say via functional Δ_k. Choose some $s \geq k$ where E' and F' agree with E and F, respectively, on block s of G. Let α and β be the restrictions of E' and F', respectively, to the union of the blocks $I_0, \ldots, I_{s-1}$. Now the witness $z_{\alpha,\beta,k}$ for $\langle E, \alpha\rangle \neq \Delta_k(\langle F, \beta\rangle)$ we found during stage s of the construction of G witnesses $E' \neq \Delta_k(F')$ due to (16) and because by assumption $\langle E, \alpha\rangle$ and $\langle F, \beta\rangle$ agree with E' and F', respectively, on the union of $I_0, \ldots, I_s$. □

3.3. The window lemma. We show now that for standard reducibilities every reduction to a recursive set is witnessed by a functional that on increasing number inputs ignores larger and larger initial parts of its set argument. In the corresponding proof we exploit that standard reducibilities possess reduction covers that are closed under delayed patching, and we apply a special form of delayed patching where we do not patch according to an arbitrary effective sequence of finite partial characteristic

functions but with increasingly long initial segments of the characteristic function of some *fixed* recursive set.

DEFINITION 29. *A function $m : \omega^2 \to \omega$ is a* MODULUS OF ORACLE SIMULATION *iff m is recursive and for all e in ω the sequence $\sigma_{m(e,0)}, \sigma_{m(e,1)}, \dots$ converges monotonously to φ_e (i.e., $\sigma_{m(e,i)} \sqsubseteq \sigma_{m(e,i+1)}$ for all i and the domain of φ_e is equal to the union of the domains of $\sigma_{m(e,0)}, \sigma_{m(e,1)}, \dots$).*

A reduction cover is CLOSED UNDER ORACLE SIMULATION *iff there is a modulus of oracle simulation m such that $\mathcal{R}$ is closed under patching with the functions $m(e,.)$ (i.e., for all Γ in $\mathcal{R}$ and all e in ω, the functional $\Gamma \otimes m(e,.)$ is in $\mathcal{R}$).*

EXAMPLE 30. *For all e and i in ω, let $m(e,i)$ be the least index such that $\sigma_{m(e,i)}$ is equal to the restriction of φ_e to the set $\{j < i : \varphi_{e,|i|}(j)\downarrow\}$, where $\varphi_{e,s}(i)$ denotes the s-step approximation to $\varphi_e(i)$. We leave it to the reader to show that the function m is recursive and is indeed a modulus of oracle simulation.*

LEMMA 31. *Let $\mathcal{F}$ be a functional simulation class. Then there is a modulus of oracle simulation m such that for all e in ω the function $m(e,.)$ is in $\mathcal{F}$.*

Proof. Let the function sim witness that $\mathcal{F}$ is a functional simulation class and fix an arbitrary modulus of oracle simulation m (for example, the one defined in Example 30). We can assume $m(e,0) = 0$ because by our convention σ_0 is the empty partial characteristic function λ; hence, for all e, the condition on the convergence of the sequence $\sigma_{m(e,0)}, \sigma_{m(e,1)}, \dots$ in Definition 29 remains valid if we change $m(e,0)$ into 0.

By the *smn*-theorem, fix a recursive function g such that for all e and x, $\phi_{g(e)}(x)$ is equal to $m(e,x)$. Then the function m' defined by

$$m'(e,x) = \phi_{\mathrm{sim}(g(e))}(x)$$

is recursive, and m' is in fact a modulus of oracle simulation because for each e the function $m'(e,.)$ is a delayed simulation of $m(e,.)$. Intuitively speaking, m' yields a simulation of φ_e that is basically the same but might be "slower" than the one provided by m. □

PROPOSITION 32. *If a reduction cover is closed under delayed patching, then it is also closed under oracle simulation. In particular, every standard reducibility has an effective reduction cover that is closed under oracle simulation.*

Proof. The first assertion follows easily from Lemma 31 and the definition of the two closure conditions involved. The second assertion is then immediate because by definition every standard reducibility has an effective reduction cover that is closed under delayed patching. □

Given a class $\mathcal{C}$ and an effective reduction cover $\mathcal{R}$ for a bounded reducibility, we will use the expression "every reduction to a set in $\mathcal{C}$ is witnessed by some functional in $\mathcal{R}$ such that ... " to express that for every C in $\mathcal{C}$ and every set X that is reducible to C, there is a functional Δ in $\mathcal{R}$ that has the properties under consideration such that X is equal to $\Delta(C)$. Furthermore, given a gap language G and some x in ω we let

$$\mathrm{Nb}(x,G) := \{y \in \omega : \mathrm{bn}(G,x) - 1 \leq \mathrm{bn}(G,y) \leq \mathrm{bn}(G,x) + 1\};$$

that is, $\mathrm{Nb}(x,G)$ consists of the block of x with respect to G and of the adjacent block to the left and to the right, respectively.

LEMMA 33 (window lemma). *Let $\mathcal{R}$ be an effective reduction cover for some bounded reducibility such that $\mathcal{R}$ is closed under delayed patching. Let the class $\mathcal{C}$ be recursively presentable.*

Then there is a recursive gap language G such that every reduction to a set in $\mathcal{C}$ is witnessed by a functional Δ in $\mathcal{R}$ such that for all but finitely many x in ω,

$$\mathrm{u}(\Delta, x) \subseteq \mathrm{Nb}(x, G).$$

Proof. We fix some effective enumeration $\Delta_0, \Delta_1, \ldots$ of $\mathcal{R}$.

Claim 1. *There is a recursive function h such that for all i and for almost all places x we have*

$$\max \mathrm{u}(\Delta_i, x) \leq h(x).$$

Proof. The set $\mathrm{u}(\Delta_i, x)$ is always finite and can be computed from i and x; hence it is sufficient to let $h(x)$ be equal to the maximal number in the union of the sets $\mathrm{u}(\Delta_0, x), \mathrm{u}(\Delta_1, x), \ldots, \mathrm{u}(\Delta_x, x)$. □

Claim 2. *There is an recursive function b that is nondecreasing and unbounded such that every reduction to a set C in $\mathcal{C}$ is witnessed by some Δ in $\mathcal{R}$ where the generalized use of Δ is almost everywhere bounded from below by b; i.e., we have for almost all x,*

$$b(x) \leq \min \mathrm{u}(\Delta, x).$$

Proof. Let r be a recursive function such that $\mathcal{C}$ is equal to $\{\varphi_{r(i)} : i \in \omega\}$. By Proposition 32, we can fix a modulus of oracle simulation m that witnesses that $\mathcal{R}$ is closed under oracle simulation. Then we let for all i and x in ω,

$$v(i, x) := \max_{y \leq x} \left[\{0, \ldots, y\} \subseteq \mathrm{dom}(\sigma_{m(r(i),x)})\right].$$

The function v is recursive because m and r are recursive. Furthermore, for every fixed i, the function $v(i, .)$ is nondecreasing and unbounded because $\varphi_{r(i)}$ is total; hence the domains of the partial characteristic functions $\sigma_{m(r(i),x)}$ converge nondecreasingly to ω as x goes to infinity. Next, we let

$$b(x+1) := \begin{cases} b(x)+1 & \text{in case } b(x)+1 \leq v(i,x) \text{ for all } i \leq b(x), \\ b(x) & \text{otherwise.} \end{cases}$$

The function b is by definition nondecreasing, and it is unbounded because the functions $v(i, .)$ are unbounded. Then given a set $C = \varphi_{r(i)}$ in $\mathcal{C}$ and a set that is reducible to C, we fix a witnessing functional Δ in $\mathcal{R}$. The functional

$$\Delta' := \Delta \otimes m(r(i), .),$$

i.e., the $m(r(i), .)$-patch of Δ, is in $\mathcal{R}$ by choice of the modulus m. The functionals Δ and Δ' agree on the set C because the patches $\sigma_{m(r(i),x)}$ agree with C on their entire domain; i.e., we have for all x

$$\Delta'(C, x) = [\Delta \otimes m(r(i), .)](C, x) = \Delta(\langle C, \sigma_{m(r(i),x)}\rangle, x) = \Delta(C, x).$$

Furthermore, by definition of b, for almost all x the numbers less than or equal to $b(x)$ are all contained in the domain of $\sigma_{m(r(i),x)}$ and hence are not in the generalized use $\mathrm{u}(\Delta', x)$. □

By Claims 1 and 2, we obtain recursive functions h and b, where b is nondecreasing and unbounded and such that every reduction to some set in $\mathcal{C}$ is witnessed by some

functional Δ in $\mathcal{R}$ where for almost all places x the generalized use $\mathrm{u}(\Delta, x)$ is a subset of the possibly empty interval between $b(x)$ and $h(x)$. Thus it is sufficient to construct a gap language G such that for all x the interval between $b(x)$ and $h(x)$ is contained in $\mathrm{Nb}(x, G)$. The blocks of G are defined inductively, where block 0 of G is equal to $\{0\}$. In the induction step, let U_s be the union of the previously defined blocks 0 through s of G and let m_s be the least number not contained in U_s such that

$$\max U_s < b(m_s) \quad \text{and} \quad \max\{h(y) : y \text{ in } U_s\} \leq m_s.$$

Then let block $s+1$ of G be equal to $\{0, \ldots, m_s\} - U_s$. □

REMARK 34. *In the conclusion of Lemma* 33 *we can replace G by any gap cover of G. This is true simply because for any gap cover M of G and for any x the set* $\mathrm{Nb}(x, G)$ *is contained in* $\mathrm{Nb}(x, M)$.

For a proof, fix x and assume that x is in block i of G and in block j of M. Then block $j-1$ of M cannot contain block i of G because the latter contains x, which is in block j of M. Thus, by M being a gap cover for G, block $j-1$ of M must contain some block k of G where $k \leq i-1$. But then, obviously, the union of the consecutive blocks $j-1$ and j of M contains block $i-1$ of G. The assertion now follows by a symmetric argument that shows that blocks j and $j+1$ of M contain block $i+1$ of G.

3.4. Admissible cases and the coding lemma. Recall from the introduction that a functional can alternatively be described as a unary function from 2^ω to 2^ω or as a binary function from $2^\omega \otimes \omega$ to $\{0,1\}$. The latter characterization suggests two ways of how two functionals might be combined into a new functional via a definition by cases. The case condition can depend on the set argument or on the number argument. In Definition 15, we have introduced a notion of DEFINITION BY ORACLE-DEPENDENT CASES, where two functionals are mixed according to a tt-condition on their set argument. We introduce now a notion of DEFINITION BY NUMBER-DEPENDENT CASES where the case condition depends on the number argument. Given two functionals Δ_0 and Δ_1 and a set M, the functional Δ obtained via definition by number-dependent cases according to M is defined by

$$\Delta(X) := \langle \Delta_0(X), \Delta_1(X) \rangle^M; \tag{17}$$

i.e., $\Delta(X, x)$ is equal to the set $\Delta_0(X, x)$ if x is in M and is equal to the set $\Delta_1(X, x)$ otherwise. We will be interested in the class of all sets M such that when applying definition by number-dependent cases according to M to a given effective reduction cover, then using the functionals obtained this way we can just reduce the same pairs of sets as before. Remark 35 gives equivalent characterizations of this class.

REMARK 35. *Let $\leq_r$ be a binary relation on 2^ω. For the moment, call a functional Γ an r-reduction iff for all sets X the set $\Gamma(X)$ is r-reducible to X. Then for any set M, the following conditions are equivalent.*

(i) *All lower r-cones are closed under definition by number-dependent cases with respect to M (i.e., if the sets A and B are both reducible to X, then so is the set $\langle A, B \rangle^M$).*

(ii) *For any r-reductions Δ_0 and Δ_1, the functional Δ defined in* (17) *is again an r-reduction.*

(iii) *For any functionals Δ_0 and Δ_1 in some fixed effective reduction cover for $\leq_r$, the functional Δ defined in* (17) *is again an r-reduction.*

First, we show that (i) *implies* (ii). *Fix any set M that satisfies* (i), *let Δ_0 and Δ_1 be arbitrary r-reductions and let Δ be defined as in* (17). *By assumption, for any set X*

the sets $\Delta_0(X)$ and $\Delta_1(X)$ are r-reducible to X; hence so is $\Delta(X)$. Now the set X has been chosen arbitrarily; thus Δ is an r-reduction. The implication from (ii) *to* (iii) *is immediate because all functionals in an effective reduction cover for $\leq_r$ must be r-reductions. It remains to show that* (iii) *implies* (i). *Fix any set M such that* (iii) *is satisfied for some reduction cover for $\leq_r$. Given sets A and B that are both reducible to some set X, choose witnessing functionals Δ_0 and Δ_1 from this reduction cover. Then the functional Δ defined in* (17) *is an r-reduction. By construction, $\Delta(X)$ is equal to $\langle A, B\rangle^M$; hence the latter set is reducible to X. As a consequence, M satisfies* (i).

DEFINITION 36. *For a binary relation on 2^ω, $\mathcal{L}_r$ denotes the* CLASS OF LEAST SETS, *i.e, $\mathcal{L}_r = \{A : A \leq_r B$ for all sets $B\}$, and $\mathcal{M}_r$ denotes the* CLASS OF ADMISSIBLE CASES, *which is defined by*

$$\mathcal{M}_r \quad := \quad \{M : \text{ for all sets } A, B, X, \\ [A \leq_r X \text{ and } B \leq_r X \text{ implies } \langle A, B\rangle^M \leq_r X]\}.$$

For the bounded reducibility $\leq_{\mathrm{T}}^{\mathcal{P}}$ it is easy to see that the class of least sets $\mathcal{L}_{\mathrm{T}}^{\mathcal{P}}$ and the class of admissible cases $\mathcal{M}_{\mathrm{T}}^{\mathcal{P}}$ are both equal to $\mathcal{P}$; hence, in particular, for the relation $\leq_{\mathrm{T}}^{\mathcal{P}}$ the class of admissible cases is closed under the set theoretical operations union, intersection, and complement. Proposition 37 shows that the latter assertion holds for all standard reducibilities and that $\mathcal{M}_r$ is always included in $\mathcal{L}_r$. Example 38 shows that this inclusion is strict for faithful relations in general.

PROPOSITION 37. *Let $\leq_r$ be a faithful relation on 2^ω.*

(i) *$\mathcal{M}_r$ is a subset of $\mathcal{L}_r$.*

(ii) *$\mathcal{M}_r$ contains $\emptyset$ and is closed under the set theoretical operations union, intersection, and complement. Equivalently, $(\mathcal{M}_r, \subseteq)$ is a subalgebra of $(2^\omega, \subseteq)$.*

Proof. For a proof of the first assertion, recall that for a faithful relation $\leq_r$ the sets $\emptyset$ and ω are reducible to all other sets; hence by definition of $\mathcal{M}_r$ we have for all sets A and for all sets M in $\mathcal{M}_r$

$$M = \langle \omega, \emptyset\rangle^M \leq_r A.$$

The second assertion follows from the definition of $\mathcal{M}_r$ and because we have for all sets A, B, M, M_0, and M_1

$$\langle A, B\rangle^{\emptyset} = B, \quad \langle A, B\rangle^{\overline{M}} = \langle B, A\rangle^M, \quad \langle A, B\rangle^{M_0 \cap M_1} = \langle \langle A, B\rangle^{M_0}, B\rangle^{M_1},$$

where the equations, from left to right, show that $\mathcal{M}_r$ contains $\emptyset$ and is closed under complement and intersection. Closure under union then follows by De Morgan's laws. □

EXAMPLE 38. *A set is called* NON-SELF-DUAL *iff it is not $\leq_{\mathrm{m}}^{\mathcal{P}}$-reducible to its complement. Ladner, Lynch, and Selman* [14] *have shown that there are recursive non-self-dual sets. Choose such a set N and let*

$$A \leq_\nu B \;:\Leftrightarrow\; A \leq_{\mathrm{m}}^{\mathcal{P}} (B \oplus \overline{N}), \tag{18}$$

where $\overline{N}$ denotes the complement of N. Then the relation $\leq_\nu$ is faithful and the class of least sets $\mathcal{L}_\nu$ is exactly the lower $\leq_{\mathrm{m}}^{\mathcal{P}}$-cone of $\overline{N}$. Hence, in particular, $\mathcal{L}_\nu$ is not closed under complement, because it contains $\overline{N}$, but cannot contain the set N due to N being non-self-dual. Consequently, by Proposition 37, *$\mathcal{M}_\nu$ is strictly contained in $\mathcal{L}_\nu$.*

PROPOSITION 39. *For a standard reducibility $\leq_r$, the class $\mathcal{M}_r$ is a simulation class.*

Proof. We choose an effective reduction cover $\mathcal{R}$ for the standard reducibility $\leq_r$ that is closed under definition by oracle-dependent cases and under delayed patching. Let the latter closure property be witnessed by some functional simulation class $\mathcal{F}$; that is, $\mathcal{R} \otimes \mathcal{F}$ is contained in $\mathcal{R}$. Furthermore, we pick i and j such that the finite partial characteristic functions σ_i and σ_j are incompatible; that is, there is some y in ω where σ_i and σ_j are both defined and disagree. In particular, by our convention that σ_0 is the empty string, i and j differ from 0. For every set L, we let

$$f_L(x) := \begin{cases} 0 & \text{if } x = 0, \\ i & \text{if } x \neq 0 \text{ and } L(x) = 0, \\ j & \text{otherwise;} \end{cases}$$

that is, basically we replace in the characteristic function of L all values 0 with i and all values 1 with j. Furthermore, we let $f_L(0)$ be equal to 0 in order to ensure that for recursive L we can effectively obtain a delayed simulation s_L of f_L in $\mathcal{F}$. We let

$$M_L \;=\; \{y \in \omega : s_L(y) = j\}.$$

We leave it to the reader to check that, first, the set M_L thus defined is a delayed simulation of L for all sets L where $L(0) = 0$ holds and that, second, the definition of M_L can be made effective in the sense that there is a recursive function sim where for all sets $L = \varphi_e$ the set M_L is equal to $\varphi_{\mathrm{sim}(e)}$. Then, in order to show that $\mathcal{M}_r$ is a simulation class, it is sufficient to show that for every recursive set L, the set M_L is in fact in $\mathcal{M}_r$. We assume that A_0, A_1, and B are sets where A_0 and A_1 are both r-reducible to B. The standard reducibility $\leq_r$ is c.f.v. and consequently A_0 and A_1 are reducible to $\langle B, \sigma_i\rangle$ and $\langle B, \sigma_j\rangle$, respectively. Let the two latter facts be witnessed by functionals Γ_0 and Γ_1 in $\mathcal{R}$, respectively, and consider the functional

$$\Gamma := \langle \Gamma_0, \Gamma_1 \rangle^{\{X : \sigma_i \sqsubseteq X\}},$$

obtained from Γ_0 and Γ_1 via definition by oracle-dependent cases; i.e., in particular, Γ is again in $\mathcal{R}$. Moreover, as σ_i and σ_j have been chosen to be incompatible, Γ reduces A_0 to $\langle B, \sigma_i\rangle$ and A_1 to $\langle B, \sigma_j\rangle$. We obtain

$$\langle A_0, A_1\rangle^{M_L} \;=\; \langle \Gamma(\langle B, \sigma_i\rangle), \Gamma(\langle B, \sigma_j\rangle)\rangle^{M_L} \;=^* \; [\Gamma \otimes s_L](B) \;\leq_r \; B. \tag{19}$$

Hence, by $\leq_r$ being c.f.v., the set $\langle A_0, A_1\rangle^{M_L}$ is reducible to B; and it follows that M_L is in $\mathcal{M}_r$. The relations in (19) hold by definition of Γ and of M_L and because $\Gamma \otimes s_L$ is an r-reduction due to the closure properties of $\mathcal{R}$. Note that in (19) we cannot replace equality up to finite variation by equality because in general the delayed simulation s_L of f_L yields the value $f_L(0) = 0$ on a finite initial segment of the natural numbers. ∎

By Propositions 37 and 39, for every standard reducibility the corresponding class of admissible cases satisfies the conditions on $\mathcal{M}$ in the assumption of Lemma 40.

LEMMA 40 (coding lemma). *Let $\mathcal{M}$ be a simulation class that contains all finite sets and where the structure $(\mathcal{M}, \subseteq)$ is a subalgebra of $(2^\omega, \subseteq)$. Let $A_0, A_1, \ldots$ be a sequence of uniformly recursive sets and let G be a gap language. Then there are sets $R_0, R_1, \ldots$ in $\mathcal{M}$ and a gap language M in $\mathcal{M}$ where*

- *the set M is a gap cover for G;*

- *for all i and s in ω and for all x in block s of M we have $R_i(x) = A_i(s)$.*

We postpone the proof of the coding lemma to section 5. The point of the coding lemma is that it yields delayed simulations R_i of the sets A_i that are "synchronized" via the gap language M; that is, for all i and s the set R_i is constant on block s of M and has the value $A_i(s)$ there. The sets R_i in the coding lemma are uniformly recursive because the sets A_i are uniformly recursive and due to the second condition in the conclusion of the coding lemma.

4. Lattice embeddings.

4.1. The countable atomless Boolean algebra. In the proof of our main result on lattice embeddings we exploit a property of the countable atomless Boolean algebra stated in Fact 42. The corresponding technique was used before in connection with lattice embeddings for polynomially time-bounded reducibilities by Ambos-Spies [1]; see there for references to results about Boolean algebras.

DEFINITION 41. *An element of a Boolean algebra is an atom iff there is exactly one element (i.e., the least element 0) strictly below it. A Boolean algebra is atomless iff it does not contain atoms.*

The theory of the atomless Boolean algebra is ω-categorical; i.e., all countable atomless Boolean algebras are the same up to isomorphism and accordingly in what follows we will speak of *the* countable atomless Boolean algebra.

FACT 42. *Every countable distributive lattice can be embedded (as a lattice) into the countable atomless Boolean algebra with least and greatest element preserved.*

By Fact 42 and because lattice embeddings compose, it suffices to embed the countable atomless Boolean algebra into a structure in order to show that indeed every countable distributive lattices can be so embedded. When constructing such embeddings we will exploit Fact 44, which gives a representation of the countable atomless Boolean algebra by the equivalence classes induced by the finite variation relation on the class $\mathcal{P}$ of sets computable in polynomial time.

DEFINITION 43. *For every subclass $\mathcal{C}$ of 2^ω, let $\mathcal{C}^*:=\{[A] : A \text{ in } \mathcal{C}\}$, where $[A]$ denotes the equivalence class $\{X : X =^* A\}$ of A in 2^ω with respect to the finite variation relation. Furthermore, let $\leq^*$ denote the p.o. induced by the relation $\subseteq^*$; i.e., $[A] \leq^* [B]$ iff $A \subseteq^* B$.*

FACT 44. *The structure $(\mathcal{P}^*, \leq^*)$ is the atomless Boolean algebra.*

Fact 44 has been shown by Ambos-Spies [1], using Breidbart's splitting theorem [10]. Fact 44 can be generalized to the assertion that for the class $\mathcal{M}_r$ of admissible cases of a standard reducibility, the structure $({\mathcal{M}_r}^*, \leq^*)$ is always the countable atomless Boolean algebra [20].

4.2. Lattice embeddings. Theorem 45 states our main technical result on lattice embeddings for standard reducibilities. Recall that the concept of standard reducibility comprises not just the reducibilities listed in Example 18 but also many other resource-bounded reducibilities that appear in the literature and observe that Theorem 45 and its corollaries can be applied to all such reducibilities.

THEOREM 45. *Let $\leq_r$ be a standard reducibility and let A and B be recursive sets where $A <_r B$. Then any countable distributive lattice can be embedded (as a lattice) into the interval between A and B of $(\mathrm{REC}, \leq_r)$ with least or greatest element preserved.*

In addition, given recursively presentable classes $\mathcal{E}_0$ and $\mathcal{E}_1$ that are both c.f.v. and where $\mathcal{E}_0$ does not contain $A \oplus \emptyset$ and $\mathcal{E}_1$ does not contain $A \oplus B$, then the range of the embedding can be chosen to be disjoint from the union of $\mathcal{E}_0$ and $\mathcal{E}_1$, except that in

case we want to preserve the least element, the latter might be mapped to an element of $\mathcal{E}_1$, and likewise for the greatest element and $\mathcal{E}_0$.

The proof of Theorem 45 follows the lines of the corresponding result for polynomially time-bounded reducibilities due to Ambos-Spies [1]. Similarly to the case of polynomially time-bounded reducibilities, we obtain as special cases of Theorem 45 several structural properties of standard reducibilities. We briefly discuss these results before proving the theorem. Another result, which can be derived from the proof of the theorem, is stated below as Corollary 53.

COROLLARY 46. *Let $\leq_r$ be a transitive standard reducibility.*

(i) *Every recursive set is the g.l.b. of a pair of recursive sets.*

(ii) *Every recursive set not in $\mathcal{L}_r$ is the l.u.b. of a pair of recursive sets.*

(iii) *Every recursive set not in $\mathcal{L}_r$ bounds a minimal pair (i.e., there are two sets that are both reducible to the given set and such that the intersection of their lower cones coincides with the class $\mathcal{L}_r$ of least sets).*

(iv) *Let A and B be recursive sets where $A<_r B$. Then every recursively presentable antichain in the open interval between A and B is not maximal with this property.*

Proof. The first assertion is immediate by embedding the diamond (i.e., the four-element Boolean algebra) above the given set with least element preserved. Likewise, the second and third assertion follow by embedding the diamond into the interval between $\emptyset$ and the given set with greatest and least element preserved, respectively. In connection with the third assertion observe that for a transitive relation $\leq_r$, first, the lower cone of a greatest lower bound of two sets is equal to the intersection of the lower cones of these two sets and, second, the lower cone of a least set is just the class of least sets.

In order to show the last assertion, let $\{C_0, C_1, \ldots\}$ be a recursively representable antichain where $A<_r C_i<_r B$ for all i in ω. In order to extend the antichain by a single set in the open interval between A and B, it suffices to embed the three-element chain between A and B according to Theorem 45 while avoiding the recursively presentable classes

$$\mathcal{E}_0 := \bigcup_{i\in\omega}\{X : C_i \leq_r X \leq_r B\} \text{ and } \mathcal{E}_1 := \bigcup_{i\in\omega}\{X : X \leq_r C_i\}. \qquad \square$$

Proof of Theorem 45. We first construct an embedding into the given interval that preserves the least element and then indicate the minor changes necessary in the symmetric case where we want to preserve the greatest element. By the discussion following Fact 42, it suffices to embed the countable atomless Boolean algebra as required. In order to do so, we specify a partial order embedding of the structure $(\mathcal{P}^*, \leq^*)$ that preserves least upper bounds. We can then argue that the restriction of this mapping to some appropriate sublattice $(\mathcal{D}^*, \leq^*)$ of $(\mathcal{P}^*, \leq^*)$ is in fact a lattice embedding, i.e., preserves also greatest lower bounds. This then finishes the proof because $(\mathcal{D}^*, \leq^*)$ is chosen such that it is the countable atomless Boolean algebra.

We fix effective enumerations $D_0, D_1, \ldots$ of $\mathcal{P}$ and $\Delta_0, \Delta_1, \ldots$ of an effective reduction cover $\mathcal{R}$ of $\leq_r$ that witnesses that $\leq_r$ is a standard reducibility. While defining the embedding, we first construct sets $R_0, R_1, \ldots$ in $\mathcal{M}_r$ and define a mapping

$$\Pi : D_i \mapsto H(R_i) \text{ where } H(X) := \langle A \oplus B, A \oplus \emptyset\rangle^X. \tag{20}$$

Then we let Π' be a function from $\mathcal{P}^*$ to $\mathcal{P}$ such that $\Pi'([D_i])$ is always a set in $[D_i]$

and we let

$$\Pi'' \;:=\; \Pi \circ \Pi'.$$

The function Π'' works by first mapping an equivalence class in $\mathcal{P}^*$ to one of its elements and then applying the function Π; hence, in particular, Π'' is well defined. Moreover, Claim 1 shows that the range of Π and thus also the range of Π'' is indeed contained in the interval between A and B.

CLAIM 1. *For every set R in $\mathcal{M}_r$, we have $A \leq_r H(R) \leq_r B$.*

Proof. By Proposition 5, the set $A \oplus \emptyset$ is r-equivalent to A and is hence reducible to B. Moreover, by faithfulness of $\leq_r$, the set $A \oplus B$ is a l.u.b. for A and B and is hence reducible to the upper bound B of these sets. As a consequence and by definition of $\mathcal{M}_r$, $H(R)$ is reducible to B for every set R in $\mathcal{M}_r$. Furthermore, by definition of the join operator, any set of the form $H(X)$ can be written in the form $A \oplus Z$ for an appropriate set Z; hence A is reducible to $H(X)$ by faithfulness of $\leq_r$. □

CLAIM 2. *The set $A \oplus B$ is not r-reducible to $A \oplus \emptyset$.*

Proof. Assuming otherwise, $A \oplus B$ was reducible to A by Proposition 5. But then contrary to assumption, B was also reducible to A because $A \oplus B$ is a locally transitive l.u.b. for B due to faithfulness of $\leq_r$. □

In order to define the sets R_i that we have used while defining Π, we construct three recursive gap languages, G_1 through G_3. First, we apply the diagonalization lemma to the sets $A \oplus \emptyset$ and $A \oplus B$ in order to obtain a gap language G_1 such that for all sets X and Y

$$(X, Y) \simeq^{G_1} (A \oplus B, A \oplus \emptyset) \quad \text{implies} \quad X \not\leq_r Y. \tag{21}$$

Second, we apply the window lemma to the recursively presentable class

$$\mathcal{C} \quad := \quad \{H(X) : X \leq_r \emptyset\} \tag{22}$$

in order to obtain a gap a gap language G_2 such that every reduction to a set in $\mathcal{C}$ is witnessed by a functional Δ in $\mathcal{R}$ where for all x,

$$\mathrm{u}(\Delta, x) \subseteq \mathrm{Nb}(x, G_2); \tag{23}$$

i.e., the generalized use of Δ at x is always contained in the neighborhood $\mathrm{Nb}(x, G_2)$ of x.

Third, we choose a recursive gap language G_3 such that, first, every set that is G_3-similar to $A \oplus \emptyset$ is not contained in $\mathcal{E}_0$ and, second, every set that is G_3-similar to $A \oplus B$ is not contained in $\mathcal{E}_1$. In order to do so, we choose block k of G_3 so large that both $A \oplus \emptyset$ and $A \oplus B$ disagree on this block with the first k sets in $\mathcal{E}_0$ and $\mathcal{E}_1$, respectively.

By the discussion preceding the coding lemma, the class $\mathcal{M}_r$ of admissible cases of the standard reducibility $\leq_r$ satisfies the conditions on the class $\mathcal{M}$ in the assumption of the coding lemma. By applying the coding lemma to $\mathcal{M}_r$, to the enumeration $D_0, D_1, \ldots$ of the class $\mathcal{P}$, and to a recursive gap cover G of G_1 through G_3, we obtain sets $R_0, R_1, \ldots$ in $\mathcal{M}_r$ and a set M in $\mathcal{M}_r$ that is a gap cover of G and hence also of G_1 through G_3.

CLAIM 3. *For all i and j, $D_i \subseteq^* D_j$ iff $R_i \subseteq^* R_j$.*

Proof. Claim 3 is immediate by choice of the sets $R_1, R_2, \ldots$ and by the second condition in the conclusion of Lemma 40. □

CLAIM 4. *For all i, the set $\Pi(D_i)$ is a finite variation of $\Pi''([D_i])$ and thus, in particular, the two sets are r-equivalent.*

Proof. Let j be the index such that D_j is equal to $\Pi'([D_i])$. Then, in particular, D_j is a finite variation of D_i and, by Claim 3, R_j is a finite variation of R_i. Now Claim 4 follows by the definitions of Π and Π''. □

In the following claims, the sets in the image of Π'' figure in such a way that the truth values of the assertions under consideration are not changed by replacing these sets with r-equivalent sets. Thus by Claim 3 in the corresponding proofs we can replace the images under Π'' with the corresponding images under Π.

CLAIM 5. *The function Π'' respects ordering.*

Proof. Assume $[D_i] \leq^* [D_j]$, i.e., $D_i \subseteq^* D_j$ and, by Claim 3, $R_i \subseteq^* R_j$. By the definition of Π, the set $\Pi(D_j)$ agrees with $A \oplus B$ on all places x in R_j and hence for almost all places x in R_i. We obtain

$$\Pi(D_i) = \langle A \oplus B, A \oplus \emptyset \rangle^{R_i} =^* \langle \Pi(D_j), A \oplus \emptyset \rangle^{R_i} \leq_r \Pi(D_j), \tag{24}$$

where the relations hold, from left to right, by definition of Π, by the preceding discussion, and finally because R_i is in $\mathcal{M}_r$ and because by Claim 1 the set A and hence by Proposition 5 also the set $A \oplus \emptyset$ is reducible to $\Pi(D_j)$. Due to $\leq_r$ being c.f.v., it is immediate from (24) that $\Pi(D_i)$ is reducible to $\Pi(D_j)$. □

CLAIM 6. *The function Π'' respects nonordering.*

Proof. Assume $[D_i] \not\leq^* [D_j]$, i.e., $D_i \not\subseteq^* D_j$ and, by Claim 3, $R_i \not\subseteq^* R_j$. Then there are infinitely many numbers and—because by construction the sets R_i and R_j are constant on the blocks of M—in fact there are infinitely many blocks of M where R_i is equal to 1, while R_j is equal to 0. By definition of Π, on each such block of M, $\Pi(D_i)$ agrees with $A \oplus B$ and $\Pi(D_j)$ agrees with $A \oplus \emptyset$; hence we have

$$(A \oplus B, A \oplus \emptyset) \simeq^M (\Pi(D_i), \Pi(D_j)). \tag{25}$$

By construction, M is a gap cover for the gap language G_1 obtained from applying the diagonalization lemma. Thus, by Remark 27, assertion (25) remains valid with M replaced by G_1 and $\Pi(D_i)$ is not reducible to $\Pi(D_j)$ because of (21). □

CLAIM 7. *The function Π'' respects l.u.b.'s.*

Proof. The l.u.b. of $[D_i]$ and $[D_j]$ in $(\mathcal{D}^*, \leq^*)$ is $[D_i \cup D_j]$. By Claim 5, the set $\Pi(D_i \cup D_j)$ is an upper bound for $\Pi(D_i)$ and $\Pi(D_j)$. So it remains to show that if the two latter sets are both reducible to some set Y, then so is $\Pi(D_i \cup D_j)$. The latter follows from

$$\begin{array}{rclcl} \Pi(D_i \cup D_j) & = & \langle A \oplus B, A \oplus \emptyset \rangle^{R_i \cup R_j} & = & \langle A \oplus B, \langle A \oplus B, A \oplus \emptyset \rangle^{R_j} \rangle^{R_i} \\ & = & \langle A \oplus B, \Pi(D_j) \rangle^{R_i} & = & \langle \Pi(D_i), \Pi(D_j) \rangle^{R_i} \leq_r Y, \end{array}$$

where the relations hold by the definition of Π and the choice of the sets R_k, by the properties of definition by number-dependent cases, by definition of $\Pi(D_j)$, because $\Pi(D_i)$ agrees with $A \oplus B$ on all numbers in R_i, and finally by assumption on Y and because R_i is in $\mathcal{M}_r$. □

Up until now we have seen that Π'' is a partial order embedding that preserves l.u.b.'s. We will now show that the restriction of Π'' to the class

$$\mathcal{D} := \{\{3 \cdot x : x \in D\} : D \in \mathcal{P}\}$$

also preserves g.l.b's. It is easy to see that $\mathcal{D}$ is a recursively presentable subclass of $\mathcal{P}$ and that $(\mathcal{D}^*, \leq^*)$ is isomorphic to $(\mathcal{P}^*, \leq^*)$. By the latter assertion and Fact 44, $(\mathcal{D}^*, \leq^*)$ is the countable atomless Boolean algebra.

CLAIM 8. *The restriction of the function Π'' to the subclass $\mathcal{D}^*$ of $\mathcal{P}^*$ respects g.l.b.'s (with respect to the structure $(\mathcal{D}^*, \leq^*)$).*

Proof. We fix sets D_i and D_j in $\mathcal{D}$. The g.l.b. of $[D_i]$ and $[D_j]$ in $(\mathcal{D}^*, \leq^*)$ is $[D_i \cap D_j]$ and so we have to show that $\Pi(D_i \cap D_j)$ is the g.l.b. of $\Pi(D_i)$ and $\Pi(D_j)$. By Claim 5, $\Pi(D_i \cap D_j)$ is a lower bound for the two latter sets. So it remains to show that if a set X is reducible to both of $\Pi(D_i)$ and $\Pi(D_j)$, then X is also reducible to $\Pi(D_i \cap D_j)$.

By Proposition 37, $\mathcal{M}_r$ is contained in the class $\mathcal{L}_r$ of least sets, which in turn is contained in the lower cone of $\emptyset$. Thus for every R in $\mathcal{M}_r$, the set $H(R)$ is in the class $\mathcal{C}$ defined in (22); i.e., in particular, the range of Π is contained in $\mathcal{C}$. Now G_2 has been obtained by applying the window lemma to the class $\mathcal{C}$ and $\mathcal{R}$; i.e., every reduction to a set in the range of Π is witnessed by a functional in $\mathcal{R}$ such that its generalized use at place x is always contained in $\mathrm{Nb}(x, G_2)$ and hence, by Remark 34, is always contained in $\mathrm{Nb}(x, M)$. As a consequence, the assumed reductions from X imply that there are functionals Δ_k and Δ_l in $\mathcal{R}$ with

$$X \;=\; \Delta_k(\Pi(D_i)) \;=\; \Delta_l(\Pi(D_j))$$

such that for all places x we have

$$\mathrm{u}(\Delta_k, x) \subseteq \mathrm{Nb}(x, M) \qquad \text{and} \qquad \mathrm{u}(\Delta_l, x) \subseteq \mathrm{Nb}(x, M). \tag{26}$$

For every m in ω, let $N(m)$ be equal to the set $\{m-1, m, m+1\} \setminus \{-1\}$ and let

$$\begin{aligned} I &:= \{m \in \omega : D_i \text{ and } D_i \cap D_j \text{ agree on } N(m)\}, \\ L &:= \{x \in \omega : x \text{ is in block } m \text{ of } M \text{ for some } m \text{ in } I\}. \end{aligned}$$

Then given x in L, we infer from the definition of Π that the sets $\Pi(D_i)$ and $\Pi(D_i \cap D_j)$ agree on $\mathrm{Nb}(x, M)$; hence by assumption on Δ_k we obtain that

$$x \in L \quad \text{implies} \quad \Delta_k(\Pi(D_i \cap D_j), x) \;=\; \Delta_k(\Pi(D_i), x) \;=\; X(x). \tag{27}$$

On the other hand, given x not in L, where we assume that x is in block m of M, then D_i and $D_i \cap D_j$ differ on $N(m)$; i.e., there must be some number in $N(m)$ that is in D_i but is not in D_j. However, the two latter sets contain only multiples of three, while for any m the set $N(m)$ contains exactly one multiple of three; hence the intersection of $N(m)$ with D_j is empty. Thus, the sets $\Pi(D_j)$ and $\Pi(D_i \cap D_j)$ have an empty intersection with $\mathrm{Nb}(x, M)$ and similar to (27) we infer

$$x \notin L \quad \text{implies} \quad \Delta_l(\Pi(D_i \cap D_j), x) \;=\; \Delta_l(\Pi(D_j), x) \;=\; X(x). \tag{28}$$

From (27) and (28) we then obtain

$$X \;=\; \langle \Delta_k(\Pi(D_i \cap D_j)), \Delta_l(\Pi(D_i \cap D_j)) \rangle^L \;.$$

This finishes the proof of Claim 8 because by definition I is in $\mathcal{P}$; i.e., I is equal to some set D_t; hence the set L is equal to R_t and is in $\mathcal{M}_r$. □

It remains to show the assertions on avoiding the classes $\mathcal{E}_0$ and $\mathcal{E}_1$ and on preserving the least or the greatest element. The former assertion follows easily from the definition of Π because G has been chosen as a gap cover of G_3 and because all equivalence classes in $\mathcal{D}^*$ that are strictly above the least equivalence class contain only sets that are infinite and coinfinite. Furthermore, the embedding Π preserves

the least element because it maps the least equivalence class in $\mathcal{D}^*$, which contains all finite sets, to a set that is r-equivalent to $A \oplus \emptyset$, where by Proposition 5 the latter set is r-equivalent to A, which is a least set of the interval between A and B. On the other hand, if we want to construct an embedding that preserves the greatest element, it suffices to replace in the construction the class $\mathcal{D}$ with the class

$$\{\{3 \cdot x : x \in D\} \cup \{3 \cdot x + 1 : x \in \omega\} \cup \{3 \cdot x + 2 : x \in \omega\} : D \in \mathcal{P}\},$$

i.e., to arrange that, intuitively speaking, the noncoding gaps of the sets in the image of the constructed embedding are filled with $A \oplus B$ instead of $A \oplus \emptyset$. Then the greatest equivalence class, which contains exactly the cofinite sets, is mapped to a set that is r-equivalent to $A \oplus B$. However, $A \oplus B$ is a locally transitive upper bound for B and thus any set that is reducible to B is also reducible to $A \oplus B$; hence the latter set is a greatest set in the interval bounded by A and B. □

4.3. Lattice embeddings for bounded reducibilities on ω^ω. The concept of standard reducibility extends canonically to binary relations on ω^ω. However, the concepts used in the definition of standard reducibilities have to be adjusted as follows.

(i) Effective reduction covers are defined with respect to the standard enumeration of partial recursive functionals from ω^ω to ω^ω.

(ii) In the definition of faithfulness, we consider arbitrary constant functions instead of just $\emptyset$ and ω; that is, for example, we require that for all functions f and for all constant functions g the lower cone of $f \oplus g$ is contained in the lower cone of f.

(iii) A tt-condition can again be defined to be a subclass of ω^ω where membership in the class depends only on the function values at a fixed finite set of places. However, when defining the concept of closure under definition by oracle-dependent cases, we consider only tt-conditions that can be generated from classes of the form $\{g : g(i) = j\}$ with i and j in ω by finitely many applications of union, intersection, and complement. Observe that it is not reasonable to require that closure under definition by oracle-dependent cases holds with respect to all tt-conditions because for any set X the class of all functions g where $g(0)$ is in X is a tt-condition.

(iv) In the definition of delayed patching the enumeration of finite partial characteristic functions is replaced by an appropriate effective enumeration of all partial functions from ω to ω with finite domain.

Using these adjusted concepts, the concept STANDARD REDUCIBILITY ON $\boldsymbol{\omega^\omega}$ can be introduced by literally the same formulation as in Definition 17; i.e., we require that the relation under consideration is faithful and c.f.v. and has an effective reduction cover that is closed under delayed patching and under definition by oracle-dependent cases. An example for a standard reducibility on ω^ω is given by the reducibility introduced by Mehlhorn [18] via his class of basic feasible functionals.

Theorem 45 on lattice embeddings for standard reducibilities on 2^ω extends to standard reducibilities on ω^ω.

THEOREM 47. *Let $\leq_r$ be a standard reducibility on ω^ω and let f and g be recursive functions where $f <_r g$. Then any countable distributive lattice can be embedded (as a lattice) into the interval between f and g of $(\mathrm{REC}, \leq_r)$ with least or greatest element preserved.*

In addition, given recursively presentable subclasses $\mathcal{E}_0$ and $\mathcal{E}_1$ of ω^ω that are c.f.v. and where $\mathcal{E}_0$ does not contain $f \oplus \emptyset$ and $\mathcal{E}_1$ does not contain $f \oplus g$, the range of the embedding can be chosen to be disjoint from the union of $\mathcal{E}_0$ and $\mathcal{E}_1$, except that in

case we want to preserve the minimum, the minimal element might be mapped to an element of $\mathcal{E}_1$, and likewise for the maximum and $\mathcal{E}_0$.

We omit the proof of Theorem 47, which is essentially the same as for Theorem 45 except that we have to take into account that in general the "use" of a recursive functional on ω^ω is unbounded. (Consider, for example, the functional Γ defined by $\Gamma(f,x) := f(f(x))$.) In order to handle this problem, we relativize the concept of generalized use to an appropriate effectively compact subclass of ω^ω, i.e., to a class $\mathcal{C}$ that can be written in the form

$$\mathcal{C} \quad = \quad \bigotimes_{i \text{ in } \omega} C_i, \tag{29}$$

where each set C_i is finite and a list of its elements can be computed from i. Defining the generalized use $\mathrm{u}^{\mathcal{C}}(\Delta, x)$ with respect to such an effectively given class $\mathcal{C}$ similarly as before, we obtain in particular that for every f in $\mathcal{C}$, the value $\Gamma(f,x)$ is determined by the restriction of f to the finite set $\mathrm{u}^{\mathcal{C}}(\Gamma, x)$. Furthermore, the following variants of the diagonalization lemma and the window lemma that are relativized to some effectively compact subclass of ω^ω can be shown as before.

LEMMA 48 (diagonalization lemma for reducibilities on $\boldsymbol{\omega^\omega}$). *Let $\leq_r$ be a bounded reducibility on ω^ω that is c.f.v., let e and f be recursive functions where $e \not\leq_r f$, and let $\mathcal{C}$ be some effectively compact subclass of ω^ω. Then there is a recursive gap language G such that we have for all functions e' and f' in $\mathcal{C}$*

$$(e,f) \simeq^G (e',f') \text{ implies } e' \not\leq_r f'.$$

LEMMA 49 (window lemma for reducibilities on $\boldsymbol{\omega^\omega}$). *Let $\mathcal{R}$ be a reduction cover for a reducibility on ω^ω and let the class $\mathcal{C}_0$ be recursively presentable such that $\mathcal{C}_0$ is contained in some effectively compact class $\mathcal{C}$. Then there is a recursive gap language G such that every reduction to a function in $\mathcal{C}_0$ is witnessed by a reduction $\Delta \in \mathcal{R}$, where $\mathrm{u}^{\mathcal{C}}(\Delta, x)$ is always contained in $\mathrm{Nb}(x, G)$.*

Lemmas 48 and 49 can be shown in exactly the same way as the original claims. The remainder of the proof of Theorem 45 then goes through by applying the adapted lemmas to the effectively compact class

$$\mathcal{C} := \bigotimes_{i \in \omega} \{[f \oplus \emptyset](i), [f \oplus g](i)\},$$

where f and g are the functions that bound the given interval. The class $\mathcal{C}$ contains all functions obtained from $f \oplus \emptyset$ and $f \oplus g$ via definition by cases with sets in $\mathcal{M}_r$, and thus $\mathcal{C}$ contains all functions in the range of the embedding Π.

4.4. Embeddings of partial orderings and decidability. In the following, we consider results on bounded reducibilities that are related to Theorem 45 on lattice embeddings for standard reducibilities but can be derived from more general assumptions. The first result of this type is Theorem 50, which amounts to an extension of Schöning's uniform diagonalization theorem [30] from a setting of polynomial time-bounds to arbitrary simulation classes. The content of Theorem 50 is roughly the same as Schmidt's Theorem 3.1 [29]; however, we formulate the result in our terms and consider only lower cones instead of arbitrary classes (see also Remark 54 below). In connection with Theorem 50, recall that a nonempty class $\mathcal{C}$ is recursively presentable if there is a recursive set E such that $\mathcal{C} = \{C_0, C_1, \dots, \}$ where $C_j = \{x : \langle x, j\rangle \text{ in } E\}$.

THEOREM 50 (Schmidt [29]). *Let $\leq_r$ be a bounded reducibility such that $\mathcal{M}_r$ is a simulation class. Then for any set A the lower r-cone of A cannot be the disjoint union of two nonempty recursively presentable classes that are c.f.v.*

Proof. Fix any set A and for a proof by contradiction assume that its lower r-cone is the disjoint union of two nonempty recursively presentable classes $\mathcal{C}_0$ and $\mathcal{C}_1$ that are c.f.v. For $i = 0, 1$, let the recursive set E_i witness that $\mathcal{C}_i$ is recursively presentable; i.e., $\mathcal{C}_i = \{C_0^i, C_1^i, \dots, \}$ where $C_j^i = \{x : \langle x, j\rangle \text{ in } E_i\}$. By assumption, we can choose sets D_0 and D_1 in the lower r-cone of A such that D_i is in $\mathcal{C}_{1-i}$ but not in $\mathcal{C}_i$.

We construct a recursive gap language G in stages $s = 0, 1, \dots$, where during stage s block s of G is defined. This then determines G by letting $G(0)$ be equal to 0. The first block I_0 just contains 0. Then, given the first s blocks $I_0, \dots, I_{s-1}$, the finite block I_s is chosen so large that every set that agrees on I_s with D_0 disagrees on I_s with C_0^0 through C_s^0 and, likewise, every set that agrees on I_s with D_1 disagrees there with C_0^1 through C_s^1. Such a block I_s exists (and can indeed be found effectively) because $\mathcal{C}_0$ and $\mathcal{C}_1$ are c.f.v.; hence D_i differs from each set in $\mathcal{C}_i$ at infinitely many places. We omit the details of the argument, which is similar to the one used in the proof of Lemma 28. Next, according to Lemma 59, we choose in the simulation class $\mathcal{M}_r$ a gap cover M for G. Then by construction and contrary to our assumption, the set

$$\langle D_0, D_1\rangle^M$$

is in the lower r-cone of A but is neither contained in $\mathcal{C}_0$ nor in $\mathcal{C}_1$. □

COROLLARY 51. *Let $\leq_r$ be a bounded reducibility that is faithful and c.f.v. and where the class $\mathcal{M}_r$ of admissible cases is a simulation class. Then the structure $(\mathrm{REC}, \leq_r)$ is dense.*

Proof. Fix recursive sets A and B where $A <_r B$ and consider the classes

$$\mathcal{C}_0 := \{X : X \leq_r A\} \qquad \mathcal{C}_1 := \{X : X \leq_r B \text{ and } B \leq_r X \oplus A\}.$$

By definition, the two classes are contained in the lower r-cone of B and by assumption on $\leq_r$ they are c.f.v. and recursively presentable. Thus by Theorem 50, the union of the two classes is strictly contained in the lower r-cone of B. However, for any set X that is r-reducible to B but is not in $\mathcal{C}_0$ or $\mathcal{C}_1$, the set $X \oplus A$ is contained in the open interval bounded by A and B. □

REMARK 52. *Let $\leq_r$ be a bounded reducibility that satisfies the assumption of Theorem 50 and let $\{\Delta_0, \Delta_1, \dots\}$ be an effective reduction cover for $\leq_r$. Consider two recursive sets A and B where A is r-reducible to B, but not vice versa, and let $\mathcal{D}$ be the nonempty class of all sets that are r-reducible to B but not to A. Then the class $\mathcal{D}$ cannot be recursively presentable. Otherwise, $\mathcal{D}$ and the lower r-cone of A were both recursively presentable and c.f.v., while the lower r-cone of B is the disjoint union of these two classes; i.e., we would obtain a contradiction to Theorem 50.*

By standard methods that involve the arithmetization of the computation of a Turing machine, we can construct an arithmetical formula δ in one free variable such that $\delta(e)$ is true in the standard model of the natural numbers iff $\Delta_e(B)$ is in $\mathcal{D}$. Then there cannot be an effective formal system in which we can derive the formula $\delta(e)$ exactly for the indices e such that $\delta(e)$ is true in the standard model. For a proof, observe that if there were such a formal system, then the set D of all such indices would be recursively enumerable and as a consequence, the class $\mathcal{D}$, which is equal to $\{\Delta_e(B) : e \in D\}$, would be recursively presentable. In a context of bounded reducibilities, results of this type have been obtained, among others, by Schöning [30]

and Regan [26]. *In a context of $\mathcal{NP}$ optimization problems similar results are given by Merkle* [21] *(unfortunately without mentioning the closely related and much earlier results of Schöning, Regan, and others, while just referring to the related work of Schmidt* [29]*).*

A strengthening of Corollary 51 to embeddings of partial orderings into intervals of the recursive sets can be obtained as a corollary to the proof of Theorem 45. A corresponding assertion has been stated by Mehlhorn [18] for reducibilities that satisfy his set of axioms.

COROLLARY 53. *Let $\leq_r$ be a faithful bounded reducibility that is c.f.v. and such that $\mathcal{M}_r$ is a simulation class. Then every countable partial ordering can be embedded into every proper interval of the structure* $(\mathrm{REC}, \leq_r)$.

Proof. It is known that every countable partial ordering can be embedded (as a partial ordering) into the countable atomless Boolean algebra. Furthermore, the proof of Theorem 45 shows that the assumption of Corollary 53 is sufficient for constructing partial order embeddings of the countable atomless Boolean algebra (indeed, of any countable distributive lattice) into any proper interval bounded by two recursive sets. So we are done because embeddings of partial orderings compose. □

REMARK 54. *The axiomatic approach of Schmidt* [29] *is based on the concept recursive gap closure, where a subclass $\mathcal{M}$ of 2^ω is* RECURSIVELY GAP CLOSED *iff for every recursive gap language G there is a set M in $\mathcal{M}$ such that the set M, as well as its complement, contain infinitely many blocks of G. (This formulation slightly deviates from the original definition, where only gap languages are considered for which strings of the same length are always in the same block.)*

Every simulation class $\mathcal{S}$ is recursively gap closed. For a proof, observe that given a recursive gap language G, then by Lemma 59 *there is some gap cover M for G in $\mathcal{S}$, and in particular M and its complement contain infinitely many blocks of G. On the other hand, there are recursively gap closed classes that are not simulation classes and that in fact do not even contain delayed simulations of all gap languages. For example, consider the subclass $\mathcal{D}$ of the recursive sets where for each set in the class there are infinitely many x where x is in the set, but $x-1$ and $x+1$ are not. Then for every set in $\mathcal{D}$, its characteristic function contains infinitely often the "pattern"* 010*. Hence the class $\mathcal{D}$, for example, contains neither a delayed simulation of the empty set nor of the recursive gap language that has the characteristic sequence* 011011011 ....

Schmidt demonstrates for several complexity classes that they are gap closed. From her proofs it is immediate that these classes are in fact simulation classes. In particular, we obtain that the class of sets simultaneously computable in linear time and logarithmic space is a simulation class. Note in this connection that Regan [26] *shows that even slightly more restrictive resource-bounds are sufficient for performing Ladner-style delayed diagonalization constructions.*

REMARK 55. *Regan and Vollmer* [28] *show results on partial order embeddings for various reducibilities of many-one type that run in logarithmic time. Typically, for given reduction function f and input x, within this time-bound just a single bit of $f(x)$ has to be computed. Hence in general for any unbounded function l, no matter how slow l grows, it will take more than logarithmic time to compute the complete value $f(x)$ even for inputs x with $f(x) \leq l(x)$. This indicates that such reducibilities will not allow patching of the oracle as required with standard reducibilities and that accordingly Theorem* 45 *on embeddings of countable distributive lattices does not apply. Yet, depending on the details of its definition, a reducibility restricted to logarithmic time will usually satisfy the assumption used to derive the results of this section.*

5. Proving the coding lemma. In this section we give the still missing proof of the coding lemma, which we have stated as Lemma 40. In its proof we use the recursion theorem in a form described in Remark 56.

REMARK 56. *If we specify a procedure that computes uniformly effectively in a given index e a partial function γ_e, then, by the smn-theorem, there is a recursive function g where $\varphi_{g(e)}$ is equal to γ_e for all e in ω. According to the recursion theorem, there is some index e_0, referred to as fixed point of g, where*

$$\varphi_{e_0} \;=\; \varphi_{g(e_0)} = \gamma_{e_0}.$$

In case we can ensure certain properties of the constructed partial function γ_e under the assumption that the given index e is an index for γ_e, we succeed in constructing a partial function with these properties because for the fixed point e_0 our assumption will be true.

In recursion theory, the following, more convenient form of this technique is widely used. Instead of viewing the index e as an argument, we assume that already during the definition of some partial recursive function γ we can use an index for γ, i.e., an index e with $\gamma = \varphi_e$. As Soare [33, p. 38] *points out, in the definition of γ we must not rely on any specific assumptions on e; i.e., while we are just interested in the cases where the number e used in our construction is indeed an index for the function under construction, in order to render the construction valid we have to ensure that the constructed partial function γ is in fact partial recursive in e.*

Subsequently, we will apply this technique by giving an effective procedure that enumerates the graph of a partial function γ, while using in the construction an index for γ. In order to address the problem mentioned in the last paragraph, we apply the following convention.

> *In case the partial function γ is eventually defined during the construction at a place x, then $\gamma(x)$ is equal to the value that is assigned first; otherwise, $\gamma(x)$ is undefined.*

It should be clear that by this convention there is always some recursive function g such that for all e, $\varphi_{g(e)}$ is equal to the function γ that we specify under the possibly wrong assumption that e is an index for the function under construction. For example, we can choose a function g that assigns to e, intuitively speaking, a Turing machine $M_{g(e)}$ that computes $\gamma(x)$ by simulating the construction of γ on the given index e until x enters the domain of γ. Then, if for some specific index e, the construction of γ "gets stuck", this simply means that the domain of $\gamma = \varphi_{g(e)}$ is finite and contains only the places that have been assigned values so far.

Remark 58 shows that in a simulation class we are not only able to find effectively delayed simulations of recursive sets but in fact we can find so—appropriately defined—delayed simulations of arbitrary partial recursive functions from ω to $\{0,1\}$.

DEFINITION 57. *A set S is a* delayed simulation *of a partial characteristic function α iff there is a nondecreasing function l from ω to ω where for all x in ω we have $S(x) = \alpha(l(x))$ and where the range of l is $\{z \in \omega : \alpha(y)$ is defined for all $y \leq z\}$.*

REMARK 58. *For every simulation class $\mathcal{S}$ there is a recursive function* sim *from ω to ω such that for all e in ω, first, $\varphi_{\mathrm{sim}(e)}$ is in $\mathcal{S}$ and, second, $\varphi_{\mathrm{sim}(e)}$ is a delayed simulation of φ_e in case $\varphi_e(0)$ is equal to 0. In fact, this is essentially the definition of simulation class except that now we require the second condition on delayed simulations for all e and not just for e where φ_e is a set.*

For a proof, recall from Example 10 *that the class of functions computable in polynomial time is a functional simulation class. Thus the class $\mathcal{P}$ of sets computable in*

polynomial time is a simulation class. Let sim_0 *be a witnessing function for the latter fact that corresponds to the type of simulation described in Example* 10. *Then* sim_0 *witnesses that the assertion we want to prove holds for* $\mathcal{S} = \mathcal{P}$. *As a consequence, if* sim_1 *witnesses that* $\mathcal{S}$ *is a simulation class, then the function* $\mathrm{sim}{:=}\mathrm{sim}_1 \circ \mathrm{sim}_0$ *has the required properties. More precisely, given some index* e *in* ω, *then* $\varphi_{\mathrm{sim}_1(\mathrm{sim}_0(e))}$ *is obviously in* $\mathcal{S}$. *Furthermore, if function* l_0 *witnesses that* $\varphi_{\mathrm{sim}_0(e)}$ *is a delayed simulation of* φ_e, *and* l_1 *witnesses that* $\varphi_{\mathrm{sim}_1(\mathrm{sim}_0(e))}$ *is a delayed simulation of* $\varphi_{\mathrm{sim}_0(e)}$, *then* $l_0 \circ l_1$ *witnesses that* $\varphi_{\mathrm{sim}(e)}$ *is a delayed simulation of* φ_e.

Before we prove the coding lemma, we consider the simpler assertion stated in Lemma 59 in order to demonstrate the techniques used. The notation and the subroutines used in the proof of Lemma 59 then are partially reused in the more involved proof of the coding lemma.

LEMMA 59. *Let* $\mathcal{S}$ *be a simulation class and let* G *be a recursive gap language. Then* $\mathcal{S}$ *contains a gap cover of* G.

Proof. Figure 1 shows an effective procedure that enumerates the graph of some partial characteristic function γ. By Remark 56, we can assume that an index e for the function under construction is already available during the construction; that is, we have $\gamma = \varphi_e$, and the specification of γ might depend on e.

Now $\mathcal{S}$ is a simulation class; hence we can choose a recursive function sim according to Remark 58; i.e., in particular, for all e in ω with $\varphi_e(0) = 0$, $\varphi_{\mathrm{sim}(e)}$ is a delayed simulation of φ_e in $\mathcal{S}$. Therefore, by letting $\gamma(0) = 0$, during the construction of γ we do not only have available some index e for γ but also the delayed simulation $M{:=}\varphi_{\mathrm{sim}(e)}$ of γ in $\mathcal{S}$. In the verification of the construction we then show that the set M in fact is a gap cover for G.

We first give an outline of the construction and sketch the ideas on which its verification is based. The course of values of γ is rather simple. If γ is defined at all at some place s, then it is equal to $\omega \oplus \emptyset$; that is, $\gamma(s)$ is 0 in case s is even, and $\gamma(s)$ is 1, otherwise. The actual load of the construction is to decide successively for $s = 1, 2, \ldots$ whether $\gamma(s)$ is to be defined or not. Thus, during the construction γ can always be written as

$$\gamma{:=}(\omega \oplus \emptyset)|\{0, \ldots, s\}, \tag{30}$$

where $s{:=}\max \mathrm{dom}(\gamma)$ is in block s of γ. (Recall that we start counting blocks with number zero.) In a situation where (30) holds, informally we denote block s of M as the open block of the construction. Now, by choice of sim and by definition of M, the set $M = \varphi_{\mathrm{sim}(e)}$ is a delayed simulation of $\gamma = \varphi_e$, and consequently at any point of the construction we have the following properties of the open block

(i) The open block exists because if γ has at least s blocks, then so does its delayed simulation M.

(ii) The open block is finite iff φ_e is defined at place $s + 1$.

The construction is based on the following idea: in the situation of (30) we refrain from defining γ at place $s + 1$ unless we can verify that the open block contains some block of G. As a consequence the open block indeed contains some block of G. In case γ indeed remained undefined at place $s+1$, then the open block would be infinite but would not contain a block of the gap language G, which is a plain contradiction. So, intuitively speaking, if we add block $s + 1$ of γ only after verifying that block s of M is large enough, then block s of M indeed is large enough. Next we give a formal proof of Lemma 59 by proving a series of claims.

CLAIM 1. *On entering stage s of the construction we have $\gamma{:=}(\omega \oplus \emptyset)|\{0,\dots,s\}$.*

Proof. Recall that the course of values of $\omega \oplus \emptyset$ is $01010\dots$. Claim 1 now follows by an easy induction argument that exploits the definitions of the construction and of the procedure *close_the_open_block*. □

CLAIM 2. *If the procedure* cover_some_block_of_G *is called during stage s of the construction, then the procedure terminates and block s of M contains some block of G.*

Proof. Let the procedure be called during some stage s of the construction. By Claim 1, on entering the procedure we have $s = \max \operatorname{dom}(\gamma)$ and γ has exactly s blocks. Thus the delayed simulation M has at least s blocks, and on start of the procedure z and y are set equal to the minimal element of block s of M, and the while loop is entered. During the iterations of the while loop, the set $\{z,\dots,y\}$ is always contained in block s of M, and consequently, by the condition in the head of the while loop, if the while loop is eventually left, then block s of M must contain some block of G. Now, assume for a contradiction that the while loop is never left and that consequently γ will remain undefined at place $s+1$ for the rest of the construction. Then the delayed simulation M of γ has exactly s blocks, and $M(y) = M(y+1)$ holds for all places y that are greater than the minimal element z in block s of M. So y goes to infinity, and the condition in the head of the while loop eventually becomes false because all blocks of the gap language G are finite. Hence the while loop is eventually left, contrary to our assumption. □

CLAIM 3. *M is a gap cover for G.*

Proof. By inspection of the construction and from Claim 2 we infer that the construction passes through all stages $s = 0, 1, \dots$. Thus Claim 1 shows that the delayed simulation M of γ has infinitely many blocks, and Claim 2 implies that each block of M contains some block of G; that is, M is a gap cover for G. □

Next we give the proof of the coding lemma, which has been already stated as Lemma 40. The proof of the coding lemma relies on the same idea as the proof of Lemma 59 but is combinatorially more involved.

LEMMA 60 (coding lemma). *Let $A_0, A_1, \dots$ be a sequence of uniformly recursive sets and let G be a recursive gap language. Let $\mathcal{M}$ be a simulation class that contains all finite sets and where the structure $(\mathcal{M}, \subseteq)$ is a subalgebra of $(2^\omega, \subseteq)$. Then there are sets $R_0, R_1, \dots$ in $\mathcal{M}$ and a gap language M in $\mathcal{M}$ such that*

- *the set M is a gap cover for G;*
- *for all i and s in ω and for all x in block s of M, it holds $R_i(x) = A_i(s)$.*

Proof. For a given gap language M, we call the blocks with numbers $0, 2, 4, \dots$ even blocks of M and, likewise, the remaining blocks of M are called odd. It is sufficient to show that given $\mathcal{M}$, G, and $A_0, A_1, \dots$ as in the assumption of the coding lemma, there is a gap language M in $\mathcal{M}$ and sets $S_1, S_2, \dots$ in $\mathcal{M}$ where, first, M is a gap cover for G and, second,

(i) for all x in an odd block s of M and for all i in ω, we have $S_{2i+1}(x) = A_i(s)$;

(ii) for all x in an even block s of M and for all i in ω, we have $S_{2i+2}(x) = A_i(s)$;

that is, for every i, the set S_{2i+2} is constant on each even block of M and attains on these blocks the values $A_i(0), A_i(2), A_i(4), \dots$, respectively, and a similar remark holds for the sets S_{2i+1} and the odd blocks of M. Given sets M and $S_1, S_2, \dots$ as above, we let

$$R_i := (S_{2i+1} \cap M) \cup (S_{2i+2} \cap \overline{M}),$$

where, due to $\mathcal{M}$ being closed under complement, we can assume $M(0) = 0$; i.e., M is the union of its odd blocks. By choice of the sets S_k, the sets R_i satisfy the

The Construction

Initially, we let $\gamma(0){:=}0$

Stage s of the construction

cover_some_block_of_G;
close_the_open_block;

The Subroutines

close_the_open_block
$m{:=}\max\operatorname{dom}(\gamma)$
$\gamma(m+1){:=}1-\gamma(m)$

cover_some_block_of_G
$z{:=}\min\{i\in\omega : \operatorname{bn}(M,i)=\max\operatorname{dom}(\gamma)\}$
$y{:=}z$
while $\{z,\dots,y\}$ *does not contain some block of* G
if $M(y)=M(y+1)$ *then* $y{:=}y+1$

FIG. 1. *The construction of the set M from the cover lemma.*

second condition required in the coding lemma. Furthermore, the sets R_i are in $\mathcal{M}$, because M and the set S_k are, and because $\mathcal{M}$ is a subalgebra of $(2^\omega,\subseteq)$. Note that it is actually sufficient to show that there are sets S_k that satisfy condition (i) and (ii) for almost all x because by assumption the subalgebra $\mathcal{M}$ contains all finite sets and is hence c.f.v.

We denote by γ^j row j of a partial characteristic function γ; that is, γ^j denotes the partial characteristic function that maps x to $\gamma(\langle x,j\rangle)$. We let r be a recursive function such that for all e the number $r(e,j)$ is an index for row j of φ_e, and we assume that $\mathcal{M}$ is a simulation class via a function sim; i.e., $\varphi_{\mathrm{sim}(e)}$ is a delayed simulation of φ_e whenever $\varphi_e(0)$ is equal to 0.

Similar to the proof of the cover lemma, we give an effective enumeration of the graph of a partial characteristic function γ where according to Remark 56 we can use an index e with $\gamma=\varphi_e$ during the specification of γ. In the construction of this enumeration, we will refer by γ to the finite partial characteristic function of which the graph has already been enumerated, and likewise we will refer by γ^j to row j of this intermediate partial characteristic function. For the index e given to the construction, we let

$$S_k \;:=\; \varphi_{\mathrm{sim}(r(e,k))} \qquad \text{and} \qquad M \;:=\; S_0 \;=\; \varphi_{\mathrm{sim}(r(e,0))};$$

that is, for all $k\geq 0$ the set S_k, which we informally denote as row k, is a delayed simulation of row k of φ_e. Note that thus for any given index e the meaning of M and S_k is fixed ahead of the construction. In the verification of the construction we

The Construction

⋆ Initially, the partial functions γ^j all have empty domain

Stage s of the construction

initialize_row(s)
for all $k \in \{1, \dots, s\}$ *where exactly one of* k *and* s *is odd*
  if $A_{[k \text{ div } 2]}(s-1) \neq A_{[k \text{ div } 2]}(s+1)$, *then*
    *compare(*0,*k)*
    close_block(k)
    *compare(k,*0*)*
cover_some_block_of_G
*close_block(*0*)*

FIG. 2. *The construction of the sets* M *and* S_k.

can then assume that the index e used in the construction is indeed an index for the function γ constructed in order to show that the sets M and $S_1, S_2, \dots$ have the required properties.

The construction is shown in Figures 2 and 3. Before we formally verify the construction, we give an informal description of stage s of the construction. Like in the verification of Lemma 59, at any stage of the construction, denote block $\max \operatorname{dom}(\gamma^k)$ as the open block of S_k. We assume that s is even, while the considerations for the symmetric case where s is odd are essentially the same. During stage s, first row s is initialized, and then we consider all rows $k \leq s$ where k is odd. We have to ensure that for all such k the sets S_k do not have a block-change in an odd block of M. Furthermore, if we let i be equal to k div 2, then in case $A_i(s-1)$ differs from $A_i(s+1)$, the set S_k must have exactly one block-change within block s of M, while in case $A_i(s-1)$ is equal to $A_i(s+1)$, the set S_k must not have a block-change within block s of M. In the latter case, we leave γ^k untouched, and otherwise we enforce the necessary block change by first forcing the open block of S_k to extend beyond the minimum of the open block of M, then closing the open block of S_k, and then extending the open block of M beyond the minimum of the (new) open block of S_k. These extensions are handled by the procedure *compare*, where by an invocation *compare(k,l)* we force the open block of S_l to contain an element that is greater or equal to the least element in the open block of S_k. Finally, we ensure that the open block of M contains some block of G and close the open block of M.

CLAIM 1. *Every call to close_the_open_block and to cover_some_block_of_G during the construction results in a terminating computation.*

Proof. The assertion is immediate in the case of the former procedure, while in the case of *cover_some_block_of_G* the argument is basically the same as for the corresponding claim in the proof of Lemma 59. □

In the remainder of the proof, we will use the following notation: a set I extends beyond the minimum of a set J iff we have $\min J \leq x$ for some x in I.

CLAIM 2. *Assume that on a call compare(k,l) during the construction we have*

$$m_k := \max \operatorname{dom}(\gamma^k) \quad \textit{and} \quad m_l := \max \operatorname{dom}(\gamma^l).$$

Some subroutines

⋆ The following two routines are the same as in Figure 1
⋆ with γ replaced by γ^j and γ^0, respectively

close_the_open_block(j)
$m{:=}\max\mathrm{dom}(\gamma^j)$
$\gamma^j(m+1){:=}1-\gamma^j(m)$

cover_some_block_of_G
$z{:=}\min\{i\in\omega:\mathrm{bn}(M,i)=\max\mathrm{dom}(\gamma^0)\}$
$y{:=}z$
while $\{z,\dots,y\}$ *does not contain some block of* G
if $M(y)=M(y+1)$, *then* $y{:=}y+1$

initialize_row(k)
$\gamma^k(0){:=}0$
if $A_{[k\ div\ 2]}(s)=1$, *then* $\gamma^k(1){:=}1$
compare(k,0)

compare(k,l)
$y{:=}\min\{i\in\omega:\mathrm{bn}(S_l,i)=\max\mathrm{dom}(\gamma^l)\}$
$z{:=}\min\{i\in\omega:\mathrm{bn}(S_k,i)=\max\mathrm{dom}(\gamma^k)\}$
while $y\not\leq z$
if $S_l(y)=S_l(y+1)$, *then* $y{:=}y+1$

FIG. 3. *Some subroutines used in the construction of the sets* M *and* S_i.

Then this call results in a terminating computation, and block m_l *of* S_l *extends beyond the minimum of block* m_k *of* S_k.

Proof. Similar to the case of the procedure *cover_some_block_of_G* we obtain that on entering *compare*, y and z are indeed set to the minimal elements in the open blocks of row k and l, respectively. Obviously, if the while loop is eventually left, then block m_l of S_l extends beyond the minimum of block m_k of S_k. Now, assuming that the while loop is never left, we infer that γ and a fortiori row l of γ will never be altered afterwards, and, consequently, the open block of row l is infinite and y goes to infinity; that is, the while loop is eventually left, contrary to our assumption. □

CLAIM 3. *The construction passes through infinitely many stages.*

Proof. The proof is immediate by the construction and Claims 1 and 2. □

CLAIM 4. *The set* M *is a gap cover for* G.

Proof. Similar to the proof of Claim 3 in the proof of Lemma 59 where γ is replaced with γ^0. □

We say that block-change n of a set C occurs within some set I iff the set C has at least $n+1$ blocks and the maximum of block n of C and the minimum of block $n+1$ are both in I.

CLAIM 5. *Let* C *and* D *be sets and let* I *be some block of* D. *If block* n *of* C *extends beyond the minimum of* I *and* I *in turn extends beyond the minimum of block* $n+1$ *of* C, *then block-change* n *of* C *occurs within* I.

Proof. The claim is immediate by the definition of the concepts involved. □

CLAIM 6. *For all k, s, and n in ω where $0 < k \leq s$, the number $n+1$ enters the domain of γ^k during stage s iff block-change n of S_k occurs in block s of M.*

Proof. We show first that if $n+1$ enters the domain of γ^k during stage $s \geq k$, then block-change n of S_k occurs in block s of M. In case $n+1$ enters the domain of γ^k during stage s, this is due to a call *close_the_open_block(k)* in the body of the *if* statement. So this call is preceded by a call *compare(0,k)*, and is followed by a call *compare(k,0)* where on both calls $\max \operatorname{dom}(\gamma^0)$ is equal to s, and $\max \operatorname{dom}(\gamma^k)$ is equal to n and $n+1$ on the first and on the second call, respectively. Then, if we let I be equal to block s of M, we obtain by Claim 4 that block n of S_k extends beyond the minimum of I and that in turn I extends beyond the minimum of block $n+1$ of S_k. Hence by Claim 5, the block-change n of S_k occurs in I. Conversely, if $n+1$ does not enter the domain of γ^k, then for S_k block-change n does not exist, and if $n+1$ enters the domain of γ^k during some stage s_0 different from s, then by the above argument block-change n of S_k occurs in block s_0 of M and not in block s. □

CLAIM 7. *For every k, each block of M contains at most one block-change of S_k.*

Proof. The proof is immediate from Claim 5 and the construction because at each stage at most one element enters the domain of any γ^k. □

CLAIM 8. *For all k, s in ω where $k \leq s$ and for $i := k \operatorname{div} 2$, the set S_k has a block-change in block s of M iff exactly one of k and s is odd and $A_i(s-1)$ differs from $A_i(s+1)$.*

Proof. It is obvious from the construction that any element enters the domain of γ^k during stage $s \geq k$ iff exactly one of k and s is odd and $A_i(s-1)$ differs from $A_i(s+1)$. So the assertion follows by Claim 6. □

CLAIM 9. *For every i in ω,*

for almost all x in an odd block s of M we have $S_{2i+1}(x) = A_i(s)$;
for almost all x in an even block s of M we have $S_{2i+2}(x) = A_i(s)$.

Proof. We show the assertion for the odd blocks and leave the similar proof for the even blocks to the reader. Let $k = 2i+1$ for some $i \geq 0$ and consider stage k of the construction. During the procedure *initialize_row*, row k of γ is changed such that $\gamma^k(\max \operatorname{dom}(\gamma^k))$ is equal to $A_i(s)$, and due to the call of the procedure *compare* and Claim 2 it follows that block s of M extends beyond the minimum of the open block of S_k. Now, the assertion follows by an easy induction argument because for each odd block $s = 2j+1 > k$ we have by Claim 8 that either $A_i(s-1)$ is equal to $A_i(s+1)$ and there occurs no block-change of S_k within the blocks $s-1$, s, and $s+1$ of M, or $A_i(s-1)$ and $A_i(s+1)$ differ and there occurs exactly one block-change of S_k within block s of M, but none within the blocks $s-1$ and $s+1$. □

Acknowledgments. We would like to thank Klaus Ambos-Spies for helpful discussions. Furthermore, we are grateful to the anonymous referees of the SIAM Journal on Computing for their comments and corrections.

REFERENCES

[1] K. AMBOS-SPIES, *Sublattices of the polynomial time degrees,* Inform. and Control, 65 (1985), pp. 63–84.

[2] K. AMBOS-SPIES, *Polynomial time reducibilities and degrees,* in Handbook of Computability Theory, E. R. Griffor, ed., Elsevier, Amsterdam, 1999, pp. 683–705.

[3] J. L. BALCÁZAR AND J. DÍAZ, *A note on a theorem by Ladner,* Inform. Process. Lett., 15 (1982), pp. 84–86.

[4] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, Vol. I, Springer-Verlag, Heidelberg, 1995.
[5] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, Vol. II, Springer-Verlag, Heidelberg, 1990.
[6] S. K. Basu, *On the structure of subrecursive degrees,* J. Comput. System Sci., 4 (1970), pp. 452–464.
[7] R. Book, D.-Z. Du, and D. Russo, *On polynomial and generalized complexity cores,* in Proceedings of the 3rd Annual Conference on Structure in Complexity Theory, J. Hartmanis, ed., Washington, D.C., 1988, IEEE Computer Society Press, 1988, pp. 236–250.
[8] R. V. Book, J. H. Lutz, and K. W. Wagner, *An observation on probability versus randomness with applications to complexity classes,* Math. Systems Theory, 27 (1994), pp. 201–209.
[9] R. V. Book, H. Vollmer, and K. W. Wagner, *Probabilistic type-2 operators and "Almost"-classes,* Comput. Complexity, 7 (1998), pp. 265–289.
[10] S. Breidbart, *On splitting recursive sets,* J. Comput. System Sci., 17 (1978), pp. 56–64.
[11] P. Chew and M. Machtey, *A note on structure and looking back applied to the relative complexity of computable functions,* J. Comput. System Sci., 22 (1981), pp. 53–59.
[12] K. Copestake, *On nondeterminism, enumeration reducibility and polynomial bounds,* Math. Logic Quart., 43 (1997), pp. 287–310.
[13] R. E. Ladner, *On the structure of polynomial time reducibility,* J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.
[14] R. E. Ladner, N. A. Lynch, and A. L. Selman, *A comparison of polynomial time reducibilities,* Theoret. Comput. Sci. 1 (1975), pp. 103–123.
[15] L. H. Landweber, R. J. Lipton, and E. L. Robertson, *On the structure of sets in* NP *and other complexity classes,* Theoret. Comput. Sci., 15 (1981), pp. 181–200.
[16] M. Machtey, *Minimal pairs of polynomial degrees with subexponential complexity,* Theoret. Comput. Sci., 2 (1976), pp. 73–76.
[17] K. Mehlhorn, *Polynomial and Abstract Subrecursive Classes*, Ph.d. dissertation, Cornell University, Ithaca, NY, 1974.
[18] K. Mehlhorn, *Polynomial and abstract subrecursive classes,* J. Comput. System Sci., 12 (1976), pp. 147–178.
[19] W. Merkle, *Exact pairs for abstract bounded reducibilities,* Math. Logic Quart., 45 (1999), pp. 343–360.
[20] W. Merkle, *A Generalized Account of Resource Bounded Reducibilities*, Doctoral dissertation, Mathematische Fakultät der Universität Heidelberg, INF 288, D-69120 Heidelberg, Germany, 1997.
[21] W. Merkle, *Structural properties of bounded relations with an application to* NP *optimization problems,* Theoret. Comput. Sci., 250 (2001), pp. 101–124.
[22] W. Merkle and Y. Wang, *Random separations and "Almost" classes for generalized reducibilities,* Math. Logic Quart., 47 (2001), pp. 249–269.
[23] W. Mueller, *Abstract Degree Structures*, Ph.d. dissertation, Mount Holyoke College, South Hadley, MA, 1991.
[24] P. Odifreddi, *Classical Recursion Theory,* Vol. I, North-Holland, Amsterdam, 1989.
[25] P. Odifreddi, *Classical Recursion Theory,* Vol. II, Elsevier, Amsterdam, 1999.
[26] K. W. Regan, *The topology of provability in complexity theory,* J. Comput. System Sci., 36 (1988), pp. 384–432.
[27] K. W. Regan and J. S. Royer, *On closure properties of bounded two-sided error complexity classes,* Math. Systems Theory, 28 (1995), pp. 229–243.
[28] K. W. Regan and H. Vollmer, *Gap-languages and log-time complexity classes,* Theoret. Comput. Sci., 188 (1997), pp. 101–116.
[29] D. Schmidt, *The recursion-theoretic structure of complexity classes,* Theoret. Comput. Sci., 38 (1985), pp. 143–156.
[30] U. Schöning, *A uniform approach to obtain diagonal sets in complexity classes,* Theoret. Comput. Sci., 18 (1982), pp. 95–103.
[31] U. Schöning, *Minimal pairs for P,* Theoret. Comput. Sci., 31 (1984), pp. 41–48.
[32] M. J. Serna, *The Parallel Approximability of P-Complete Problems*, Tesis doctoral, Universitat Politècnica de Catalunya, Barcelona, Spain, 1990.
[33] R. I. Soare, *Recursively Enumerable Sets and Degrees*, Springer-Verlag, Heidelberg, 1987.
[34] H. Vollmer, *The gap-language-technique revisited,* in Computer Science Logic '90, Lecture Notes in Comput. Sci. 533, E. Börger, H. Kleine-Büning, and M. M. Richter, eds., Springer-Verlag, Heidelberg, 1991, pp. 389–399.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1156–1168

DECIDABILITY OF TERMINATION OF GRID STRING REWRITING RULES*

ALFONS GESER†

Abstract. Termination of string rewriting is known undecidable. Termination of string rewriting with only one rule is neither known decidable nor known undecidable. This paper presents a decision procedure for rules $u \to v$ such that some letter b from u occurs as often or less often in v. We call such rules "grid" rules. By far most rules are grid rules. Grid rules cover all rules which terminate by a total division order. Thus total division orders are shown to be irrelevant for the termination problem of one-rule string rewriting.

Key words. semi-Thue system, string rewriting system, one-rule, termination, grid rule, total division order

AMS subject classifications. 06F05, 16S15, 20M05, 68Q42

PII. S009753979833297X

1. Introduction and related work. String rewriting systems (SRSs) are a widely used model for computation and reasoning in finitely generated monoids. SRSs can simulate Turing machines, so many interesting properties such as confluence, local confluence, termination, and decidability of the word problem are undecidable. An intriguing question is which restricted classes of SRSs still have Turing power and which do not.

SRSs that comprise of only *one rule* have decidable local confluence and confluence problems [7, 16]. We are interested in the open problem of whether termination, too, is decidable for one-rule SRSs. It is reasonable to expect that the study of this problem will have a major impact on termination of arbitrary SRSs.

The slightly more extended class of one-rule left-linear, nonoverlapping term rewriting systems has an undecidable termination problem [2]. String rewriting with three rules also suffices for an undecidable termination problem [10]. In contrast one-rule string rewriting looks too simple. As McNaughton [12] put it, "We seem to be able to decide whether any such system is uniformly terminating."

Kurth [7] started a systematic exploration of the termination problem for one-rule SRSs. He gave a number of criteria for termination and for the existence of looping reductions of lengths 1 or 2 (and of length 3 in [8]), and implemented a sieve that, from a lexicographic enumeration of all canonical representatives of one-rule SRSs, passes only those that satisfy none of his criteria.

Kurth arrived at a characterization of termination for all (100207 representatives of) length increasing SRSs $u \to v$, $|v| \leq 6$. He had to solve by hand 32 systems that passed his sieve. By the variety and difficulty of the proofs he was led to conjecture that termination is undecidable for one-rule SRSs.

Shikishima-Tsuji, Katsura, and Kobayashi [15] reduce the termination problem of confluent one-rule SRSs to the termination problem of nonoverlapping one-rule SRSs.

*Received by the editors January 23, 1998; accepted for publication (in revised form) July 17, 2001; published electronically April 12, 2002. This work was carried out while the author was employed at the Wilhelm-Schickard-Institut für Informatik, Universität Tübingen and was partially supported by grant Ku 966/3-1 of the Deutsche Forschungsgemeinschaft (DFG) within the Schwerpunkt Deduktion at the University of Tübingen.

http://www.siam.org/journals/sicomp/31-4/33297.html

†ICASE, NASA Langley Research Center, Mail Stop 132C, Hampton, VA 23681 (geser@icase.edu).

McNaughton [11] conjectures that nonoverlapping one-rule SRSs have a decidable termination problem.

McNaughton [11] gives a powerful decidable property, called "left barren," that entails termination of nonoverlapping SRSs. McNaughton [12], moreover, proves that "well-behaved" SRSs have a decidable termination problem, not only in the one-rule case. This class contains terminating SRSs as well as looping ones. Kobayashi, Katsura, and Shikishima-Tsuji [6] introduce a wider class of "gentle" rules but do not offer a decidability result for this class.

Zantema and Geser [17, 18] characterized termination for systems of the form $0^p1^q \to 1^r0^s$. Sénizergues [14] showed that termination is decidable for the class $\{0^p1^q \to v \mid p, q \geq 1, v \in \{0,1\}^*\}$. For the same class, Kobayashi, Katsura, and Shikishima-Tsuji [6] added a complete characterization.

In this paper we contribute a decidability result for a wide class of rules.

DEFINITION 1.1. *A string rewriting rule $u \to v$ is called a* grid *rule if some letter b from u occurs in v as often or less often.*

The case "less often" is trivial; "as often" is the interesting case. We obtain as the main result of this paper the following characterization.

THEOREM 1.2. *Every nonterminating grid rule has a loop of length* 1 *or* 2.

By Kurth's characterizations (see Theorems 4.2 and 4.3), the existence of a loop of length 1 or 2 is decidable, whence termination of grid rules is decidable. It can even be shown that there is a linear-time decision procedure. Moreover, it is possible to extract an exponential upper bound on reduction lengths from the proof of Theorem 1.2. The interested reader may find proofs of these two claims in the technical report [3].

The class of grid rules is indeed a wide class, not only for its trivial part. It contains all rules that terminate by a total division order (Theorem 6.3). Total division orders are a widely used method for termination proofs in string rewriting. For instance, the recursive path order on strings [9] or the Knuth–Bendix order on strings [5] are total division orders. In spite of an enormous research effort, the space of total division orders is largely unexplored. Our results show that total division orders are irrelevant for the termination problem of one-rule SRSs.

The main part of the paper provides a proof of Theorem 1.2. After the preliminaries (section 2) we derive termination criteria of ascending strength. In section 3 we show that $u \to v$ terminates unless u and v have certain shapes. Particularly, we identify two strings, x_0 and x_n, on which the termination behavior crucially depends. In section 4 we give proofs for the remaining hard cases and finish the proof of Theorem 1.2. Section 5 gives an impression of the distribution of grid rules for the case $|v| = 7$. In section 6 we state and prove Theorem 6.3.

2. Preliminaries. A string is a sequence of letters from a given alphabet. Variables ranging over letters will be denoted by a, b, c, d; variables ranging over strings by z, y, x, etc., and indexed variants thereof.

We denote the empty string by ε; the concatenation of strings s and t by st; the length of a string s by $|s|$; the number of occurrences of a letter b in s by $\#_b(s)$.

Let $u \to v$ be a one-rule SRS over alphabet Σ. The induced reduction relation $\to$ is defined by the following: $u' \to v'$ if there are strings s, t such that $u' = sut$ and $v' = svt$.

s is called a *factor* of t if $t = xsy$ for some strings x, y; a *prefix* if $t = sy$ for some string y; a *suffix* if $t = xs$ for some string x. A prefix or suffix s of t is called *proper* if $s \neq t$. A string s is said to have the self-overlap t if $t \neq \varepsilon$, $s \neq t$, and there are strings x, y such that $s = xt = ty$.

Let $\to^*$ denote the reflexive-transitive closure of $\to$; i.e., $s \to^* t$ denotes that there is a reduction from s to t. Then $u \to v$ is called *confluent* if for every forking pair of reductions, $w_0 \to^* w_1$ and $w_0 \to^* w_2$, there is a joining pair of reductions, $w_1 \to^* w_3$ and $w_2 \to^* w_3$. By a result of Wrathall [16], confluence of length increasing one-rule SRSs is characterized by the property that all self-overlaps of u are also self-overlaps of v. This is the case, particularly, if $u \to v$ is *nonoverlapping*; i.e., u has no self-overlap.

The SRS $u \to v$ is called *terminating* if there is no infinite reduction sequence $u_1 \to u_2 \to \cdots$. Conveniently, termination is proven by giving a well-founded order $>$ on strings such that $sut > svt$ holds for all strings s, t.

An order $>$ on Σ induces a lexicographic order $>_{lex}$ on Σ^* by the following: $s = a_0a_1 \cdots a_n >_{lex} b_0b_1 \cdots b_n = t$ if (i) t is a proper prefix of s or (ii) there is $0 \le i \le \min\{m, n\}$ such that $a_i > b_i$ and $a_j = b_j$ for all $0 \le j < i$. It is known that for a well-founded order $>$ its lexicographic order $>_{lex}$ is well founded on strings of bounded length.

Every string s admits a unique *decomposition*, $s = s_0bs_1b \ldots bs_n$, where b does not occur in s_i, $0 \le i \le n$. The *tuple representation*, $T(s)$, of a string s is the sequence of strings $(s_0, s_1, \ldots, s_n)$ if $s = s_0bs_1b \ldots bs_n$ is the decomposition of s. For f, a function from strings to a set $\mathcal{D}$, the tuple representation *through* f is the sequence $T_f(s) = (f(s_0), f(s_1), \ldots, f(s_n))$ of elements of $\mathcal{D}$.

Our class of SRSs is particularly amenable to tuple representations (through any f) for proving termination by a lexicographic order because the tuples $T_f(sut)$ and $T_f(svt)$ have the same length.

3. The simple cases. Throughout the paper let $u \to v$ be a string rewriting rule, and let Σ denote the least alphabet containing all letters of uv. Let $b \in \Sigma$ be a letter such that $\#_b(u) > 0$ and $\#_b(u) \ge \#_b(v)$; i.e., $u \to v$ is a grid rule. If $\#_b(u) > \#_b(v)$, then $u \to v$ terminates (see, e.g., [7, Kriterium A]). This is the trivial case. Henceforth let $\#_b(u) = \#_b(v) = n$ for some $n > 0$. Then u and v are uniquely decomposed as

$$u = u_0bu_1 \cdots bu_n,$$
$$v = v_0bv_1 \cdots bv_n$$

for some $u_i, v_i \in (\Sigma \setminus \{b\})^*$, $0 \le i \le n$. In this section we are going to show that $u \to v$ terminates unless the u_i and v_i have a particular shape.

LEMMA 3.1. *If u_0 is not a suffix of v_0, then $u \to v$ terminates.*

By duality w.r.t. reversal of strings, one obtains at once the following: If u_n is not a prefix of v_n, then $u \to v$ terminates.

$acba \to accbac$ terminates by Lemma 3.1 because ac is not a suffix of acc. Likewise, $acbcbcbac \to acabacbcbac$ terminates by Lemma 3.1 because ac is not a suffix of aca. Note that in these examples the left-hand side of the rule is a subsequence of the right-hand side of the rule. This shows that termination cannot be proven by a division order (see also section 6).

Proof. Let $sut \to svt$ be an arbitrary reduction step by the rule $u \to v$. Let s' be the longest suffix of s that does not contain the letter b, and let t' be the longest prefix of t that does not contain the letter b. Distinguish cases whether $|u_0| > |v_0|$ or not.

Case $|u_0| > |v_0|$. Form the b-tuple representation through $f(w) = |w|$. Since $|s'u_0| > |s'v_0|$, one gets

$$\begin{aligned} T_f(sut) &= (\dots, |s'u_0|, |u_1|, \dots, |u_{n-1}|, |u_n t'|, \dots) \\ &> {}_{lex}(\dots, |s'v_0|, |v_1|, \dots, |v_{n-1}|, |v_n t'|, \dots) = T_f(svt), \end{aligned}$$

where the leftmost and the rightmost dots indicate the same left and right contexts.

Case $|u_0| \le |v_0|$. Form the b-tuple representation through

$$f(w) = \begin{cases} 1 & \text{if } w \text{ has suffix } u_0, \\ 0 & \text{else.} \end{cases}$$

By premise, then $f(s'u_0) = 1$ and $f(s'v_0) = 0$. Now

$$\begin{aligned} T_f(sut) &= (\dots, 1, f(u_1), \dots, f(u_{n-1}), f(u_n t'), \dots) \\ &> {}_{lex}(\dots, 0, f(v_1), \dots, f(v_{n-1}), f(v_n t'), \dots) = T_f(svt). \end{aligned}$$ □

Henceforth let us exclude the case where Lemma 3.1 or its dual apply. In other words we will assume that $v_0 = x_0 u_0$ and $v_n = u_n x_n$ for some strings x_0, x_n. That is,

$$u = \quad u_0 b u_1 \dots b u_{n-1} b u_n, \tag{3.1}$$

$$v = x_0 u_0 b v_1 \dots b v_{n-1} b u_n x_n \tag{3.2}$$

for some $x_0, x_n, u_0, \dots, u_n, v_1, \dots, v_{n-1} \in (\Sigma \setminus \{b\})^*$.

Now if $u_i = v_i$ for all $1 \le i \le n-1$, then u is a factor of v. Particularly for $n = 1$ one gets $v = x_0 u_0 b u_1 x_1$ which contains $u = u_0 b u_1$ as a factor; this already proves the case $n = 1$ of Theorem 1.2. If u is not a factor of v, then $n \ge 2$.

In the following lemma we use the regular expression $y x_0^* u_0$ to denote the set of strings $\{y x_0^p u_0 \mid p \ge 0\}$.

LEMMA 3.2. *If for some $1 \le i \le n-1$ there is no string y such that $u_i \ne v_i$ and $u_i, v_i \in y x_0^* u_0$, then $u \to v$ terminates.*

In *abba* $\to$ *aabaabaa* the factor $u_1 = \varepsilon$ has not the suffix $u_0 = a$ and $v_1 = aa \ne u_1$. In *bab* $\to$ *aabaaba*, one gets $u_0 = \varepsilon$, and so $u_1 = a = y u_0$ with $y = a$. Here $v_1 = aa \notin a(aa)^* \varepsilon = y x_0^* u_0$. Both SRSs are instances of Lemma 3.2, one per case of the proof below.

Again, by duality one gets termination if, for some $1 \le i \le n-1$, there is no string z such that one has $u_i \ne v_i$ and $u_i, v_i \in u_n x_n^* z$.

Proof. Let i be minimal. We distinguish cases whether or not u_i has suffix u_0.

Case 1. u_i has no suffix u_0. Then form the b-tuple representation through

$$f(w) = \begin{cases} 1 & \text{if } w = u_i, \\ 0 & \text{else.} \end{cases}$$

Note that $f(w) = 0$ holds for all strings w that have suffix u_0. Particularly, $f(s'u_0) = 0 = f(s'x_0u_0)$. By minimality of i one gets $f(u_j) = f(v_j)$ for all $1 \le j \le i-1$, either because $u_j = v_j$ or because both u_j and v_j have suffix u_0. So

$$\begin{aligned} T_f(sut) &= (\dots, 0, f(u_1), \dots, f(u_{i-1}), 1, f(u_{i+1}), \dots, f(u_n t'), \dots) \\ &> {}_{lex}(\dots, 0, f(v_1), \dots, f(v_{i-1}), 0, f(v_{i+1}), \dots, f(u_n x_n t'), \dots) = T_f(svt). \end{aligned}$$

Case 2. If u_i has suffix u_0, then there is y such that $u_i \in yx_0^*u_0$. Let y be shortest: y has no suffix x_0. By premise, now $v_i \notin yx_0^*u_0$. Form a b-tuple representation through

$$f(w) = \begin{cases} 1 & \text{if } w \in yx_0^*u_0, \\ 0 & \text{else.} \end{cases}$$

Let s' denote the longest suffix of s that does not contain b, and let t' denote the longest prefix of t that does not contain b. If $s' \in yx_0^*$, then $f(s'u_0) = 1 = f(s'x_0u_0)$, or else $f(s'u_0) = 0 = f(s'x_0u_0)$. By minimality of i, for all $1 \le j \le i-1$ either $u_j = v_j$, and so $f(u_j) = f(v_j)$, or there is a string y' that has no suffix x_0 such that $u_j, v_j \in y'x_0^*u_0$. If $y' = y$, then $f(u_j) = 1 = f(v_j)$, or else $f(u_j) = 0 = f(v_j)$. Then

$$\begin{aligned} &T_f(sut) \\ &\quad = (\dots, \quad f(s'u_0), f(u_1), \dots, f(u_{i-1}), 1, f(u_{i+1}), \dots, f(u_{n-1}), f(u_nt'), \dots) \\ &\quad >_{lex} (\dots, f(s'x_0u_0), f(v_1), \dots, f(v_{i-1}), 0, f(v_{i+1}), \dots, f(v_{n-1}), f(u_nx_nt'), \dots) \\ &\quad = T_f(svt). \qquad \square \end{aligned}$$

One can draw a few consequences of Lemma 3.2 that are useful during section 4.

LEMMA 3.3. *If $x_0 = \varepsilon$ and u is not a factor of v, then $u \to v$ terminates.*

Proof. If $x_0 = \varepsilon$ and $u_i, v_i \in yx_0^*u_0$ for some string y, then $u_i = yu_0 = v_i$. So let $x_0 = \varepsilon$ and $u_i \ne v_i$ for some $1 \le i \le n-1$. Then there is no y such that $u_i, v_i \in yx_0^*u_0$. Lemma 3.2 entails the claim. $\square$

LEMMA 3.4. *If $|u_i| = |v_i|$ and $u_i \ne v_i$ for some $1 \le i \le n-1$, then $u \to v$ terminates.*

Proof. If $x_0 = \varepsilon$, then the claim follows by Lemma 3.3, so let $x_0 \ne \varepsilon$. If $|u_i| = |v_i|$, $u_i = yx_0^pu_0$, $v_i = yx_0^{p'}u_0$, then $p = p'$ and hence $u_i = v_i$. So let for some $1 \le i \le n-1$ both $|u_i| = |v_i|$ and $u_i \ne v_i$. Then there is no y such that $u_i, v_i \in yx_0^*u_0$. By Lemma 3.2 the claim follows. $\square$

LEMMA 3.5. *Let $x_0 \ne \varepsilon$. Define the equivalence relation $\sim$ on Σ^* by $s \sim t$ if*

$$|s| \equiv |t| \mod |x_0|.$$

If $u_i \not\sim v_i$ for some $1 \le i \le n-1$, then $u \to v$ terminates.

Proof. If $u_i = yx_0^pu_0$, $v_i = yx_0^{p'}u_0$ for some y, then $|u_i| - |v_i| = (p-p')|x_0|$. So if $|u_i| - |v_i|$ is not an integer multiple of $|x_0|$, then $u_i \ne v_i$ and there is no y such that $u_i, v_i \in yx_0^*u_0$. Lemma 3.2 then implies the claim. $\square$

This finishes the conclusions drawn from Lemma 3.2. Let us now cover two further simple cases.

LEMMA 3.6. *Let $\Delta = |x_0ux_n| - |v|$. If $|x_0| \ge |x_n|$ and $\Delta > |x_0|$, then $u \to v$ terminates or $u = v$.*

Proof. By Lemma 3.5 $u \to v$ terminates if for some $1 \le i \le n-1$, $|u_i| - |v_i|$ is not an integer multiple of $|x_0|$. So let $|u_i| - |v_i|$ be an integer multiple of $|x_0|$ for all $1 \le i \le n-1$. Then $\Delta = \sum_{1 \le i \le n-1}(|u_i| - |v_i|)$ must be an integer multiple of $|x_0|$. By premise, then $\Delta \ge 2|x_0|$. So

$$|u| - |v| = \Delta - |x_0| - |x_n| \ge |x_0| - |x_n| \ge 0;$$

hence the rule is terminating unless $u = v$ (see, e.g., [13]). $\square$

LEMMA 3.7. *If $|u_\ell| < |v_\ell|$ for some $1 \le \ell \le n-1$ and $|u_j| = |v_j|$ for all $1 \le j \le \ell - 1$, then $u \to v$ terminates.*

By duality $u \to v$ terminates, too, if $|u_\ell| < |v_\ell|$ for some $1 \le \ell \le n-1$ and $|u_j| = |v_j|$ for all $\ell + 1 \le j \le n-1$.

The two SRSs *bbab* $\to$ *abbaaabaa* and *babaab* $\to$ *abaaabbaa* are not confluent as the self-overlap b of u is not a self-overlap of v. The two SRSs terminate by Lemma 3.7.

Proof. Form the b-tuple representation through

$$f(w) = \begin{cases} 1 & \text{if } |w| \le |u_\ell|, \\ 0 & \text{else.} \end{cases}$$

To prove $T_f(sut) >_{lex} T_f(svt)$ let s' be the longest suffix of s that does not contain b, and let t' denote the longest prefix of t that does not contain b. By definition of f one gets $f(s'u_0) \ge f(s'x_0u_0)$ and $f(u_j) = f(v_j)$ for all $1 \le j \le \ell - 1$; hence

$$\begin{aligned} &T_f(sut) \\ &\quad = (\dots, \quad f(s'u_0), f(u_1), \dots, f(u_{\ell-1}), 1, f(u_{\ell+1}), \dots, f(u_{n-1}), f(u_nt'), \dots) \\ &\quad >_{lex} (\dots, f(s'x_0u_0), f(v_1), \dots, f(v_{\ell-1}), 0, f(v_{\ell+1}), \dots, f(v_{n-1}), f(u_nx_nt'), \dots) \\ &\quad = T_f(svt). \qquad \square \end{aligned}$$

By Lemma 3.4, u not a factor of v and $|u_\ell| = |v_\ell|$ for all $1 \le \ell \le n-1$ implies termination. Together with Lemma 3.7 this yields the following.

LEMMA 3.8. *Let u not be a factor of v. Then $u \to v$ terminates unless there is $1 \le \ell \le n-1$ such that $|u_\ell| > |v_\ell|$ and $|u_j| = |v_j|$ for all $1 \le j \le \ell - 1$.*

4. The hard cases. Throughout this section let (3.1) and (3.2) hold.

Let $\mathsf{maxu} = \max\{|u_i| \mid 1 \le i \le n-1\}$. Call a string w that contains no b and such that $|w| > \mathsf{maxu}$ a *wall*. Intuitively, a wall separates rewrite steps; the factor $bu_1bu_2 \dots bu_{n-1}b$ of u never overlaps with a wall. The only overlaps u can form with a wall are u_0 and u_n. Application of $u \to v$ then can only increase the wall by x_0, x_n, respectively.

Let us call a string $bw_0bw_1 \dots bw_{N-1}bw_Nb, N \ge 2$ a *basin* if w_0 and w_N are walls and $w_1, \dots, w_{N-1}$ are no walls and $w_1bw_2 \dots bw_{N-1} \ne v_1bv_2 \dots bv_{n-1}$. Let u not be a factor of v. Then basins are the only locations where rewrite steps may still take place. Particularly, u is not a factor of any string $bw'bv_1 \dots bv_{v-1}bw''b$, where w', w'' are walls since u is not a factor of v, and so $bu_1bu_2 \dots bu_{n-1}b \ne bv_1bv_2 \dots bv_{n-1}b$. The only way to increase by a rewrite step the number of basins is to turn a nonwall into a wall.

We are going to distinguish the cases $|x_0| > |x_n|$ and $|x_0| = |x_n|$. For these two cases we have only two complicated termination proofs to offer.

LEMMA 4.1. *If $|x_0| > |x_n|$ and u is not a factor of v then $u \to v$ terminates.*

By duality $u \to v$ also terminates if $|x_n| > |x_0|$ and u is not a factor of v. The system *babaab* $\to$ *aababba* is an example where Lemma 4.1 applies. The system *abaab* $\to$ *aabbaa* for which Kurth gave an ad hoc proof [7, p. 127], is an example where the dual of Lemma 4.1 applies. Both systems are not confluent.

Proof. First let us state a few properties that we will use during the proof.

Since Lemma 3.3 handles the case $|x_0| = 0$, we may assume $|x_0| > 0$. Lemma 3.5 leaves the case where

$$u_i \sim v_i \quad \text{for all } 1 \le i \le n-1. \tag{4.1}$$

By premise and the dual of Lemma 3.3 we have $|x_0| > |x_n| > 0$, and hence

$$x_n \not\sim 0. \tag{4.2}$$

By Lemma 3.8 we need to prove termination only in the case where there is some $1 \le \ell \le n-1$ such that

$$|u_\ell| > |v_\ell| \quad \text{and} \quad |u_j| = |v_j| \quad \text{for all } 1 \le j \le \ell - 1. \tag{4.3}$$

Obviously ℓ is unique. Let $\Delta = |x_0 u x_n| - |v|$. By Lemma 3.6, termination is proven if $\Delta > |x_0|$. So assume

$$\Delta \le |x_0|. \tag{4.4}$$

If there is an infinite reduction, $w_1 \to w_2 \to \dots$, and p, p' are arbitrary strings, then $pw_1p' \to pw_2p' \to \cdots$ is an infinite reduction as well. One may choose p and p' such that pw_1p' has prefix bw' and suffix $w''b$, where w', w'' are walls. This property is preserved by reduction steps.

We prove the claim by showing a strict decrease from sut to svt, where $sut \to svt$ is a reduction step and s starts with bw' and t ends with $w''b$ and w', w'' are walls. Let s' and t' denote the longest suffix of s and the longest prefix of t, respectively, that contains no letter b.

We apply five termination functions lexicographically to prove termination. More precisely, we construct $f_1, \dots, f_5$ such that

$$(f_1(sut), \dots, f_5(sut)) >_{lex} (f_1(svt), \dots, f_5(svt)).$$

Termination function f_1. The termination function f_1 yields the number of nonwalls: $f_1(bw_0bw_1 \dots bw_{m-1}bw_mb) = \sum_{0 \le i \le m} g_1(w_i)$, where the w_i contain no letter b and where g_1 is defined by

$$g_1(w) = \begin{cases} 1 & \text{if } |w| \le \mathsf{maxu}, \\ 0 & \text{else.} \end{cases}$$

One gets that $f_1(sut) \ge f_1(svt)$:

$$\begin{aligned} f_1(sut) - f_1(svt) &= f_1(bs'ut'b) - f_1(bs'vt'b) \\ &= \underbrace{g_1(s'u_0) - g_1(s'x_0u_0)}_{\ge 0} + \underbrace{f_1(bu_1bu_2 \dots bu_{n-1}b)}_{=n-1} \\ &\quad - \underbrace{f_1(bv_1bv_2 \dots bv_{n-1}b)}_{\le n-1} + \underbrace{g_1(u_nt') - g_1(u_nx_nt')}_{\ge 0} \ge 0. \end{aligned}$$

If one of the $|v_i|$, $1 \le i \le n-1$, is greater than maxu, then $f_1(bv_1bv_2 \dots bv_{n-1}b) < n-1$ and hence $f_1(sut) > f_1(svt)$. In this case we have already proved termination. This leaves the case where

$$|v_i| \le \mathsf{maxu} \quad \text{for all } 1 \le i \le n-1. \tag{4.5}$$

If a wall is created, i.e., $g_1(s'u_0) > g_1(s'x_0u_0)$ or $g_1(u_nt') > g_1(u_nx_nt')$, then again $f_1(sut) > f_1(svt)$ and termination is already proven. So henceforth one may assume that the number of walls is not changed during a rewrite step:

$$g_1(s'u_0) = g_1(s'x_0u_0), \tag{4.6}$$

$$g_1(u_nt') = g_1(u_nx_nt'). \tag{4.7}$$

We are going to distinguish four cases:

$$|s'u_0| > \mathsf{maxu},\ |s'x_0u_0| > \mathsf{maxu},\ |u_nt'| > \mathsf{maxu},\ |u_nx_nt'| > \mathsf{maxu}, \tag{4.8}$$

$$|s'u_0| > \mathsf{maxu},\ |s'x_0u_0| > \mathsf{maxu},\ |u_nt'| \le \mathsf{maxu},\ |u_nx_nt'| \le \mathsf{maxu}, \tag{4.9}$$

$$|s'u_0| \le \mathsf{maxu},\ |s'x_0u_0| \le \mathsf{maxu},\ |u_nt'| > \mathsf{maxu},\ |u_nx_nt'| > \mathsf{maxu}, \tag{4.10}$$

$$|s'u_0| \le \mathsf{maxu},\ |s'x_0u_0| \le \mathsf{maxu},\ |u_nt'| \le \mathsf{maxu},\ |u_nx_nt'| \le \mathsf{maxu}\,. \tag{4.11}$$

As explained in detail below, the termination functions f_2, f_3, f_4, and f_5 take care of the cases (4.8), (4.9), (4.10), and (4.11), respectively.

Termination function f_2. The termination function f_2 assigns to a string w its number of basins. A basin cannot be created during a rewrite step that preserves f_1. A basin is destroyed in case (4.8): $f_2(sut) - f_2(svt) = 1$ because $bs'ut'b$ is a basin, whereas $bs'vt'b$ is not. Hence termination is proven for case (4.8), and the termination proof of cases (4.9), (4.10), and (4.11) is postponed.

Termination function f_3. Based on premises (4.3) and (4.1) the auxiliary function g_3 is defined as follows:

$$g_3(bw_0bw_1 \dots bw_{n-1}bw_nb) = \begin{cases} 2 & \text{if } |w_\ell| = |u_\ell| \text{ and } w_n \not\sim u_{n-\ell}, \\ 1 & \text{if } |w_\ell| \le |u_\ell|,\ w_\ell \sim u_\ell, \text{ and } w_n \sim u_{n-\ell}, \\ 0 & \text{else.} \end{cases}$$

Next, let the set $W(w)$ of strings be defined by

$$W(w) = \{bw_0bw_1 \dots bw_{n-1}bw_nb \mid w_0, \dots, w_n \in (\Sigma \setminus \{b\})^*, \\ |w_0| > \mathsf{maxu}, |w_1|, \dots, |w_n| \le \mathsf{maxu}\}.$$

Observe that by definition two strings $w', w'' \in W(w), w' \ne w''$ have the only overlap b. Now the termination function f_3 is defined by

$$f_3(w) = \sum_{w' \in W(w)} g_3(w').$$

We are going to prove that $f_3(sut) > f_3(svt)$ in case (4.9) and $f_3(sut) \ge f_3(svt)$ in cases (4.10) and (4.11).

Case (4.9). $|s'u_0| > \mathsf{maxu}$ and $|u_nt'| \le \mathsf{maxu}$ and so

$$f_3(sut) - f_3(svt) = g_3(bs'ut'b) - g_3(bs'vt'b) \\ = \begin{cases} 2 & \text{if } u_nt' \not\sim u_{n-\ell}, \\ 1 & \text{else} \end{cases} \quad - \quad \begin{cases} 1 & \text{if } u_nx_nt' \sim u_{n-\ell}, \\ 0 & \text{else} \end{cases} \\ = \begin{cases} 2-1 & \text{if } u_nx_nt' \sim u_{n-\ell}, \\ 1-0 & \text{if } u_nt' \sim u_{n-\ell} \\ 2-0 & \text{else.} \end{cases} \quad \ge 1,$$

The case $1 - 1$ cannot occur since $u_nx_nt' \not\sim u_nt'$ by (4.2). Thus termination of case (4.9) is proven.

In the remaining cases (4.10) and (4.11), where $|s'u_0| \le \mathsf{maxu}$, we have to prove $f_3(sut) \ge f_3(svt)$. Let $s = bs_1bs_2 \dots bs_{m-1}bs'$, $m \ge 1$, $s_1, \dots, s_{m-1}, s' \in (\Sigma \setminus \{b\})^*$,

and let D be the least positive integer such that $|s_{m-D}| > \mathsf{maxu}$. ($D$ is well defined, for $|s_1| > \mathsf{maxu}$ is presupposed.) Let $w = bw_0bw_1 \dots bw_nb$ denote the unique string that begins and ends with b, contains b exactly $n+2$ times, and such that sut has prefix $bs_1bs_2 \dots bs_{m-D-1}w$. Likewise, let $w' = bw'_0bw'_1 \dots bw'_nb$ denote the unique string that begins and ends with b, contains b exactly $n+2$ times, and such that svt has prefix $bs_1bs_2 \dots bs_{m-D-1}w'$.

There are walls neither in $bs'u_0bu_1 \dots bu_{n-1}b$ nor in $bs'x_0u_0bv_1 \dots bv_{n-1}b$ by premises $|s'u_0| \le \mathsf{maxu}$, (4.5), and (4.6); so one gets

$$f_3(sut) - f_3(svt) = g_3(w) - g_3(w'). \tag{4.12}$$

We are going to prove that $g_3(w) - g_3(w') \ge 0$.

The string w is a prefix of the string

$$\underbrace{bs_{m-D} \dots bs_{m-1}b}_{1}\, s'u_0 \underset{\uparrow}{b}\underbrace{u_1 \dots bu_{n-\ell-1}b}_{3}\, u_{n-\ell}\underset{\uparrow}{b}\underbrace{u_{n-\ell+1} \dots bu_{n-1}b}_{5}\, u_nt'b.$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad 2 \qquad\qquad\qquad\quad 4$$

The string w may end in any of the domains $1, \dots, 5$. By premise, it must not end at $u_nt'b$. Accordingly we perform a case analysis on n.

Case 1. $n < D$. Then $w_\ell = s_{m-D+\ell} = w'_\ell$ and $w_n = s_{m-D+n} = w'_n$ and so $g_3(w) = g_3(w')$.

Case 2. $n = D$. Then $w_\ell = s_{m-D+\ell} = w'_\ell$, $w_n = s'u_0$, and $w'_n = s'x_0u_0$ and so $g_3(w) = g_3(w')$ by $s'u_0 \sim s'x_0u_0$.

Case 3. $D < n < D+n-\ell$ or, equivalently, $\ell < D < n$. Then $w_\ell = s_{m-D+\ell} = w'_\ell$, $w_n = u_{n-D}$, $w'_n = v_{n-D}$; hence $w_n \sim w'_n$ by (4.1) which implies that $g_3(w) = g_3(w')$.

Case 4. $n = D+n-\ell$ or, equivalently, $D = \ell$. Then $w_\ell = s'u_0$, $w'_\ell = s'x_0u_0$, $w_n = u_{n-\ell}$, and $w'_n = v_{n-\ell}$. Note that $g_3(w) \ne 2$ and $g_3(w') \ne 2$ by $w_n \sim u_{n-\ell} \sim w'_n$. Hence $g_3(w) \ge g_3(w')$ by $|w_\ell| \le |w'_\ell|$ and by $w_\ell \sim w'_\ell$.

Case 5. $D+n-\ell < n \le D+n-1$ or, equivalently, $1 \le D < \ell$. Then $w_\ell = u_{\ell-D}$, $w'_\ell = v_{\ell-D}$, $w_n = u_{n-D}$, and $w'_n = v_{n-D}$. From $|w_\ell| = |w'_\ell|$ by (4.3) and $w_n \sim w'_n$ we infer $g_3(w) = g_3(w')$.

Summarizing this case analysis, we have shown $g_3(w) \ge g_3(w')$, and so $f_3(sut) \ge f_3(svt)$ by (4.12), in the remaining cases (4.10) and (4.11). These cases are again postponed.

Termination function f_4. The termination function f_4 is defined by

$$f_4(bw_0bw_1 \dots bw_{m-1}bw_mb) = \sum_{0 \le i \le m} g_4(w_i),$$

where $w_i \in (\Sigma \setminus \{b\})^*$ for all $0 \le i \le m$, and g_4 is defined by

$$g_4(w) = \begin{cases} \mathsf{maxu} - |w| & \text{if } |w| \le \mathsf{maxu}, \\ 0 & \text{else.} \end{cases}$$

Recall that $\Delta = |x_0ux_n| - |v| \le |x_0|$ by premise (4.4).

Case (4.11). Here $|s'x_0u_0| \le \mathsf{maxu}$ and $|u_nx_nt'| \le \mathsf{maxu}$, so $f_4(sut) > f_4(svt)$:

$$\begin{aligned} &f_4(sut) - f_4(svt) \\ &\qquad = \underbrace{g_4(s'u_0) - g_4(s'x_0u_0)}_{=|x_0|} - \underbrace{\Delta}_{\le |x_0|} + \underbrace{g_4(u_nt') - g_4(u_nx_nt')}_{=|x_n|} \ge |x_n| > 0. \end{aligned}$$

Case (4.10). $|s'x_0u_0| \leq \mathsf{maxu}$ and $|u_nt'| > \mathsf{maxu}$ hold and so

$$f_4(sut) - f_4(svt) = \underbrace{g_4(s'u_0) - g_4(s'x_0u_0)}_{=|x_0|} - \underbrace{\Delta}_{\leq |x_0|} + \underbrace{g_4(u_nt') - g_4(u_nx_nt')}_{=0} \geq 0.$$

Because the decrease in case (4.10) need not be strict we need a fifth termination function f_5.

Termination function f_5. $f_5(w)$ counts the factors in w of the form

$$bw_0u_0bu_1 \dots bu_{n-1}bu_nw_nb,$$

where u_nw_n is a wall. We have $f_5(sut) - f_5(svt) = 1$ for the remaining case (4.10) by the fact that $u_i \neq v_i$ for some $1 \leq i \leq n-1$.

We conclude that $(f_1(sut), \dots, f_5(sut)) >_{lex} (f_1(svt), \dots, f_5(svt))$ for every step $sut \to svt$. Hence $u \to v$ terminates. □

At last, there is the case $|x_0| = |x_n|$. Here we show that $u \to v$ terminates unless it admits a loop of lengths 1 or 2. A loop is a reduction sequence where the start string reappears as a factor in the final string. Kurth characterized the existence of loops of lengths 1 and 2 for arbitrary one-rule SRSs.

THEOREM 4.2 (Kurth [7, p. 139]). *A string rewriting rule $u \to v$ admits a loop of length 1 if and only if u is a factor of v.*

THEOREM 4.3 (Kurth [7, p. 142]). *If u is not a factor of v, then the following two propositions are equivalent:*

1. *$u \to v$ has a loop of length 2;*
2. *there are strings g, h, k such that $u = gh$, $v = hk$, and ggh is a factor of hkk.*

The case where a grid rule $u \to v$ admits a loop of length 2 really exists, as witnessed by the SRS $abaab \to aababa$. It has the following loop of length 2:

$$abaabab \to aababaab \to aabaababa.$$

To apply Kurth's characterization, choose $g = ab$, $h = aab$, $k = aba$. Then $ggh = ab\,ab\,aab$, which is a factor of $hkk = aab\,aba\,aba$.

As a surprising fact, neither loops of other lengths nor infinite nonlooping reductions are essential to grid rules: It suffices to exclude loops of lengths 1 and 2 to ensure termination.

LEMMA 4.4. *If $|x_0| = |x_n|$ and $u \to v$ does not terminate, then $u \to v$ has a loop of length 1 or 2.*

Proof. Let us exclude loops of length 1.

Suppose that $u \to v$ is nonterminating; i.e., there is an infinite reduction sequence. First we proceed as in the proof of Lemma 4.1. The termination functions f_1, f_2, and f_4 as defined in the proof of Lemma 4.1 are applied lexicographically so as to show that an infinite reduction exists where the values of f_1, f_2, and f_4 remain constant. (Termination function f_3 yields no progress in the case $|x_0| = |x_n|$ because the crucial property (4.2) fails to hold.) We have to show that $f_4(sut) \geq f_4(svt)$ in case (4.9); this is the dual to case (4.10). We refer to Lemma 4.1 for details. From such an infinite reduction we are going to extract a loop of length 2.

The infinite reduction sequence must finally contain steps of sort case (4.9) or case (4.10) exclusively, as these are the only steps that may keep $f_1, \dots, f_4$ constant. Because f_1 is finally constant, the number of basins is bounded. By the pigeonhole principle, infinitely many steps of the infinite reduction sequence take place in one basin. Steps of sort case (4.9) take place with u_0 overlapping the left wall of the

basin; steps of sort case (4.10) take place with u_n overlapping the right wall of the basin. Each step strictly increases the thickness of its assigned wall. As soon as a wall is as thick as $|u_n| + |u_0|$, rewrite steps at opposite sides of the wall can be done independently because redexes will no longer overlap. So an infinite reduction can be extracted that starts from a single basin. If two steps of the same sort follow each other, then $v_i = u_i$ holds for all $1 \leq i \leq n-1$, a contradiction. Hence the steps of the two kinds must alternate. By the same argument they must also overlap. However, with that we have the required loop of length 2. □

Theorem 1.2 now follows immediately from Lemma 3.1 and its dual and Lemmas 4.1 and 4.4.

5. A glimpse of practice. A few figures may illustrate the distribution of grid rules $u \to v$ for $|v| = 7$. They are obtained using the author's reimplementation of Kurth's sieve. There are 3,151,054 length increasing rule representatives in total. They divide into 3,145,038 grid rules and 6,016 nongrid rules. As much as 3,004 nongrid rules have loops of length 1 and 57 have loops of length 2 but not of length 1. Among the grid rules 2,736,925 rules satisfy $\#_b(u) > \#_b(v)$ for some letter b; 7,118 rules have loops of length 1; 400,007 rules satisfy Lemma 3.1 or 3.3 or their duals; 964 rules satisfy one of Lemmas 3.2, 3.6, 3.7, or 4.1, or their duals; and the following 16 rules are proper, terminating instances of Lemma 4.4 or its dual:

$abaabb \to aababba$	$abbaab \to aabbaba$	$baabb \to aabbbaa$
$baaabb \to abaabba$	$baabab \to abaabba$	$baabab \to abababa$
$baabbb \to ababbba$	$bababb \to ababbba$	$babbab \to ababbba$
$bababb \to abbabba$	$babbbb \to abbbbba$	$bbabbb \to abbbbba$
$bbcabc \to abbcbca$	$bcabbc \to abcbbca$	$babccb \to abbccba$
$bacbb \to acbbbac$		

Finally, the following 8 rules admit loops of length 2:

$abaaab \to aabaaba$	$baaab \to aababaa$	$abaaba \to aababaa$
$baaaab \to abaaaba$	$babbab \to abbabba$	$bbaabb \to abbabba$
$bcaabc \to abcabca$	$bcacb \to acbcbca$	

6. Total division orders. A *division order* [4] is an order $>$ on strings over an alphabet Σ such that

- $s > t$ if s is a factor of t;
- if $s > t$, then $usv > utv$ for all $u, v \in \Sigma^*$.

If $s > t$ or $t > s$ or $s = t$ for all $s, t \in \Sigma^*$, then $>$ is called *total*. Termination of a SRS R is usually shown by constructing a total division order $>$ such that $u > v$ for each rule $u \to v$ of R. We say that in this case R *is ordered by the total division order* $>$.

Total division orders and grid rules are closely related by the following result of Cohen and Scott.

Definition 6.1 (dominates, rational [1]). *A string $u \in \Sigma^*$ is said to* dominate *$v \in \Sigma^*$ if $\#_b(u) > \#_b(v)$ for all $b \in \Sigma$. An order $> \subseteq \Sigma^* \times \Sigma^*$ is called* rational *if $u > v$ holds for all $u, v \in \Sigma$ such that u dominates v.*

Theorem 6.2 (Cohen and Scott [1]). *Every total division order on strings is rational.*

Theorem 6.3. *Every string rewriting rule that can be ordered by a total division order is a grid rule.*

Proof. Assume that $u \to v$ is not a grid rule. Then v dominates u, and so $v > u$ for every total division order $>$ by Theorem 6.2. If $>$ orders $u \to v$, then $v > u > v$, a contradiction to irreflexivity of $>$. □

In effect, this means that total division orders are irrelevant for the termination problem of one-rule SRSs. Note, however, that Theorem 6.3 does not imply decidability whether a rule can be ordered by a total division order.

7. Conclusion. Termination of SRSs is an undecidable property. Whether termination is undecidable even for *one-rule SRSs* stays an exciting open problem. This paper solves this problem partially: Termination is decidable for the class of $u \to v$, where some letter from u occurs in v as often or less often. We call such rules *grid* rules.

All rules that can be ordered by a total division order are grid rules (Theorem 6.3), so a termination sieve can dispose of all total division orders. Even better: the only interesting rules left are the nongrid rules. These can be enumerated directly, and their number grows far slower than the total number of length increasing rules. Thus the sieve becomes faster and finer and gives access to larger and more interesting rules.

Acknowledgments. The idea to pursue this work was triggered by Kurth's "Criterion C" and by McNaughton's Theorem 1.4. Hans Zantema suggested to consider reduction lengths and pointed out an error in a preliminary version of the proof of Lemma 3.1. One referee gave very useful hints which helped improve the presentation.

REFERENCES

[1] D. Cohen and E. Scott, *The rationality of total division orderings*, Inform. Process. Lett., 44 (1992), pp. 307–311.

[2] M. Dauchet, *Simulation of Turing machines by a regular rewrite rule*, Theoret. Comput. Sci., 103 (1992), pp. 409–420.

[3] A. Geser, *Decidability of Termination in a Large Class of One-Rule String Rewriting Systems*, Tech. Rep. WSI 97-11, Wilhelm-Schickard-Institut, Universität Tübingen, Tübingen, Germany, 1997.

[4] G. Higman, *Ordering by divisibility in abstract algebras*, Proc. London Math. Soc. (3), 2 (1952), pp. 326–336.

[5] D. E. Knuth and P. B. Bendix, *Simple word problems in universal algebras*, in Computational Problems in Abstract Algebra, J. Leech, ed., Pergamon Press, Oxford, UK, 1970, pp. 263–297.

[6] Y. Kobayashi, M. Katsura, and K. Shikishima-Tsuji, *Termination and derivational complexity of confluent one-rule string rewriting systems*, Theoret. Comput. Sci., 262 (2001), pp. 583–632.

[7] W. Kurth, *Termination und Konfluenz von Semi-Thue-Systemen mit nur einer Regel*, dissertation, Technische Universität Clausthal, Clausthal-Zellerfeld, Germany, 1990.

[8] W. Kurth, *One-rule semi-Thue systems with loops of length one, two, or three*, RAIRO Inform. Théor. Appl., 30 (1996), pp. 415–429.

[9] P. Lescanne, *Two implementations of the recursive path ordering on monadic terms*, in Proceedings of the 19th Allerton House Conference on Communication, Control, and Computing, University of Illinois Press, Urbana-Champaign, IL, 1981, pp. 634–643.

[10] Y. Matiyasevitch and G. Sénizergues, *Decision problems for semi-Thue systems with a few rules*, in Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, NJ, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 523–531.

[11] R. McNaughton, *The Uniform Halting Problem for One-Rule Semi-Thue Systems*, Tech. Rep. 94-18, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1994. See also "Correction to 'The Uniform Halting Problem for One-Rule Semi-Thue Systems,'" unpublished paper, August, 1996.

[12] R. McNaughton, *Well-Behaved Derivations in One-Rule Semi-Thue Systems*, Tech. Rep. 95-15, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1995. See also "Correction by the author to 'Well-Behaved Derivations in One-Rule Semi-Thue Systems,'" unpublished paper, July, 1996.

[13] Y. Métivier, *Calcul de longuers de chaînes de réécriture dans le monoïde libre*, Theoret. Comput. Sci., 35 (1985), pp. 71–87.

[14] G. Sénizergues, *On the termination problem for one-rule semi-Thue Systems*, in Rewriting Techniques and Applications, H. Ganziger, ed., Lecture Notes in Comput. Sci. 1103, Springer, Berlin, 1996, pp. 302–316.

[15] K. Shikishima-Tsuji, M. Katsura, and Y. Kobayashi, *On termination of confluent one-rule string rewriting systems*, Inform. Process. Lett., 61 (1997), pp. 91–96.

[16] C. Wrathall, *Confluence of one-rule Thue systems*, in Word Equations and Related Topics, K. U. Schulz, ed., Lecture Notes in Comput. Sci. 572, Springer, Berlin, 1992.

[17] H. Zantema and A. Geser, *A complete characterization of termination of* $\mathtt{0}^p\mathtt{1}^q \to \mathtt{1}^r\mathtt{0}^s$, in Rewriting Techniques and Applications, J. Hsiang, ed., Lecture Notes in Comput. Sci. 914, Springer, Berlin, 1995, pp. 41–55.

[18] H. Zantema and A. Geser, *A complete characterization of termination of* $0^p1^q \to 1^r0^s$, Appl. Algebra Engrg. Comm. Comput., 11 (2000), pp. 1–25.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1169–1183

RANDOMNESS, COMPUTABILITY, AND DENSITY*

ROD G. DOWNEY†, DENIS R. HIRSCHFELDT‡, AND ANDRÉ NIES§

Abstract. We study effectively given positive reals (more specifically, computably enumerable reals) under a measure of relative randomness introduced by Solovay [manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975] and studied by Calude, Hertling, Khoussainov, and Wang [*Theoret. Comput. Sci.*, 255 (2001), pp. 125–149], Calude [*Theoret. Comput. Sci.*, 271 (2002), pp. 3–14], Kučera and Slaman [*SIAM J. Comput.*, 31 (2002), pp. 199–211], and Downey, Hirschfeldt, and LaForte [*Mathematical Foundations of Computer Science* 2001, Springer-Verlag, Berlin, 2001, pp. 316–327], among others. This measure is called *domination* or *Solovay reducibility* and is defined by saying that α dominates β if there are a constant c and a partial computable function φ such that for all positive rationals $q < \alpha$ we have $\varphi(q)\downarrow < \beta$ and $\beta - \varphi(q) \leqslant c(\alpha - q)$. The intuition is that an approximating sequence for α generates one for β whose rate of convergence is not much slower than that of the original sequence. It is not hard to show that if α dominates β, then the initial segment complexity of α is at least that of β.

In this paper we are concerned with structural properties of the degree structure generated by Solovay reducibility. We answer a natural question in this area of investigation by proving the density of the Solovay degrees. We also provide a new characterization of the random computably enumerable reals in terms of splittings in the Solovay degrees. Specifically, we show that the Solovay degrees of computably enumerable reals are dense, that any incomplete Solovay degree splits over any lesser degree, and that the join of any two incomplete Solovay degrees is incomplete, so that the complete Solovay degree does not split at all. The methodology is of some technical interest, since it includes a priority argument in which the injuries are themselves controlled by randomness considerations.

Key words. randomness, computably enumerable reals, algorithmic information theory, Kolmogorov complexity, Solovay reducibility

AMS subject classifications. 68Q30, 03D30, 03D80

PII. S0097539700376937

1. Introduction.

In this paper we are concerned with effectively generated reals in the interval $(0, 1]$ and their relative randomness. In what follows, *real* and *rational* will mean positive real and positive rational, respectively. It will be convenient to work modulo 1, that is, identifying $n + \alpha$ and α for any $n \in \omega$ and $\alpha \in (0, 1]$, and we do this below without further comment.

Our basic objects are reals that are limits of computable increasing sequences of rationals. We call such reals *computably enumerable* (c.e.), but they have also been called *recursively enumerable*, *left computable* (by Ambos-Spies, Weihrauch, and Zheng [1]), and *left semicomputable*. If, in addition to the existence of a computable increasing sequence $q_0, q_1, \ldots$ of rationals with limit α, there is a total computable function f such that $\alpha - q_{f(n)} < 2^{-n}$ for all $n \in \omega$, then α is called *computable*.

*Received by the editors August 21, 2000; accepted for publication (in revised form) August 2, 2001; published electronically April 12, 2002.

http://www.siam.org/journals/sicomp/31-4/37693.html

†School of Mathematical and Computing Sciences, Victoria University of Wellington, P.O. Box 600, Wellington, New Zealand (Rod.Downey@mcs.vuw.ac.nz). The research of this author was supported by the Marsden Fund for Basic Science and a US/NZ cooperative science grant.

‡School of Mathematical and Computing Sciences, Victoria University of Wellington. Current address: Department of Mathematics, The University of Chicago, 5734 S. University Ave., Chicago, IL 60637 (drh@math.uchicago.edu). The research of this author was supported by the Marsden Fund for Basic Science.

§Department of Mathematics, The University of Chicago, 5734 S. University Ave., Chicago, IL 60637 (nies@math.uchicago.edu). The research of this author was supported by a US/NZ cooperative science grant and NSF grant DMS-9803482.

These and related concepts have been widely studied. In addition to the papers and books mentioned elsewhere in this introduction, we may cite, among others, early work of Rice [26], Lachlan [21], Soare [28], and Ceĭtin [8] and more recent papers by Ko [17, 18], Calude, Coles, Hertling, and Khoussainov [5], Ho [16], and Downey and LaForte [14].

A computer M is *self-delimiting* if, for each binary string σ, $M(\sigma)\downarrow$ implies that $M(\sigma')\uparrow$ for all σ' properly extending σ. It is *universal* if for each self-delimiting computer N there is a constant c such that, for each binary string σ, if $N(\sigma)\downarrow$, then $M(\tau)\downarrow = N(\sigma)$ for some τ with $|\tau| \leqslant |\sigma| + c$.

Fix a self-delimiting universal computer M. We can define Chaitin's number $\Omega = \Omega_M$ via

$$\Omega = \sum_{M(\sigma)\downarrow} 2^{-|\sigma|}.$$

The properties of Ω relevant to this paper are independent of the choice of M. A c.e. real is an Ω-*number* if it is Ω_M for some self-delimiting universal computer M.

The c.e. real Ω is random in the canonical Martin-Löf sense. Recall that a *Martin-Löf test* is a uniformly c.e. sequence $\{V_e : e > 0\}$ of c.e. subsets of $\{0,1\}^*$ such that for all $e > 0$,

$$\mu(V_e\{0,1\}^\omega) \leqslant 2^{-e},$$

where μ denotes the usual product measure on $\{0,1\}^\omega$. The string $\sigma \in \{0,1\}^\omega$ and the real $0.\sigma$ are *random* or, more precisely, 1-random if $\sigma \notin \bigcap_{e>0} V_e\{0,1\}^\omega$ for every Martin-Löf test $\{V_e : e > 0\}$.

An alternate characterization of the random reals can be given via the notion of a *Solovay test*. We give a somewhat nonstandard definition of this notion, which will be useful below. A Solovay test is a c.e. multiset $\{I_i : i \in \omega\}$ of intervals with rational endpoints such that $\sum_{i\in\omega} |I_i| < \infty$, where $|I|$ is the length of the interval I. As Solovay [32] showed, a real α is random if and only if $\{i \in \omega : \alpha \in I_i\}$ is finite for every Solovay test $\{I_i : i \in \omega\}$.

Many authors have studied Ω and its properties, notably Chaitin [10, 11, 12] and Martin-Löf [25]. In the very long and widely circulated manuscript [32] (a fragment of which appeared in [33]), Solovay carefully investigated relationships between Martin-Löf-Chaitin prefix-free complexity, Kolmogorov complexity, and properties of random languages and reals. See Chaitin [10] for an account of some of the results in this manuscript.

Solovay discovered that several important properties of Ω (whose definition is model-dependent) are shared by another class of reals he called Ω-like, whose definition is model-independent. To define this class, he introduced the following reducibility relation among c.e. reals, called *domination* or *Solovay reducibility*.

DEFINITION 1.1. *Let α and β be c.e. reals. We say that α* dominates *β, and write $\beta \leqslant_s \alpha$, if there are a constant c and a partial computable function $\varphi : \mathbb{Q} \to \mathbb{Q}$ such that for each rational $q < \alpha$ we have $\varphi(q)\downarrow < \beta$ and*

$$\beta - \varphi(q) \leqslant c(\alpha - q).$$

We write $\beta <_s \alpha$ if $\beta \leqslant_s \alpha$ and $\alpha \not\leqslant_s \beta$, and we write $\alpha \equiv_s \beta$ if $\alpha \leqslant_s \beta$ and $\beta \leqslant_s \alpha$.

The notation $\leqslant_{\mathrm{dom}}$ has sometimes been used instead of $\leqslant_{\mathrm{s}}$.

The prefix-free complexity $H(\tau)$ of a binary string τ is the length of the shortest binary string σ such that $M(\sigma)\downarrow = \tau$, where M is a fixed self-delimiting universal

computer. (The choice of M does not affect the prefix-free complexity, up to a constant additive factor.) Most of the statements about $H(\tau)$ made below also hold for the standard Kolmogorov complexity $K(\tau)$. For more on the definitions and basic properties of $H(\tau)$ and $K(\tau)$, see Chaitin [12], Calude [3], and Li and Vitanyi [24]. Among the many works dealing with these and related topics, and in addition to those mentioned elsewhere in this paper, we may cite Solomonoff [30, 31], Kolmogorov [19], Levin [22, 23], Schnorr [27], and the expository article by Calude and Chaitin [4].

As shown by Schnorr (see Chaitin [9]), a real α is random if and only if there is a constant c such that $H(\alpha \upharpoonright n) > n - c$ for all $n \in \omega$. (We identify a real $\alpha \in (0, 1]$ with the infinite binary string σ such that $\alpha = 0.\sigma$. The fact that certain reals have two different dyadic expansions need not concern us here, since all such reals are rational.) Solovay reducibility is naturally associated with randomness because of the following fact, whose proof we sketch for completeness.

THEOREM 1.2 (Solovay [32]). *Let $\beta \leqslant_s \alpha$ be c.e. reals. There is a constant k such that $H(\beta \upharpoonright n) \leqslant H(\alpha \upharpoonright n) + k$ for all $n \in \omega$.*

Proof sketch. We first sketch the proof of the following lemma, implicit in [32] and noted by Calude, Hertling, Khoussainov, and Wang [6].

LEMMA 1.3. *Let $c \in \omega$. There is a constant k such that for all $n \geqslant 1$ and all binary strings σ, τ of length n with $|0.\sigma - 0.\tau| < c2^{-n}$, we have $|H(\tau) - H(\sigma)| \leqslant k$.*

The proof of the lemma is relatively simple. We can easily write a program P that, for each sufficiently long σ, generates the $2c+1$ binary strings τ' of length n with $|0.\sigma - 0.\tau'| < c2^{-n}$. For any binary strings σ, τ of length n with $|0.\sigma - 0.\tau| < c2^{-n}$, in order to compute τ it suffices to know a program for σ and the position of τ on the list generated by P on input σ.

Turning to the proof of the theorem, let φ and c be as in Definition 1.1. Let $\alpha_n = 0.(\alpha \upharpoonright n)$. Since α_n is rational and $\alpha - \alpha_n < 2^{-(n+1)}$, we have $\beta - \varphi(\alpha_n) < c2^{-(n+1)}$. Thus, by the lemma, there is a constant k such that $H(\beta \upharpoonright n) \leqslant H(\varphi(\alpha_n)) + k$ for all $n \geqslant 1$, which implies that $H(\beta \upharpoonright n) \leqslant H(\alpha \upharpoonright n) + k$. □

Solovay observed that Ω dominates all c.e. reals, and Theorem 1.2 implies that if a c.e. real dominates all c.e. reals then it must be random. This led Solovay to define a c.e. real to be Ω-*like* if it dominates all c.e. reals. The point is that the definition of Ω-like seems quite model-independent (in the sense that it does not require a choice of self-delimiting universal computer), as opposed to the model-dependent definition of Ω. However, Calude, Hertling, Khoussainov, and Wang [6] showed that the two notions coincide.

THEOREM 1.4 (Calude, Hertling, Khoussainov, and Wang [6]). *A c.e. real is Ω-like if and only if it is an Ω-number.*

This circle of ideas was completed recently by Kučera and Slaman [20], who proved the converse to the fact that Ω-like reals are random.

THEOREM 1.5 (Kučera and Slaman [20]). *A c.e. real is random if and only if it is Ω-like.*

It is natural to seek to understand the c.e. reals under Solovay reducibility. A useful characterization of this reducibility is given by the following lemma, which we prove in the next section.

LEMMA 1.6. *Let α and β be c.e. reals. The following are equivalent.*

1. $\beta \leqslant_s \alpha$.
2. *For some computable sequence of rationals $a_0, a_1, \ldots$ such that*

$$\alpha = \sum_{n \in \omega} a_n$$

there are a constant c and a computable sequence of rationals $\varepsilon_0, \varepsilon_1, \ldots < c$ such that

$$\beta = \sum_{n\in\omega} \varepsilon_n a_n.$$

3. *For* every *computable sequence of rationals $a_0, a_1, \ldots$ such that $\alpha = \sum_{n\in\omega} a_n$ there are a constant c and a computable sequence of rationals $\varepsilon_0, \varepsilon_1, \ldots < c$ such that $\beta = \sum_{n\in\omega} \varepsilon_n a_n$.*

Phrased another way, Lemma 1.6 says that the c.e. reals dominated by a given c.e. real α essentially correspond to splittings of α under arithmetic addition.

COROLLARY 1.7. *Let $\beta \leqslant_s \alpha$ be c.e. reals. There are a c.e. real γ and a rational c such that $c\alpha = \beta + \gamma$.*

Proof. Let $a_0, a_1, \ldots$ be a computable sequence of rationals such that $\alpha = \sum_{n\in\omega} a_n$. Let $c \in \mathbb{Q}$ and $\varepsilon_0, \varepsilon_1, \ldots < c$ be as in Lemma 1.6. Define $\gamma = \sum_{n\in\omega}(c - \varepsilon_n)a_n$. Since each ε_n is less than c, the real γ is c.e., and of course $\beta + \gamma = c\alpha$. □

Solovay reducibility has a number of other beautiful interactions with arithmetic, as we now discuss.

The relation $\leqslant_{\mathrm{S}}$ is reflexive and transitive, and hence $\equiv_{\mathrm{S}}$ is an equivalence relation on the c.e. reals. Thus we can define the *Solovay degree* $[\alpha]$ of a c.e. real α as its $\equiv_{\mathrm{S}}$ equivalence class. (When we mention Solovay degrees below, we always mean Solovay degrees of c.e. reals.) The Solovay degrees form an upper semilattice, with the join of $[\alpha]$ and $[\beta]$ being $[\alpha+\beta]$=$[\alpha\beta]$, a fact observed by Solovay and others, such as Calude, Hertling, Khoussainov, and Wang [6]. ($\oplus$ is definitely not a join operation here.) We note the following slight improvement of this result. Recall that an upper semilattice U is *distributive* if for all $a_0, a_1, b \in U$ with $b \leqslant a_0 \vee a_1$ there exist $b_0, b_1 \in U$ such that $b_0 \vee b_1 = b$ and $b_i \leqslant a_i$ for $i = 0, 1$.

LEMMA 1.8. *The Solovay degrees of c.e. reals form a distributive upper semilattice with $[\alpha] \vee [\beta] = [\alpha + \beta] = [\alpha\beta]$.*

Proof. Suppose that $\beta \leqslant_{\mathrm{S}} \alpha_0 + \alpha_1$. Let $a_0^0, a_1^0, \ldots$ and $a_0^1, a_1^1, \ldots$ be computable sequences of rationals such that $\alpha_i = \sum_{n\in\omega} a_n^i$ for $i = 0, 1$. By Lemma 1.6, there are a constant c and a computable sequence of rationals $\varepsilon_0, \varepsilon_1, \ldots < c$ such that $\beta = \sum_{n\in\omega} \varepsilon_n(a_n^0 + a_n^1)$. Let $\beta_i = \sum_{n\in\omega} \varepsilon_n a_n^i$. Then $\beta = \beta_0 + \beta_1$ and, again by Lemma 1.6, $\beta_i \leqslant_{\mathrm{S}} \alpha_i$ for $i = 0, 1$. This establishes distributivity.

To see that the join in the Solovay degrees is given by addition, we again apply Lemma 1.6. Certainly, for any c.e. reals β_0 and β_1 we have $\beta_i \leqslant_{\mathrm{S}} \beta_0 + \beta_1$ for $i = 0, 1$, and hence $[\beta_0 + \beta_1] \geqslant_{\mathrm{S}} [\beta_0], [\beta_1]$. Conversely, suppose that $\beta_0, \beta_1 \leqslant \alpha$. Let $a_0, a_1, \ldots$ be a computable sequence of rationals such that $\alpha = \sum_{n\in\omega} a_n$. For each $i = 0, 1$ there are a constant c_i and a computable sequence of rationals $\varepsilon_0^i, \varepsilon_1^i, \ldots < c_i$ such that $\beta_i = \sum_{n\in\omega} \varepsilon_n^i a_n$. Thus $\beta_0 + \beta_1 = \sum_{n\in\omega}(\varepsilon_n^0 + \varepsilon_n^1)a_n$. Since each $\varepsilon_n^0 + \varepsilon_n^1$ is less than $c_0 + c_1$, a final application of Lemma 1.6 shows that $\beta_0 + \beta_1 \leqslant_{\mathrm{S}} \alpha$.

The proof that the join in the Solovay degrees is also given by multiplication is a similar application of Lemma 1.6. □

There is a least Solovay degree, the degree of the computable reals, as well as a greatest one, the degree of Ω. For proofs of these facts and more on c.e. reals and Solovay reducibility, see, for instance, Chaitin [10, 11, 12], Calude, Hertling, Khoussainov, and Wang [6], Calude and Nies [7], Calude [2], Kučera and Slaman [20], and Downey, Hirschfeldt, and LaForte [15].

Despite the many attractive features of the Solovay degrees, their structure is largely unknown. Downey, Hirschfeldt, and LaForte (to be submitted) have shown that this structure is very complicated by proving that it has an undecidable first-order theory.

One question addressed in the present paper is whether the structure of the Solovay degrees is dense. Indeed, up to now, it was not known even whether there is a *minimal* Solovay degree. That is, intuitively, if a c.e. real α is not computable, must there be a c.e. real that is also not computable, yet is strictly less random than α? In the process of understanding a degree structure, the question of density has always played a key role and has been one of the first to be addressed. For instance, the Sacks Density Theorem (see [29]) was one of the earliest and most important results in the study of the c.e. Turing degrees.

In this paper, we show that the Solovay degrees of c.e. reals are dense. To do this we divide the proof into two parts. We prove that if $\alpha <_{\mathrm{S}} \Omega$, then there is a c.e. real γ with $\alpha <_{\mathrm{S}} \gamma <_{\mathrm{S}} \Omega$, and we also prove that every *incomplete* Solovay degree splits over each lesser degree.

The nonuniform nature of the argument is essential given the techniques we use, since, in the splitting case, we have a priority construction in which the control of the injuries is directly tied to the enumeration of Ω. The fact that if a c.e. real α is Solovay-incomplete, then Ω must grow more slowly than α is what allows us to succeed. (We will discuss this more fully in section 3.) This unusual technique is of some technical interest and clearly cannot be applied to proving upwards density, since in that case the top degree is the degree of Ω itself. To prove upwards density, we use a different technique, taking advantage of the fact that, however we construct a c.e. real, it is automatically dominated by Ω.

In light of these results, and further motivated by the general question of how randomness can be produced, it is natural to ask whether the complete Solovay degree can be split, or in other words, whether there exist nonrandom c.e. reals α and β such that $\alpha + \beta$ is random. We give a negative answer to this question, thus characterizing the random c.e. reals as those c.e. reals that cannot be written as the sum of two c.e. reals of lesser Solovay degrees.

We remark that there are (non-c.e.) nonrandom reals whose sum is random; the following is an example of this phenomenon. Define the real α by letting $\alpha(n) = 0$ if n is even and $\alpha(n) = \Omega(n)$ otherwise. (Here we identify a real with its dyadic expansion as above.) Define the real β by letting $\beta(n) = 0$ if n is odd and $\beta(n) = \Omega(n)$ otherwise. Now α and β are clearly nonrandom, but $\alpha + \beta = \Omega$ is random.

Before turning to the details of the paper, we point out that there are other reducibilities one can study in this context. Downey [13] and Downey, Hirschfeldt, and LaForte [15] introduced two such reducibilities, sw-reducibility and rH-reducibility, and showed, among other things, that the results of this paper also hold for rH-reducibility. The proofs are essentially the same as those in this paper. Ultimately, the basic reducibility we seek to understand is H-reducibility, where $\sigma \leqslant_H \tau$ if $H(\sigma \upharpoonright n) \leqslant H(\tau \upharpoonright n) + O(1)$. Not much is known about this directly, but it is again possible to adapt the methods of this paper to prove the analogous results for H-reducibility.

2. Preliminaries. The following lemma, implicit in [32] and proved in [15], provides an alternate characterization of Solovay reducibility, which is the one that we will use below.

LEMMA 2.1. *Let α and β be c.e. reals, and let $\alpha_0, \alpha_1, \ldots$ and $\beta_0, \beta_1, \ldots$ be computable increasing sequences of rationals converging to α and β, respectively. Then $\beta \leqslant_s \alpha$ if and only if there are a constant d and a total computable function f such that for all $n \in \omega$,*

$$\beta - \beta_{f(n)} < d(\alpha - \alpha_n).$$

Whenever we mention a c.e. real α, we assume that we have chosen a computable increasing sequence $\alpha_0, \alpha_1, \ldots$ converging to α. The previous lemma guarantees that, in determining whether one c.e. real dominates another, the particular choice of such sequences is irrelevant. For convenience of notation, we adopt the convention that, for any c.e. real α mentioned below, the expression $\alpha_s - \alpha_{s-1}$ is equal to α_0 when $s = 0$.

We will also make use of two more lemmas, the first of which has Lemma 1.6 as a corollary.

LEMMA 2.2. *Let $\beta \leqslant_s \alpha$ be c.e. reals and let $\alpha_0, \alpha_1, \ldots$ be a computable increasing sequence of rationals converging to α. There is a computable increasing sequence $\hat{\beta}_0, \hat{\beta}_1, \ldots$ of rationals converging to β such that for some constant c and all $s \in \omega$,*

$$\hat{\beta}_s - \hat{\beta}_{s-1} < c(\alpha_s - \alpha_{s-1}).$$

Proof. Fix a computable increasing sequence $\beta_0, \beta_1, \ldots$ of rationals converging to β, let d and f be as in Lemma 2.1, and let $c > d$ be such that $\beta_{f(0)} < c\alpha_0$. We may assume without loss of generality that f is increasing. Define $\hat{\beta}_0 = \beta_{f(0)}$.

There must be an $s_0 > 0$ for which $\beta_{f(s_0)} - \beta_{f(0)} < d(\alpha_{s_0} - \alpha_0)$, since otherwise we would have

$$\beta - \beta_{f(0)} = \lim_s \beta_{f(s)} - \beta_{f(0)} \geqslant \lim_s d(\alpha_s - \alpha_0) = d(\alpha - \alpha_0),$$

contradicting our choice of d and f. It is now easy to define $\hat{\beta}_1, \ldots, \hat{\beta}_{s_0}$ so that $\hat{\beta}_0 < \cdots < \hat{\beta}_{s_0} = \beta_{f(s_0)}$ and $\hat{\beta}_s - \hat{\beta}_{s-1} \leqslant d(\alpha_s - \alpha_{s-1}) < c(\alpha_s - \alpha_{s-1})$ for all $s \leqslant s_0$. For example, if we let μ be the minimum value of $d(\alpha_s - \alpha_{s-1})$ for $s \leqslant s_0$ and let t be least such that $\hat{\beta}_0 + d(\alpha_t - \alpha_0) < \beta_{f(s_0)} - 2^{-t}\mu$, then we can define

$$\hat{\beta}_{s+1} = \begin{cases} \hat{\beta}_s + d(\alpha_{s+1} - \alpha_s) & \text{if } s+1 < t, \\ \beta_{f(s_0)} - 2^{-(s+1)}\mu & \text{if } t \leqslant s+1 < s_0, \\ \beta_{f(s_0)} & \text{if } s+1 = s_0. \end{cases}$$

We can repeat the procedure in the previous paragraph with s_0 in place of 0 to obtain an $s_1 > s_0$ and $\hat{\beta}_{s_0+1}, \ldots, \hat{\beta}_{s_1}$ such that $\hat{\beta}_{s_0} < \cdots < \hat{\beta}_{s_1} = \beta_{f(s_1)}$ and $\hat{\beta}_s - \hat{\beta}_{s-1} < c(\alpha_s - \alpha_{s-1})$ for all $s_0 < s \leqslant s_1$.

Proceeding by recursion in this way, we define a computable increasing sequence $\hat{\beta}_0, \hat{\beta}_1, \ldots$ of rationals with the desired properties. □

We are now in a position to prove Lemma 1.6.

LEMMA 1.6. *Let α and β be c.e. reals. The following are equivalent.*

1. *$\beta \leqslant_s \alpha$.*
2. *For some computable sequence of rationals $a_0, a_1, \ldots$ such that*

$$\alpha = \sum_{n \in \omega} a_n,$$

there are a constant c and a computable sequence of rationals $\varepsilon_0, \varepsilon_1, \ldots < c$ such that

$$\beta = \sum_{n \in \omega} \varepsilon_n a_n.$$

3. *For* every *computable sequence of rationals $a_0, a_1, \ldots$ such that $\alpha = \sum_{n \in \omega} a_n$ there are a constant c and a computable sequence of rationals $\varepsilon_0, \varepsilon_1, \ldots < c$ such that $\beta = \sum_{n \in \omega} \varepsilon_n a_n$.*

Proof. It is easy to see that $3 \Rightarrow 2 \Rightarrow 1$. We prove that $1 \Rightarrow 3$.

Suppose that $\beta \leqslant_{\mathrm{S}} \alpha$. Given a computable sequence of rationals $a_0, a_1, \ldots$ such that $\alpha = \sum_{n\in\omega} a_n$, let $\alpha_n = \sum_{i\leqslant n} a_i$ and apply Lemma 2.2 to obtain c and $\hat{\beta}_0, \hat{\beta}_1, \ldots$ as in that lemma. Define $\varepsilon_n = (\hat{\beta}_n - \hat{\beta}_{n-1})a_n^{-1}$. Now $\sum_{n\in\omega} \varepsilon_n a_n = \sum_{n\in\omega} \hat{\beta}_n - \hat{\beta}_{n-1} = \beta$, and for all $n \in \omega$,

$$\varepsilon_n = (\hat{\beta}_n - \hat{\beta}_{n-1})a_n^{-1} = (\hat{\beta}_n - \hat{\beta}_{n-1})(\alpha_n - \alpha_{n-1})^{-1} < c. \qquad \square$$

We finish this section with a simple lemma which will be quite useful below.

LEMMA 2.3. *Let $\alpha \not\leqslant_s \beta$ be c.e. reals. The following hold for all total computable functions f and all $k \in \omega$.*

1. *For each $n \in \omega$ there is an $s \in \omega$ such that either*
   (i) $\alpha_t - \alpha_{f(n)} < k(\beta_t - \beta_n)$ *for all $t > s$, or*
   (ii) $\alpha_t - \alpha_{f(n)} > k(\beta_t - \beta_n)$ *for all $t > s$.*
2. *There are infinitely many $n \in \omega$ for which there is an $s \in \omega$ such that $\alpha_t - \alpha_{f(n)} > k(\beta_t - \beta_n)$ for all $t > s$.*

Proof. If there are infinitely many $t \in \omega$ such that $\alpha_t - \alpha_{f(n)} \leqslant k(\beta_t - \beta_n)$ and infinitely many $t \in \omega$ such that $\alpha_t - \alpha_{f(n)} \geqslant k(\beta_t - \beta_n)$, then

$$\alpha - \alpha_{f(n)} = \lim_t \alpha_t - \alpha_{f(n)} = \lim_t k(\beta_t - \beta_n) = k(\beta - \beta_n),$$

which implies that $\alpha \equiv_{\mathrm{S}} \beta$.

If there are infinitely many $t \in \omega$ such that $\alpha_t - \alpha_{f(n)} \leqslant k(\beta_t - \beta_n)$, then

$$\alpha - \alpha_{f(n)} = \lim_t \alpha_t - \alpha_{f(n)} \leqslant \lim_t k(\beta_t - \beta_n) = k(\beta - \beta_n).$$

So if this happens for all but finitely many n, then $\alpha \leqslant_{\mathrm{S}} \beta$. (The finitely many n for which $\alpha - \alpha_{f(n)} > k(\beta - \beta_n)$ can be brought into line by increasing the constant k.) $\square$

3. Main results. We now proceed with the proofs of our main results. We begin by showing that every incomplete Solovay degree can be split over any lesser Solovay degree.

THEOREM 3.1. *Let $\gamma <_s \alpha <_s \Omega$ be c.e. reals. There are c.e. reals β^0 and β^1 such that $\gamma <_s \beta^i <_s \alpha$ for $i = 0, 1$ and $\beta^0 + \beta^1 = \alpha$.*

Proof. We want to build β^0 and β^1 so that $\gamma \leqslant_{\mathrm{S}} \beta^i \leqslant_{\mathrm{S}} \alpha$ for $i = 0, 1$, $\beta^0 + \beta^1 = \alpha$, and the following requirement is satisfied for each $e, k \in \omega$ and $i < 2$:

$$R_{i,e,k} : \Phi_e \text{ total } \Rightarrow \exists n(\alpha - \alpha_{\Phi_e(n)} \geqslant k(\beta^i - \beta^i_n)).$$

By Lemma 2.2 and the fact that $\gamma/c \equiv_{\mathrm{S}} \gamma$ for any rational c, we may assume without loss of generality that $2(\gamma_s - \gamma_{s-1}) \leqslant \alpha_s - \alpha_{s-1}$ for each $s \in \omega$. (Recall our convention that $\mu_0 - \mu_{-1} = \mu_0$ for any c.e. real μ.)

In the absence of requirements of the form $R_{1-i,e,k}$, it is easy to satisfy simultaneously all requirements of the form $R_{i,e,k}$: for each $s \in \omega$, simply let $\beta^i_s = \gamma_s$ and $\beta^{1-i}_s = \alpha_s - \gamma_s$. In the presence of requirements of the form $R_{1-i,e,k}$, however, we cannot afford to be quite so cavalier in our treatment of β^{1-i}; enough of α has to be kept out of β^{1-i} to guarantee that β^{1-i} does not dominate α.

Most of the essential features of our construction are already present in the case of two requirements $R_{i,e,k}$ and $R_{1-i,e',k'}$, which we now discuss. We assume that $R_{i,e,k}$ has priority over $R_{1-i,e',k'}$ and that both Φ_e and $\Phi_{e'}$ are total. We will think of the β^j as being built by adding amounts to them in stages. Thus β^j_s will be the

total amount added to β^j by the end of stage s. At each stage s we begin by adding $\gamma_s - \gamma_{s-1}$ to the current value of each β^j; in the limit, this ensures that $\beta^j \geqslant_{\mathrm{s}} \gamma$.

We will say that $R_{i,e,k}$ is satisfied through n at stage s if $\Phi_e(n)[s]\downarrow$ and $\alpha_s - \alpha_{\Phi_e(n)} > k(\beta^i_s - \beta^i_n)$. The strategy for $R_{i,e,k}$ is to act whenever either it is not currently satisfied or the least number through which it is satisfied changes. Whenever this happens, $R_{i,e,k}$ initializes $R_{1-i,e',k'}$, which means that the amount of $\alpha - 2\gamma$ that $R_{1-i,e',k'}$ is allowed to funnel into β^i is reduced. More specifically, once $R_{1-i,e',k'}$ has been initialized for the mth time, the total amount that it is thenceforth allowed to put into β^i is reduced to 2^{-m}.

The above strategy guarantees that if $R_{1-i,e',k'}$ is initialized infinitely often, then the amount put into β^i by $R_{1-i,e',k'}$ (which in this case is all that is put into β^i except for the coding of γ) adds up to a computable real. In other words, $\beta^i \equiv_{\mathrm{s}} \gamma <_{\mathrm{s}} \alpha$. But it is not hard to argue, with the help of Lemma 2.3, that this means that there is a stage s after which $R_{i,e,k}$ is always satisfied and the least number through which it is satisfied does not change. So we conclude that $R_{1-i,e',k'}$ is initialized only finitely often, and that $R_{i,e,k}$ is eventually permanently satisfied.

This leaves us with the problem of designing a strategy for $R_{1-i,e',k'}$ that respects the strategy for $R_{i,e,k}$. The problem is one of timing. To simplify notation, let $\hat{\alpha} = \alpha - 2\gamma$ and $\hat{\alpha}_s = \alpha_s - 2\gamma_s$. Since $R_{1-i,e',k'}$ is initialized only finitely often, there is a certain amount 2^{-m} that it is allowed to put into β^i after the last time it is initialized. Thus if $R_{1-i,e',k'}$ waits until a stage s such that $\hat{\alpha} - \hat{\alpha}_s < 2^{-m}$, adding nothing to β^i until such a stage is reached, then from that point on it can put all of $\hat{\alpha} - \hat{\alpha}_s$ into β^i, which of course guarantees its success. The problem is that, in the general construction, a strategy working with a quota 2^{-m} cannot effectively find an s such that $\hat{\alpha} - \hat{\alpha}_s < 2^{-m}$. If it uses up its quota too soon, it may find itself unsatisfied and unable to do anything about it.

The key to solving this problem (and the reason for the hypothesis that $\alpha <_{\mathrm{s}} \Omega$) is the observation that, since the sequence $\Omega_0, \Omega_1, \ldots$ converges much more slowly than the sequence $\hat{\alpha}_0, \hat{\alpha}_1, \ldots, \Omega$ can be used to modulate the amount that $R_{1-i,e',k'}$ puts into β^i. More specifically, at a stage s, if $R_{1-i,e',k'}$'s current quota is 2^{-m}, then it puts into β^i as much of $\hat{\alpha}_s - \hat{\alpha}_{s-1}$ as possible, subject to the constraint that the total amount put into β^i by $R_{1-i,e',k'}$ since the last stage before stage s at which $R_{1-i,e',k'}$ was initialized must not exceed $2^{-m}\Omega_s$. As we will see below, the fact that $\Omega >_{\mathrm{s}} \alpha$ implies that there is a stage v after which $R_{1-i,e',k'}$ is allowed to put in all of $\hat{\alpha} - \hat{\alpha}_v$ into β^i.

In general, at a given stage s there will be several requirements, each with a certain amount that it wants (and is allowed) to direct into one of the β^j. We will work backwards, starting with the weakest priority requirement that we are currently considering. This requirement will be allowed to direct as much of $\hat{\alpha}_s - \hat{\alpha}_{s-1}$ as it wants (subject to its current quota, of course). If any of $\hat{\alpha}_s - \hat{\alpha}_{s-1}$ is left then the next weakest priority strategy will be allowed to act, and so on up the line.

We now proceed with the full construction. We say that $R_{i,e,k}$ has stronger priority than $R_{i',e',k'}$ if $2\langle e,k\rangle + i < 2\langle e',k'\rangle + i'$.

We say that $R_{i,e,k}$ is satisfied through n at stage s if

$$\Phi_e(n)[s]\downarrow \wedge\, \alpha_s - \alpha_{\Phi_e(n)} > k(\beta^i_s - \beta^i_n).$$

Let $n^{i,e,k}_s$ be the least n through which $R_{i,e,k}$ is satisfied at stage s, if such an n exists, and let $n^{i,e,k}_s = \infty$ otherwise.

A stage s is e-expansionary if

$$\max\{n \mid \forall m \leqslant n(\Phi_e(m)[s]\downarrow)\} > \max\{n \mid \forall m \leqslant n(\Phi_e(n)[s-1]\downarrow)\}.$$

Let q be the last e-expansionary stage before stage s (or let $q = 0$ if there have been none). We say that $R_{i,e,k}$ requires attention at stage s if s is an e-expansionary stage and there is an $r \in [q, s)$ such that either $n_r^{i,e,k} = \infty$ or $n_r^{i,e,k} \neq n_{r-1}^{i,e,k}$.

If $R_{i,e,k}$ requires attention at stage s, then we say that each requirement of weaker priority than $R_{i,e,k}$ is initialized at stage s.

Each requirement $R_{i,e,k}$ has associated with it a c.e. real $\tau^{i,e,k}$, which records the amount put into β^{1-i} for the sake of $R_{i,e,k}$.

We decide how to distribute $\delta = \alpha_s - \alpha_{s-1}$ between β^0 and β^1 at stage s as follows.

1. Let $j = s$ and $\varepsilon = 2(\gamma_s - \gamma_{s-1})$, and add $\gamma_s - \gamma_{s-1}$ to the current value of each β^i.

2. Let $i < 2$ and $e, k \in \omega$ be such that $2\langle e, k\rangle + i = j$. Let m be the number of times $R_{i,e,k}$ has been initialized and let t be the last stage at which $R_{i,e,k}$ was initialized. Let

$$\zeta = \min(\delta - \varepsilon, 2^{-(j+m)}\Omega_s - (\tau_{s-1}^{i,e,k} - \tau_t^{i,e,k})).$$

(It is not hard to check that ζ is nonnegative.) Add ζ to ε and to the current values of $\tau^{i,e,k}$ and β^{1-i}.

3. If $\varepsilon = \delta$ or $j = 0$, then add $\delta - \varepsilon$ to the current value of β^0 and end the stage. Otherwise, decrease j by one and go to step 2.

This completes the construction. Clearly, $\gamma \leqslant_{\mathrm{S}} \beta^i \leqslant_{\mathrm{S}} \alpha$ for $i = 0, 1$ and $\beta^0 + \beta^1 = \alpha$.

We now show by induction that each requirement initializes requirements of weaker priority only finitely often and is eventually satisfied. Assume by induction that $R_{i,e,k}$ is initialized only finitely often. Let $j = 2\langle e, k\rangle + i$, let m be the number of times $R_{i,e,k}$ is initialized, and let t be the last stage at which $R_{i,e,k}$ is initialized. If Φ_e is not total then $R_{i,e,k}$ is vacuously satisfied and eventually stops initializing requirements of weaker priority, so we may assume that Φ_e is total. It suffices to prove the following statements, which are clearly equivalent and imply that $R_{i,e,k}$ is satisfied:

1. $\lim_s n_s^{i,e,k}$ exists and is finite;
2. $R_{i,e,k}$ eventually stops requiring attention.

Assume for a contradiction that $R_{i,e,k}$ requires attention infinitely often. Since $\Omega \not\leqslant_{\mathrm{S}} \alpha$, part 2 of Lemma 2.3 implies that there are $v > u > t$ such that for all $w > v$ we have $2^{-(j+m)}(\Omega_w - \Omega_u) > \alpha_w - \alpha_u$. Furthermore, by the way the amount ζ added to $\tau^{i,e,k}$ at a given stage is defined in step 2 of the construction, $\tau_u^{i,e,k} - \tau_t^{i,e,k} \leqslant 2^{-(j+m)}\Omega_u$ and $\tau_{w-1}^{i,e,k} - \tau_u^{i,e,k} \leqslant \alpha_{w-1} - \alpha_u$. Thus for all $w > v$,

$$\begin{aligned}
&\alpha_w - \alpha_{w-1} = \alpha_w - \alpha_u - (\alpha_{w-1} - \alpha_u)\\
&< 2^{-(j+m)}(\Omega_w - \Omega_u) - (\alpha_{w-1} - \alpha_u) = 2^{-(j+m)}\Omega_w - (2^{-(j+m)}\Omega_u + \alpha_{w-1} - \alpha_u)\\
&\leqslant 2^{-(j+m)}\Omega_w - (\tau_u^{i,e,k} - \tau_t^{i,e,k} + \tau_{w-1}^{i,e,k} - \tau_u^{i,e,k}) = 2^{-(j+m)}\Omega_w - (\tau_{w-1}^{i,e,k} - \tau_t^{i,e,k}).
\end{aligned}$$

From this we conclude that, after stage v, the reverse recursion performed at each stage never gets past j, and hence everything put into β^i after stage v is put in either to code γ or for the sake of requirements of weaker priority than $R_{i,e,k}$.

Let τ be the sum of all $\tau^{1-i,e',k'}$ such that $R_{1-i,e',k'}$ has weaker priority than $R_{i,e,k}$. Let $s_l > t$ be the lth stage at which $R_{i,e,k}$ requires attention. If $R_{1-i,e',k'}$ is the $(p+1)$st requirement on the priority list and $p > j$, then $\tau^{1-i,e',k'} - \tau^{1-i,e',k'}_{s_l} \leqslant 2^{-(p+l)}\Omega$. Thus

$$\tau - \tau_{s_l} \leqslant \sum_{p\in\omega} 2^{-(p+l)}\Omega = 2^{-l}\Omega \leqslant 2^{-l},$$

and hence τ is computable.

Putting together the results of the previous two paragraphs, we see that $\beta^i \leqslant_{\mathrm{S}} \gamma$. Since $\alpha \not\leqslant_{\mathrm{S}} \gamma$, this means that $\alpha \not\leqslant_{\mathrm{S}} \beta^i$. It now follows from Lemma 2.3 that there is an $n \in \omega$ such that $R_{i,e,k}$ is eventually permanently satisfied through n, and such that $R_{i,e,k}$ is eventually never satisfied through any $n' < n$. Thus $\lim_s n^{i,e,k}_s$ exists and is finite, and hence $R_{i,e,k}$ is satisfied and eventually stops requiring attention. □

We now show that the Solovay degrees are upwards dense, which together with the previous result implies that they are dense.

THEOREM 3.2. *Let $\gamma <_S \Omega$ be a c.e. real. There is a c.e. real β such that $\gamma <_S \beta <_S \Omega$.*

Proof. We want to build $\beta \geqslant_{\mathrm{S}} \gamma$ to satisfy the following requirements for each $e, k \in \omega$:

$$R_{e,k} : \Phi_e \text{ total } \Rightarrow \exists n(\beta - \beta_{\Phi_e(n)} \geqslant k(\gamma - \gamma_n))$$

and

$$S_{e,k} : \Phi_e \text{ total } \Rightarrow \exists n(\Omega - \Omega_{\Phi_e(n)} \geqslant k(\beta - \beta_n)).$$

As in the previous proof, the analysis of an appropriate two-strategy case will be enough to outline the essentials of the full construction. Let us consider the strategies $S_{e,k}$ and $R_{e',k'}$, the former having priority over the latter. We assume that both Φ_e and $\Phi_{e'}$ are total.

The strategy for $S_{e,k}$ is basically to make β look like γ. At each point of the construction, $R_{e',k'}$ has a certain fraction of Ω that it is allowed to put into β. (This is in addition to the coding of γ into β, of course.) We will say that $S_{e,k}$ is satisfied through n at stage s if $\Phi_e(n)[s]\downarrow$ and $\Omega_s - \Omega_{\Phi_e(n)} > k(\beta_s - \beta_n)$. Whenever either it is not currently satisfied or the least number through which it is satisfied changes, $S_{e,k}$ initializes $R_{e',k'}$, which means that the fraction of Ω that $R_{e',k'}$ is allowed to put into β is reduced.

As in the previous proof, if $S_{e,k}$ is not eventually permanently satisfied through some n, then the amount put into β by $R_{e',k'}$ is computable, and hence $\beta \equiv_{\mathrm{S}} \gamma$. But, as before, this implies that there is a stage after which $S_{e,k}$ is permanently satisfied through some n and never again satisfied through any $n' < n$. Once this stage has been reached, $R_{e',k'}$ is free to code a fixed fraction of Ω into β, and hence it too succeeds.

We now proceed with the full construction. We say that a requirement $X_{e,k}$ has stronger priority than a requirement $Y_{e',k'}$ if either $\langle e,k\rangle < \langle e',k'\rangle$ or $\langle e,k\rangle = \langle e',k'\rangle$, $X = R$, and $Y = S$.

We say that $R_{e,k}$ is satisfied through n at stage s if $\Phi_e(n)[s]\downarrow$ and

$$\beta_s - \beta_{\Phi_e(n)} > k(\gamma_s - \gamma_n).$$

We say that $S_{e,k}$ is satisfied through n at stage s if $\Phi_e(n)[s]\downarrow$ and

$$\Omega_s - \Omega_{\Phi_e(n)} > k(\beta_s - \beta_n).$$

For a requirement $X_{e,k}$, let $n_s^{X_{e,k}}$ be the least n through which $X_{e,k}$ is satisfied at stage s, if such an n exists, and let $n_s^{X_{e,k}} = \infty$ otherwise.

As before, a stage s is e-expansionary if

$$\max\{n \mid \forall m \leqslant n(\Phi_e(m)[s]\downarrow)\} > \max\{n \mid \forall m \leqslant n(\Phi_e(n)[s-1]\downarrow)\}.$$

Let $X_{e,k}$ be a requirement and let q be the last e-expansionary stage before stage s (or let $q = 0$ if there have been none). We say that $X_{e,k}$ requires attention at stage s if s is an e-expansionary stage and there is an $r \in [q, s)$ such that either $n_r^{X_{e,k}} = \infty$ or $n_r^{X_{e,k}} \neq n_{r-1}^{X_{e,k}}$.

At stage s, proceed as follows. First add $\gamma_s - \gamma_{s-1}$ to the current value of β. If no requirement requires attention at stage s, then end the stage. Otherwise, let $X_{e,k}$ be the strongest priority requirement requiring attention at stage s. We say that $X_{e,k}$ acts at stage s. If $X = S$, then initialize all weaker priority requirements and end the stage. If $X = R$, then let $j = \langle e, k\rangle$ and let m be the number of times that $R_{e,k}$ has been initialized. If s is the first stage at which $R_{e,k}$ acts after the last time it was initialized, then let t be the last stage at which $R_{e,k}$ was initialized (or let $t = 0$ if $R_{e,k}$ has never been initialized), and otherwise let t be the last stage at which $R_{e,k}$ acted. Add $2^{-(j+m)}(\Omega_s - \Omega_t)$ to the current value of β and end the stage.

This completes the construction. Since β is bounded by $\gamma + \sum_{i\geqslant 0} 2^{-i}\Omega = \gamma + 2\Omega$, it is a well-defined c.e. real. Furthermore, $\gamma \leqslant_{\mathrm{S}} \beta$.

We now show by induction that each requirement initializes requirements of weaker priority only finitely often and is eventually satisfied. Assume by induction that there is a stage u such that no requirement of stronger priority than $X_{e,k}$ requires attention after stage u. If Φ_e is not total, then $X_{e,k}$ is vacuously satisfied and eventually stops requiring attention, so we may assume that Φ_e is total. It suffices to prove the following statements, which are clearly equivalent and imply that $X_{e,k}$ is satisfied:

1. $\lim_s n_s^{X_{e,k}}$ exists and is finite;
2. $X_{e,k}$ eventually stops requiring attention;
3. $X_{e,k}$ acts only finitely often.

First suppose that $X = R$. Let $j = \langle e, k\rangle$ and let m be the number of times that $R_{e,k}$ is initialized. (Since $R_{e,k}$ is not initialized at any stage after stage u, this number is finite.) Suppose that $R_{e,k}$ acts infinitely often. Then the total amount added to β for the sake of $R_{e,k}$ after any stage $v > u$ is greater than or equal to $2^{-(j+m)}(\Omega - \Omega_v)$, and hence $\beta \equiv_{\mathrm{S}} 2^{-(j+m)}\Omega \equiv_{\mathrm{S}} \Omega \not\leqslant_{\mathrm{S}} \gamma$. It now follows from Lemma 2.3 that there is an $n \in \omega$ such that $R_{e,k}$ is eventually permanently satisfied through n, and such that $R_{e,k}$ is eventually never satisfied through $n' < n$. Thus $\lim_s n_s^{R_{e,k}}$ exists and is finite, and hence $R_{e,k}$ is satisfied and eventually stops requiring attention.

Now suppose that $X = S$ and $S_{e,k}$ acts infinitely often. If $v > u$ is the mth stage at which $S_{e,k}$ acts then the total amount added to β after stage v for purposes other than coding γ is bounded by $\sum_{i\geqslant 0} 2^{-(i+m)}\Omega < 2^{-m+1}$. This means that $\beta \equiv_{\mathrm{S}} \gamma \not\geqslant_{\mathrm{S}} \Omega$. It now follows from Lemma 2.3 that there is an $n \in \omega$ such that $S_{e,k}$ is eventually permanently satisfied through n, and such that $S_{e,k}$ is eventually never satisfied through $n' < n$. Thus $\lim_s n_s^{S_{e,k}}$ exists and is finite, and hence $S_{e,k}$ is satisfied and eventually stops requiring attention. □

Combining Theorems 3.1 and 3.2, we have the following result.

THEOREM 3.3. *The Solovay degrees of c.e. reals are dense.*

We finish by showing that the hypothesis that $\alpha <_{\mathrm{S}} \Omega$ in the statement of Theorem 3.1 is necessary. This fact will follow easily from a stronger result which shows

that, despite the upwards density of the Solovay degrees, there is a sense in which the complete Solovay degree is very much above all other Solovay degrees. We begin with a lemma giving a sufficient condition for domination.

LEMMA 3.4. *Let α and β be c.e. reals and let $\alpha_0, \alpha_1, \dots$ and $\beta_0, \beta_1, \dots$ be computable increasing sequences of rationals converging to α and β, respectively. Let f be an increasing total computable function and let $k > 0$ be a natural number. If there are infinitely many $s \in \omega$ such that $k(\alpha - \alpha_s) > \beta - \beta_{f(s)}$, but only finitely many $s \in \omega$ such that $k(\alpha_t - \alpha_s) > \beta_{f(t)} - \beta_{f(s)}$ for all $t > s$, then $\beta \leqslant_s \alpha$.*

Proof. By taking $\beta_{f(0)}, \beta_{f(1)}, \dots$ instead of $\beta_0, \beta_1, \dots$ as an approximating sequence for β, we may assume that f is the identity.

By hypothesis, there is an $r \in \omega$ such that for all $s > r$ there is a $t > s$ with $k(\alpha_t - \alpha_s) \leqslant \beta_t - \beta_s$. Furthermore, there is an $s_0 > r$ such that $k(\alpha - \alpha_{s_0}) > \beta - \beta_{s_0}$. Given s_i, let s_{i+1} be the least number greater than s_i such that $k(\alpha_{s_{i+1}} - \alpha_{s_i}) \leqslant \beta_{s_{i+1}} - \beta_{s_i}$.

Assuming by induction that $k(\alpha - \alpha_{s_i}) > \beta - \beta_{s_i}$, we have

$$k(\alpha - \alpha_{s_{i+1}}) = k(\alpha - \alpha_{s_i}) - k(\alpha_{s_{i+1}} - \alpha_{s_i}) > \beta - \beta_{s_i} - (\beta_{s_{i+1}} - \beta_{s_i}) = \beta - \beta_{s_{i+1}}.$$

Thus $s_0 < s_1 < \cdots$ is a computable sequence such that $k(\alpha - \alpha_{s_i}) > \beta - \beta_{s_i}$ for all $i \in \omega$.

Now define the computable function g by letting $g(n)$ be the least s_i that is greater than or equal to n. Then $\beta - \beta_{g(n)} < k(\alpha - \alpha_{g(n)}) \leqslant k(\alpha - \alpha_n)$ for all $n \in \omega$, and hence $\beta \leqslant_{\mathrm{S}} \alpha$. □

THEOREM 3.5. *Let α and β be c.e. reals and let $\alpha_0, \alpha_1, \dots$ and $\beta_0, \beta_1, \dots$ be computable increasing sequences of rationals converging to α and β, respectively. Let f be an increasing total computable function and let $k > 0$ be a natural number. If β is random and there are infinitely many $s \in \omega$ such that $k(\alpha - \alpha_s) > \beta - \beta_{f(s)}$, then α is random.*

Proof. As in Lemma 3.4, we may assume that f is the identity. If α is rational then we can replace it with a nonrational computable real α' such that $\alpha' - \alpha'_s \geqslant \alpha - \alpha_s$ for all $s \in \omega$, so we may assume that α is not rational.

We assume that α is nonrandom and there are infinitely many $s \in \omega$ such that $k(\alpha - \alpha_s) > \beta - \beta_s$, and we show that β is nonrandom. The idea is to take a Solovay test $A = \{I_i : i \in \omega\}$ such that $\alpha \in I_i$ for infinitely many $i \in \omega$ and use it to build a Solovay test $B = \{J_i : i \in \omega\}$ such that $\beta \in J_i$ for infinitely many $i \in \omega$.

Let

$$U = \{s \in \omega \mid k(\alpha - \alpha_s) > \beta - \beta_s\}.$$

Except in the trivial case in which $\beta \equiv_{\mathrm{S}} \alpha$, Lemma 2.3 guarantees that U is Δ^0_2. Thus a first attempt at building B could be to run the following procedure for all $i \in \omega$ in parallel. Look for the least t such that there is an $s < t$ with $s \in U[t]$ and $\alpha_s \in I_i$. If there is more than one number s with this property then choose the least among such numbers. Begin to add the intervals

$$(*) \qquad [\beta_s, \beta_s + k(\alpha_{s+1} - \alpha_s)], [\beta_s + k(\alpha_{s+1} - \alpha_s), \beta_s + k(\alpha_{s+2} - \alpha_s)], \dots$$

to B, continuing to do so as long as s remains in U and the approximation of α remains in I_i. If the approximation of α leaves I_i, then end the procedure. If s leaves U, say at stage u, then repeat the procedure (only considering $t \geqslant u$, of course).

If $\alpha \in I_i$, then the variable s in the above procedure eventually assumes a value in U. For this value, $k(\alpha - \alpha_s) > \beta - \beta_s$, from which it follows that $k(\alpha_u - \alpha_s) > \beta - \beta_s$

for some $u > s$, and hence that $\beta \in [\beta_s, \beta_s + k(\alpha_u - \alpha_s)]$. So β must be in one of the intervals $(*)$ added to B by the above procedure.

Since α is in infinitely many of the I_i, running the above procedure for all $i \in \omega$ guarantees that β is in infinitely many of the intervals in B. The problem is that we also need the sum of the lengths of the intervals in B to be finite, and the above procedure gives no control over this sum, since it could easily be the case that we start working with some s, see it leave U at some stage t (at which point we have already added to B intervals whose lengths add up to $\alpha_{t-1} - \alpha_s$), and then find that the next s with which we have to work is much smaller than t. Since this could happen many times for each $i \in \omega$, we would have no bound on the sum of the lengths of the intervals in B.

This problem would be solved if we had an infinite computable subset T of U. For each I_i, we could look for an $s \in T$ such that $\alpha_s \in I_i$, and then begin to add the intervals $(*)$ to B, continuing to do so as long as the approximation of α remained in I_i. (Of course, in this easy setting, we could also simply add the single interval $[\beta_s, \beta_s + k\,|I_i|]$ to B.) It is not hard to check that this would guarantee that if $\alpha \in I_i$ then β is in one of the intervals added to B, while also ensuring that the sum of the lengths of these intervals is less than or equal to $k\,|I_i|$. Following this procedure for all $i \in \omega$ would give us the desired Solovay test B. Unless $\beta \leqslant_{\mathrm{S}} \alpha$, however, there is no infinite computable $T \subseteq U$, so we use Lemma 3.4 to obtain the next best thing.

Let

$$S = \{s \in \omega \mid \forall t > s(k(\alpha_t - \alpha_s) > \beta_t - \beta_s)\}.$$

If $\beta \leqslant_{\mathrm{S}} \alpha$ then β is nonrandom, so, by Lemma 3.4, we may assume that S is infinite. Note that $k(\alpha - \alpha_s) \geqslant \beta - \beta_s$ for all $s \in S$. In fact, we may assume that $k(\alpha - \alpha_s) > \beta - \beta_s$ for all $s \in S$, since if $k(\alpha - \alpha_s) = \beta - \beta_s$, then $k\alpha$ and β differ by a rational amount, and hence β is nonrandom.

The set S is co-c.e. by definition, but it has an additional useful property. Let

$$S[t] = \{s \in \omega \mid \forall u \in (s, t](k(\alpha_u - \alpha_s) > \beta_u - \beta_s)\}.$$

If $s \in S[t-1] - S[t]$, then no $u \in (s, t)$ is in S, since for any such u we have

$$k(\alpha_t - \alpha_u) = k(\alpha_t - \alpha_s) - k(\alpha_u - \alpha_s) \leqslant \beta_t - \beta_s - (\beta_u - \beta_s) = \beta_t - \beta_u.$$

In other words, if s leaves S at stage t then so do all numbers in (s, t).

To construct B, we run the following procedure P_i for all $i \in \omega$ in parallel. Note that B is a multiset, so we are allowed to add more than one copy of a given interval to B.

1. Look for an $s \in \omega$ such that $\alpha_s \in I_i$.
2. Let $t = s + 1$. If $\alpha_t \notin I_i$, then terminate the procedure.
3. If $s \notin S[t]$, then let $s = t$ and go to step 2. Otherwise, add the interval

$$[\beta_s + k(\alpha_{t-1} - \alpha_s), \beta_s + k(\alpha_t - \alpha_s)]$$

to B, increase t by one, and repeat step 3.

This concludes the construction of B. We now show that the sum of the lengths of the intervals in B is finite and that β is in infinitely many of the intervals in B.

For each $i \in \omega$, let B_i be the set of intervals added to B by P_i and let l_i be the sum of the lengths of the intervals in B_i. If P_i never leaves step 1, then $B_i = \emptyset$. If

P_i eventually terminates then $l_i \leqslant k(\alpha_t - \alpha_s)$ for some $s, t \in \omega$ such that $\alpha_s, \alpha_t \in I_i$, and hence $l_i \leqslant k\,|I_i|$. If P_i reaches step 3 and never terminates, then $\alpha \in I_i$ and $l_i \leqslant k(\alpha - \alpha_s)$ for some $s \in \omega$ such that $\alpha_s \in I_i$, and hence again $l_i \leqslant k\,|I_i|$. Thus the sum of the lengths of the intervals in B is less than or equal to $k \sum_{i \in \omega} |I_i| < \infty$.

To show that β is in infinitely many of the intervals in B, it is enough to show that, for each $i \in \omega$, if $\alpha \in I_i$ then β is in one of the intervals in B_i.

Fix $i \in \omega$ such that $\alpha \in I_i$. Since α is not rational, $\alpha_u \in I_i$ for all sufficiently large $u \in \omega$, so P_i must eventually reach step 3. By the properties of S discussed above, the variable s in the procedure P_i eventually assumes a value in S. For this value, $k(\alpha - \alpha_s) > \beta - \beta_s$, from which it follows that $k(\alpha_u - \alpha_s) > \beta - \beta_s$ for some $u > s$, and hence that $\beta \in [\beta_s, \beta_s + k(\alpha_u - \alpha_s)]$. So β must be in one of the intervals $(*)$, all of which are in B_i. □

COROLLARY 3.6. *If α^0 and α^1 are c.e. reals such that $\alpha^0 + \alpha^1$ is random then at least one of α^0 and α^1 is random.*

Proof. Let $\beta = \alpha^0 + \alpha^1$. For each $s \in \omega$, either $3(\alpha^0 - \alpha^0_s) > \beta - \beta_s$ or $3(\alpha^1 - \alpha^1_s) > \beta - \beta_s$, so for some $i < 2$ there are infinitely many $s \in \omega$ such that $3(\alpha^i - \alpha^i_s) > \beta - \beta_s$. By Theorem 3.5, α^i is random. □

Combining Theorem 3.1 and Corollary 3.6, we have the following results, the second of which also depends on Theorem 1.5.

THEOREM 3.7. *A c.e. real γ is random if and only if it cannot be written as $\alpha + \beta$ for c.e. reals $\alpha, \beta <_s \gamma$.*

THEOREM 3.8. *Let* **d** *be a Solovay degree of c.e. reals. The following are equivalent:*

1. **d** *is incomplete.*
2. **d** *splits.*
3. **d** *splits over any lesser Solovay degree.*

REFERENCES

[1] K. AMBOS-SPIES, K. WEIHRAUCH, AND X. ZHENG, *Weakly computable real numbers*, J. Complexity, 16 (2000), pp. 676–690.

[2] C. S. CALUDE, *A characterization of c.e. random reals*, Theoret. Comput. Sci., 271 (2002), pp. 3–14.

[3] C. S. CALUDE, *Information and Randomness, an Algorithmic Perspective*, Monogr. Theoret. Comput. Sci. EATCS Ser., Springer–Verlag, Berlin, 1994.

[4] C. S. CALUDE AND G. CHAITIN, *Randomness everywhere*, Nature, 400 (1999), pp. 319–320.

[5] C. S. CALUDE, R. J. COLES, P. H. HERTLING, AND B. KHOUSSAINOV, *Degree-theoretic aspects of computably enumerable reals*, in Models and Computability (Leeds, 1997), London Math. Soc. Lecture Note Ser. 259, S. B. Cooper and J. K. Truss, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 23–39.

[6] C. S. CALUDE, P. H. HERTLING, B. KHOUSSAINOV, AND Y. WANG, *Recursively enumerable reals and Chaitin Ω numbers*, Theoret. Comput. Sci., 255 (2001), pp. 125–149.

[7] C. S. CALUDE AND A. NIES, *Chaitin Ω numbers and strong reducibilities*, J. Univ. Comp. Sci., 3 (1998), pp. 1162–1166.

[8] G. S. CEĬTIN, *A pseudofundamental sequence that is not equivalent to a monotone one*, Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI), 20 (1971), pp. 263–271, 290, (in Russian with English summary).

[9] G. J. CHAITIN, *A theory of program size formally identical to information theory*, J. Assoc. Comput. Mach., 22 (1975), pp. 329–340; reprinted in [12].

[10] G. J. CHAITIN, *Algorithmic information theory*, IBM J. Res. Develop., 21 (1977), pp. 350–359, 496; reprinted in [12].

[11] G. J. CHAITIN, *Incompleteness theorems for random reals*, Adv. in Appl. Math., 8 (1987), pp. 119–146; reprinted in [12].

[12] G. J. Chaitin, *Information, Randomness and Incompleteness*, Papers on Algorithmic Information Theory., 2nd ed., Series in Computer Science 8, World Scientific, River Edge, NJ, 1990.
[13] R. Downey, *Computability, definability, and algebraic structures*, in Proceedings of the 7th Asian Logic Conference, Hsi-Tou, Taiwan, Association for Symbolic Logic, to appear.
[14] R. Downey and G. LaForte, *Presentations of computably enumerable reals*, Theoret. Comput. Sci., to appear.
[15] R. G. Downey, D. R. Hirschfeldt, and G. LaForte, *Randomness and reducibility*, in Mathematical Foundations of Computer Science 2001, Lecture Notes in Comput. Sci. 2136, J. Sgall, A. Pultr, and P. Kolman, eds., Springer-Verlag, Berlin, 2001, pp. 316–327.
[16] C. Ho, *Relatively recursive reals and real functions*, Theoret. Comput. Sci., 210 (1999), pp. 99–120.
[17] K.-I. Ko, *On the definition of some complexity classes of real numbers*, Math. Systems Theory, 16 (1983), pp. 95–100.
[18] K.-I. Ko, *On the continued fraction representation of computable real numbers*, Theoret. Comput. Sci., 47 (1986), pp. 299–313.
[19] A. N. Kolmogorov, *Three approaches to the quantitative definition of information*, Internat. J. Comput. Math., 2 (1968), pp. 157–168.
[20] A. Kučera and T. Slaman, *Randomness and recursive enumerability*, SIAM J. Comput., 31 (2002), pp. 199–211.
[21] A. H. Lachlan, *Recursive real numbers*, J. Symbolic Logic, 28 (1963), pp. 1–16.
[22] L. Levin, *On the notion of a random sequence*, Soviet Math. Dokl., 14 (1973), pp. 1413–1416.
[23] L. Levin, *The various measures of the complexity of finite objects (an axiomatic description)*, Soviet Math. Dokl., 17 (1976), pp. 522–526.
[24] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer–Verlag, New York, 1997.
[25] P. Martin-Löf, *The definition of random sequences*, Inform. and Control, 9 (1966), pp. 602–619.
[26] H. Rice, *Recursive real numbers*, Proc. Amer. Math. Soc., 5 (1954), pp. 784–791.
[27] C.-P. Schnorr, *Process complexity and effective random tests*, J. Comput. System Sci., 7 (1973), pp. 376–388.
[28] R. I. Soare, *Cohesive sets and recursively enumerable Dedekind cuts*, Pacific J. Math., 31 (1969), pp. 215–231.
[29] R. I. Soare, *Recursively Enumerable Sets and Degrees*, Perspect. Math. Logic, Springer–Verlag, Heidelberg, 1987.
[30] R. J. Solomonoff, *A formal theory of inductive inference* I, Inform. and Control, 7 (1964), pp. 1–22.
[31] R. J. Solomonoff, *A formal theory of inductive inference* II, Inform. and Control, 7 (1964), pp. 224–254.
[32] R. M. Solovay, *Draft of a paper (or series of papers) on Chaitin's work ... done for the most part during the period of Sept.–Dec.* 1974, manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.
[33] R. M. Solovay, *On random r.e. sets*, in Non-Classical Logics, Model Theory, and Computability. Proceedings of the Third Latin-American Symposium on Mathematical Logic, Campinas, 1976, A. Arruda, N. Da Costa, and R. Chuaqui, eds., Stud. Logic and the Foundations of Math. 89, North-Holland, Amsterdam, 1977, pp. 283–307.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1184–1211

SPACE COMPLEXITY IN PROPOSITIONAL CALCULUS*

MICHAEL ALEKHNOVICH†, ELI BEN-SASSON‡, ALEXANDER A. RAZBOROV§, AND AVI WIGDERSON‡

Abstract. We study space complexity in the framework of propositional proofs. We consider a natural model analogous to Turing machines with a read-only input tape and such popular propositional proof systems as *resolution*, *polynomial calculus*, and *Frege* systems. We propose two different space measures, corresponding to the maximal number of bits, and clauses/monomials that need to be kept in the memory simultaneously. We prove a number of lower and upper bounds in these models, as well as some structural results concerning the clause space for resolution and Frege systems.

Key words. proof complexity, resolution, Frege, polynomial calculus

AMS subject classification. 03F20

PII. S0097539700366735

1. Introduction.

Complexity of propositional proofs plays as important a role in the theory of feasible proofs as the role played by the complexity of Boolean circuits in the theory of efficient computations. It is also well recognized that there exists a very productive cross-fertilization of techniques between the two fields. Partly because of this similarity, most of the research in the proof-complexity area concentrated on complexity measures related to *size*, which is the most interesting measure in the circuit complexity framework. In other words, the main effort in proof complexity was invested in investigating the amount of *time* (or at least time-like resources) taken by proofs; we recommend the excellent recent survey [BP98] for further reading on this subject.

During the workshop "Complexity Lower Bounds" held at the Fields Institute in Toronto in 1998, A. Haken raised the question of whether something intelligent can be said about the amount of *space* taken by propositional proofs. Quite surprisingly, it turned out that this very natural question had been virtually untouched in the past. Apparently, the only early paper devoted to the space of proofs is [Koz77], but it dealt only with *equational* theories involving no propositional connectives.

Recently, Esteban and Toran [ET99] proposed a convenient definition of space complexity for resolution which measures the number of clauses to be kept simultaneously in the memory to infer the tautology.[1] This model is analogous to a Turing machine computation, with a special read-only input tape from which the axioms can be downloaded to the working memory when needed and erased from the working memory as many times as necessary. They showed some upper and lower bounds for

*Received by the editors January 10, 2000; accepted for publication (in revised form) December 6, 2001; published electronically April 26, 2002.

http://www.siam.org/journals/sicomp/31-4/36673.html

†Moscow State University, Moscow, Russia (mike@mccme.ru). This author's research was supported by INTAS grant 96-753 and by the Russian Basic Research Foundation.

‡Institute of Computer Science, Hebrew University, Jerusalem, Israel (elli@cs.huji.ac.il, avi@cs.huji.ac.il). The research of the fourth author was supported by grant 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities, and by a grant from the Alfred P. Sloan Foundation.

§Steklov Mathematical Institute, Moscow, Russia (razborov@genesis.mi.ras.ru). This author's research was supported by INTAS grant 96-753 and by the Russian Basic Research Foundation.

[1]Throughout this paper, all propositional systems will prove that ϕ is a tautology by actually proving that its negation, $\neg\phi$, is unsatisfiable.

clause space (see section 3) and noticed the connection between the clause complexity for resolution and the pebbling game on the graph of a derivation.

Our goals in this paper are to generalize the natural notion of space complexity to other propositional proof systems and complexity measures and to research its properties. The first question arising is how to measure the memory content at any given moment of time for a specified proof system. Recall (see, e.g., [Kra95]) that the most customary measures for size complexity of propositional proofs are the bit size and the number of lines. Of these two, the bit size is by far more important, and can be defined analogously, and naturally, in the context of space complexity. The only simplification we allowed ourselves is in fact customary for the size complexity as well. Namely, instead of the bit space we consider the *variable* space (Definition 3.3) which is the overall number of occurrences of variables. This changes the complexity only by at most a logarithmic factor but makes things substantially cleaner.

It turns out that the line complexity is less adequate a measure for space complexity than it is a size measure. The reason is quite simple: if the language of the proof system is sufficiently strong and allows unbounded fan-in AND gates, then one gets only trivial results. Specifically, we can prove everything that is provable with just $O(1)$ memory cells, one of them containing a big AND of *all* formulae derived at previous steps.

One notable proof system that is not closed under the AND operation is *resolution*, in which case lines are just clauses. In the current paper we in particular show that the clause complexity, as opposed to the line complexity, makes perfect sense even for rather strong systems and can be considered as its natural replacement in space-complexity studies.

It turns out that all tautologies can be proven within *polynomial* space for any "reasonable" space measure. Specifically, *every* disjunctive normal form tautology in n variables already has a resolution proof with the clause space $(n + 1)$ [ET99], and this upper bound trivially holds for stronger proof systems. This in itself implies a *quadratic* upper bound on the *variable* space, but for the case of Frege systems we are able to improve it to a *linear* upper bound in the number of variables (Theorem 6.3).

These upper bounds determine the range of parameters in which the whole story develops. We ask which tautologies indeed require that much space and which can be proved within, say, (quasi)logarithmic space resources. We propose some lower bound techniques that in many cases allow us to answer this question for specific tautologies, proof systems, and space measures. It is worth noting that all these techniques are purely semantic in nature and thus can be applied to stronger *semantic* versions of the proof systems in question (see definitions in section 3.2). Let us also point out that it is not quite clear to which extent semantic versions of propositional proof systems are actually stronger than ordinary ones in the context of space complexity. On the contrary, we show for our weakest proof system (resolution) and for our strongest one (Frege system) that the space complexity differs from its semantic analogue by at most a constant multiplicative factor (Theorem 3.7, Corollary 6.6).

For many good reasons, bounded fan-in conjunctive normal forms (CNFs) (like Tseitin's tautologies) are always preferred in proving lower bounds or separation results. For tautologies from this class (and in the clause space model) we were able to prove strong lower bounds only for resolution (Theorem 3.18, Corollary 3.27.)[2]

Finally, we prove one lower bound in the variable space model that does not follow from our clause space bounds. Our argument applies in parallel to both

[2]Corollary 3.27 was also independently proved in [Tor99].

resolution and polynomial calculus (abbreviated throughout the paper by PC), and this situation is already familiar from the proof *size* complexity. For example, [CEI96] proved that every resolution proof of size S can be transformed into a PC proof of degree $O(\sqrt{n \log S})$, and [IPS97] used essentially the same argument for showing that every PC proof of size S can be also transformed into another PC proof of degree $O(\sqrt{n \log S})$. For that reason we find it very instructive to introduce a natural minimal common extension of resolution and PC called polynomial calculus augmented with resolution (PCR) that is at least as efficient as resolution in terms of proof size and space. The above-cited results [CEI96, IPS97] can then be considered as specifications of one general theorem about PCR which says that the degree of every size S PCR proof can be reduced to $O(\sqrt{n \log S})$. Our quadratic lower bounds for the variable space are also very naturally formulated and proved in terms of this new system (Theorem 5.1).

1.1. Summary of results. We introduce the clause space and variable space measures for resolution and PC. The *clause space* of a proof is the maximal number of clauses/monomials that need to be kept in the memory during the verification of a proof, and the *variable space* is the maximal number of overall symbols that need to be kept during such a verification.

We prove tight lower bounds for resolution clause space for a variety of formulas that includes the *pigeonhole principle*, *counting principles*, and several other interesting cases. This is done by proving a general lower bound that applies to all these cases. Via a different technique, we present a lower bound for the graph-based *Tseitin tautologies* that is *linear* in the number of variables appearing in this formula, and hence optimal.

For PC we prove nearly optimal (up to a small multiplicative constant) lower bounds for *wide tautologies* that include the pigeonhole principle. For this proof we use more complicated techniques than those used for the case of resolution. We show that these techniques are needed by proving that, for some cases, PC is strictly more efficient than resolution.

Using our clause space lower bounds for resolution and PC, we derive nearly optimal (up to a small multiplicative constant) lower bounds for the variable space of wide tautologies, such as the pigeonhole principle.

Finally, we prove *linear*, and hence optimal, *upper* bounds on the variable space of Frege proofs for *any* tautology.

1.2. Paper organization. Following several general definitions (section 2) we define the resolution clause space measure and prove lower bounds for it in section 3. In section 4 we define PC and its extension to multivalued logic and prove our clause space lower bounds for this system. The variable space lower bounds for resolution and PC appear in section 5. Finally, we present optimal upper bounds for Frege variable space in section 6 and conclude with some interesting open problems (section 7).

2. General definitions. Let x be a Boolean variable, i.e., a variable that ranges over the set $\{0,1\}$. Throughout this paper we shall identify $\mathbf{1}$ with $True$ and $\mathbf{0}$ with $False$. A *literal* of x is either x (denoted sometimes as x^1) or $\bar{x}$ (denoted sometimes as x^0). A *clause* is a disjunction of literals. We write $x^\epsilon \in C$ iff the clause C contains the literal x^ϵ. A *CNF formula* is a set of clauses.

For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, $Vars(f)$ will denote its set of variables. An *assignment to* f is a mapping $\alpha : Vars(f) \to \{0,1\}$. A *restriction* is a mapping $\rho : Vars(f) \to \{0,1,\star\}$. We denote by $|\rho|$ the number of assigned

variables, $|\rho| = |\rho^{-1}(\{0,1\})|$. We say that a restriction ρ' *extends* ρ iff they coincide on $\rho^{-1}(\{0,1\})$.

The restriction of f by ρ, denoted $f|_\rho$, is the Boolean function obtained from f by setting the value of each $x \in \rho^{-1}(\{0,1\})$ to $\rho(x)$ and leaving each $x \in \rho^{-1}(\star)$ as a variable.

We say that an assignment α *satisfies* f if $f(\alpha) = 1$. For Boolean functions $f_1, \ldots, f_k, g$ we say that $f_1, \ldots, f_k$ *semantically imply* g, $f_1, \ldots, f_k \models g$ if every assignment to $V \stackrel{\text{def}}{=} Vars(f_1) \cup \cdots \cup Vars(f_k) \cup Vars(g)$ satisfying $f_1, \ldots, f_k$ satisfies g as well (i.e., for all $\alpha \in \{0,1\}^V (f_1(\alpha) = \cdots = f_k(\alpha) = 1 \Rightarrow g(\alpha) = 1)$). For $\mathcal{F}, \mathcal{G}$, two sets of functions, we say that $\mathcal{F}$ implies $\mathcal{G}$ ($\mathcal{F} \models \mathcal{G}$) if, for all $g \in \mathcal{G}$, $\mathcal{F} \models g$.

Notation. Throughout this paper, a, b will denote Boolean constants, x, y, z will denote Boolean variables; f, g, h will denote functions; φ, ψ will denote formulas; A, B, C, D will denote clauses; α, β will denote assignments; and ρ will denote restrictions. Calligraphic letters $\mathcal{A}, \mathcal{M}, \mathcal{N}, \mathcal{T}$ will denote sets of formulas. For n, a nonnegative integer, let $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. For $\mathcal{M}$, a set, we denote by $|\mathcal{M}|$ its cardinality.

3. Resolution clause space. In this section we prove lower bounds for resolution clause space for a number of principles which include various modifications of the pigeonhole principle, counting principles, and the principle GT_n. (The latter was used in the recent work of [BG99] to produce a tautology with large minimal proof width and polynomially bounded proof size.) These results follow from a general lower bound for *semiwide tautologies*, also introduced in this section (Theorem 3.13). We also show how to transform any semiwide tautology (and, in particular, any of the above mentioned examples) to an equivalent 3-CNF form while preserving clause space hardness (Theorem 3.18).

We will start by giving definitions of the resolution clause space, proceed to show the equivalence of *semantic* and *syntactic* resolution, and then prove the lower bounds.

Notation. Throughout section 3 we do not distinguish between a clause C and the Boolean function computed by it.

3.1. Resolution clause space—definitions. Recall that a CNF formula is a set of clauses. The *resolution rule* is the following derivation rule.

Resolution rule. Derive $A \vee B$ from $\{A \vee x, B \vee \overline{x}\}$, where A, B are any clauses and x is any variable.

Let $\mathcal{T} = \{C_1, C_2, \ldots, C_m\}$ be a CNF formula over n variables. A *resolution proof* (or *derivation*) of a clause E from $\mathcal{T}$ is a sequence of clauses $\pi = \{D_1, D_2 \ldots, D_s\}$ such that the last clause is E and each D_i is either some initial clause $C_j \in \mathcal{T}$ or is derived from previous clauses using the resolution rule. A *resolution refutation of* $\mathcal{T}$ is a resolution derivation of the empty clause, $\mathbf{0}$ from $\mathcal{T}$. Notice that by the definition of the resolution rule, all lines in a resolution proof must be clauses. Notice that every refutation gives rise to a labeled directed acyclic graph, called the *refutation DAG*. Each node in this graph is labeled by a clause of the proof. The sources are labeled by $C \in \mathcal{T}$, the single sink by $\mathbf{0}$, and each node in the middle is connected to the two clauses that were used to derive it.

The definition of the resolution clause space was first given in [ET99]. We recast it here in slightly different terms (so that it will be easier for us to generalize this definition for stronger proof systems in the forthcoming sections).

Suppose we are given a resolution proof from $\mathcal{T}$ and wish to verify it using a minimal amount of memory. The proof π, as well as the CNF $\mathcal{T}$, are written on a

read-only memory tape. We keep in our *working memory* a subset of the clauses in the proof (starting with the empty set), and at each time step we either add a clause $C \in \mathcal{T}$ to our memory, *or* apply a single resolution derivation to two clauses *in the memory*, and add the resulting clause to it, *or* remove an unnecessary clause from the memory. The *clause space* of the proof π is the maximal number of clauses we need to keep at some time during the verification of the proof. Of course, there are many ways to verify a single proof (e.g., we may keep all clauses in the proof, in which case the space will be $|\pi|$), and we naturally define the space to be the minimal one over all possible verifications of π. Finally, the *resolution clause space* of $\mathcal{T}$ is the minimal space of a refutation π of $\mathcal{T}$ (if such a refutation exists, and ∞ otherwise).

We now present a formal definition of the clause space that captures our intuition and will be easy to work with rigorously. We start with a different definition of a resolution proof that exposes the clause space naturally. Comparing this definition with our previous one, we see they are equivalent in size, up to a polynomial factor.

DEFINITION 3.1 (syntactical resolution derivation). *A* configuration *is a set of clauses. A* proof π *from a CNF* $\mathcal{T}$ *is a sequence of configurations* $\mathcal{M}_0, \ldots, \mathcal{M}_s$ *such that* $\mathcal{M}_0 = \emptyset$ *and, for all* $t \in [s]$, $\mathcal{M}_t$ *is obtained from* $\mathcal{M}_{t-1}$ *by one of the following rules:*

AXIOM DOWNLOAD. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup C$ *for some clause* $C \in \mathcal{T}$;
MEMORY ERASING. $\mathcal{M}_t := \mathcal{M}_{t-1} - \mathcal{M}'$ *for some* $\mathcal{M}' \subseteq \mathcal{M}_{t-1}$;
INFERENCE. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup C$, *for some* C *obtained by a single application of the resolution rule to two clauses in* $\mathcal{M}_{t-1}$.

We use the notation $\mathcal{M} \rightsquigarrow \mathcal{M}'$ *to mean that* $\mathcal{M}'$ *is the immediate successor of* $\mathcal{M}$ *in a derivation. If* $\mathcal{M}_s = \{\mathbf{0}\}$ *(the empty clause), then the derivation is called a* refutation of $\mathcal{T}$.

DEFINITION 3.2 (clause space). *The* clause space *of a set of configurations* $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_s\}$ *is*

$$CSpace(\pi) \stackrel{\text{def}}{=} \max\{|\mathcal{M}_i| : i \in [s]\}.$$

The resolution clause space *of a CNF* $\mathcal{T}$ *is*

$$CSpace(\mathcal{T}) \stackrel{\text{def}}{=} \min\{CSpace(\pi)\},$$

where the minimum is taken over all refutations of $\mathcal{T}$, *and is defined to be* ∞ *if no such refutation exists (i.e.,* $\mathcal{T}$ *is satisfiable).*

Although we will prove variable space lower bounds only in section 5 (and in this section concentrate on *clause* space lower bounds), we nonetheless present its definition now (for the sake of coherence).

DEFINITION 3.3 (variable space). *The* variable space *of a configuration* $\mathcal{M}$ *is* $\sum_{C \in \mathcal{M}} |C|$, *where* $|C|$ *is the number of literals in* C. *The* variable space of a set of configurations $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_s\}$ *is*

$$VSpace_R(\pi) \stackrel{\text{def}}{=} \max\left\{\sum_{C \in \mathcal{M}_i} |C| : i \in [s]\right\}.$$

The resolution variable space *of a CNF* $\mathcal{T}$ *is*

$$VSpace_R(\mathcal{T}) \stackrel{\text{def}}{=} \min\{VSpace_R(\pi)\},$$

where the minimum is taken over all refutations π of $\mathcal{T}$, and is defined to be ∞ if no such refutation exists (i.e., $\mathcal{T}$ is satisfiable).

As we already mentioned, the research of clause space in resolution was started by Esteban and Toran in [ET99]. They in particular defined the following tautologies[3] CT_n and showed hardness of their proof in terms of clause space.

DEFINITION 3.4 (COMPLETE-TREE tautologies). *CT_n is the following set of axioms:*

$$\{ x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_n^{\epsilon_n} \mid \vec{\epsilon} \in \{0,1\}^n \}.$$

They also gave an upper bound for *any* tautology $\mathcal{T}$ over n variables. (We state it in our language.)

THEOREM 3.5 (see [ET99]). *If $\mathcal{T}$ is a contradictory set of clauses over n variables, then $CSpace(\mathcal{T}) \leq n+1$.*

This bound is tight for the principle CT_n.

CT_n contains an exponential number of axioms. In this section we define the class of *semiwide* tautologies of polynomial size which are hard to refute for resolution in terms of clause space. This class contains such popular principles as PHP_n^m, $onto - PHP_n^m$, $Count_p$, GT_n. We also show via a slightly different approach the space hardness of Tseitin tautologies for expander graphs.

3.2. Equivalence of syntactic and semantic resolution. All our lower bounds in this paper will work for *semantical* proof systems, which are stronger versions of the regular proof systems. In this subsection we define semantical resolution and then prove that, with respect to clause space, syntactical and semantical derivations are equivalent, up to a constant factor.

We start by pointing out that the *inference* rule of Definition 3.1 is *sound*; i.e., if $\mathcal{M}_t$ was derived from $\mathcal{M}_{t-1}$ by an application of this rule, then $\mathcal{M}_{t-1} \models \mathcal{M}_t$. The *semantical resolution proof system* replaces this rule by the following stronger *semantical inference* rule.

SEMANTICAL INFERENCE. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup C$ for some C such that $\mathcal{M}_{t-1} \models C$.

The definition of proof space in semantical resolution is analogous to that of the syntactical resolution system. Denote by $CSpace^{sem}(\mathcal{T})$ the clause space of refuting $\mathcal{T}$ in semantical resolution.

The semantical inference rule generalizes the inference rule, and hence any syntactical resolution derivation is also a semantical one. Notice that when it comes to *size*, semantical resolution is much stronger than syntactical resolution because once $\mathcal{M}_t$ is unsatisfiable, we can derive $\mathbf{0}$ in a single step. We will now prove that with respect to space, there is no big difference between the two, and they are equivalent up to a constant factor.

A CNF formula is *minimal unsatisfiable* if it is unsatisfiable and removing any clause from it will make it satisfiable. We will need a very useful theorem, due to Tarsi [Tar], which originally appeared in [AL86].

THEOREM 3.6 (Tarsi's theorem). *If $\mathcal{T}$ is a minimal unsatisfiable CNF formula on n variables and m clauses, then $m > n$.*

Proof. Consider the following bipartite graph on $\mathcal{T} \times Vars(\mathcal{T})$: a clause C is connected to a variable x iff x appears in C (either as a positive or negative literal).

[3]Throughout the paper we assume that all tautologies are represented in the negated form of a contradictory CNF $\mathcal{T}$.

Clearly, there is no matching from $\mathcal{T}$ to $Vars(\mathcal{T})$ in this graph because the formula is unsatisfiable. Hence, by Hall's theorem, there must be some set $S \subset \mathcal{T}$ such that $|N(S)| < |S|$. Let S be such a set of maximal size. By the maximality of S, for any $V \subseteq \mathcal{T} - S$ we have $|N(V) - N(S)| \geq |V|$. Thus there is a matching from $\mathcal{T} - S$ into $Vars(\mathcal{T}) - N(S)$. By the minimal unsatisfiability of $\mathcal{T}$, if $S \neq \mathcal{T}$, then S is satisfiable, and $\mathcal{T} - S$ is satisfiable by its matching into $Vars(\mathcal{T}) - N(S)$. Thus we conclude that $S = \mathcal{T}$ and hence $m = |S| = |\mathcal{T}| > |N(S)| = |Vars(\mathcal{T})| = n$. □

The main theorem of this subsection is the following.

THEOREM 3.7. *For any unsatisfiable CNF* $\mathcal{T}$*,*

$$CSpace^{sem}(\mathcal{T}) \leq CSpace(\mathcal{T}) \leq 2 \cdot CSpace^{sem}(\mathcal{T}).$$

Proof. The first inequality is trivial because a syntactical resolution derivation is also a legitimate semantical one. Let π be a semantical refutation of $\mathcal{T}$ with clause space s. We wish to show that $\mathcal{T}$ has a clause space $2s$ syntactical refutation. The only difference between the two systems is in the inference rule, so we focus on this rule. Suppose $\mathcal{M}_{t+1}$ was inferred from $\mathcal{M}_t$ by the semantic inference rule ($\mathcal{M}_{t+1} := \mathcal{M}_t \cup \{C\}$, $\mathcal{M}_t \models C$), where $|\mathcal{M}_t|, |\mathcal{M}_{t+1}| \leq s$.

Let ρ be the unique minimal size restriction on the variables of C such that $C|_\rho = 0$. Now we use the soundness of the step to claim that $\mathcal{M}_t|_\rho$ must be unsatisfiable. $\mathcal{M}_t|_\rho$ contains a minimal unsatisfiable subformula $\mathcal{M}'$, which, by Tarsi's theorem, has at most $s - 1$ variables. By Theorem 3.5, the contradictory set of clauses $\mathcal{M}'$ can be refuted in space s. It is an easy exercise to adjust this proof and derive C from $\mathcal{M}_t$ with the same clause space s.

We need a space of s cells for saving $\mathcal{M}_t$. Additionally, we use s cells to derive $C \in \mathcal{M}_{t+1}$ from $\mathcal{M}_t$. □

3.3. Lower bounds for semiwide tautologies. We are ready to prove our lower bounds for the clause space of semantical resolution proofs. The main idea is the following. Consider a derivation that keeps at most k clauses in the memory. We shall show that for some classes of tautologies (called *semiwide*; see Definition 3.10) we can inductively construct restrictions of maximal size k that satisfy the memory content. Thus, there can be no space k refutation because the empty clause cannot be satisfied.

Although this proof technique is very simple, we will use similar ideas when proving lower bounds for stronger proof systems. Also, these results will be useful for proving lower bounds for the variable space complexity. For this reason we present our proof method in a somewhat fancy style.

The main idea of our lower bounds is to come up with some set of memory-configurations $\mathbb{A}$ ("$\mathbb{A}$" stands for "admissible") such that

- it does not contain contradictory configurations;
- any memory-configuration achievable in small memory is the semantical corollary of some configuration from $\mathbb{A}$.

Notice that we do *not* require $\mathbb{A}$ to contain all the formulas which can be derived using small space. We require only that, for any such formula φ, there exists a memory configuration $\mathcal{M} \in \mathbb{A}$ implying φ.

In all our cases these "dominating" memory-configurations will be CNF's of a very simple nature, namely, sets of *disjoint* clauses:

$$\mathcal{M} \in \mathbb{A} \Rightarrow \mathcal{M} = \left\{ \bigvee_{j \in J_i} x_j^{\epsilon_j} \mid 1 \leq i \leq k \right\},$$

where $J_i \cap J_{i'} = \emptyset$ for different i, i'. Moreover, in this section we will be interested only in the following partial case of this definition naturally corresponding to ordinary restrictions.

DEFINITION 3.8 (proper 1-CNF's). *$\mathcal{M}$ is called a* proper 1-CNF *if it is a set of pairwise distinct literals, i.e.,*

$$\mathcal{M} = \{x_{j_1}^{\epsilon_1}, x_{j_2}^{\epsilon_2}, \ldots, x_{j_k}^{\epsilon_k}\},$$

and $j_{i_1} \neq j_{i_2}$ for different i_1, i_2.

The heart of our lower bounds for clause space for resolution is the following (trivial) locality lemma which informally claims that small 1-CNF's are enough to imply any small space consequence of the axioms. Later on we shall present an analogous locality lemma, Lemma 4.14, for PC, which will be less trivial.

LEMMA 3.9 (locality lemma for resolution). *Let $\mathcal{M}$ be a proper 1-CNF. Suppose that $\mathcal{M}_1$ is the semantical resolution corollary of $\mathcal{M}$ (i.e., $\mathcal{M}_1$ is a set of clauses and $\mathcal{M} \models \mathcal{M}_1$); let $|\mathcal{M}_1| = s$. Then there exists $\mathcal{M}_1^{-1} \subseteq \mathcal{M}$ such that $\mathcal{M}_1^{-1} \models \mathcal{M}_1$ and $|\mathcal{M}_1^{-1}| \leq s$.*

Proof. Suppose $\mathcal{M}_1 = \{C_1, C_2, \ldots, C_s\}$. For any clause C_i which is the semantical corollary of $\mathcal{M}$ there exists some $x_{j_i}^{\epsilon_i} \in \mathcal{M} \cap C_i$. Thus $\mathcal{M}_1$ is the semantical corollary of the configuration $\mathcal{M}_1^{-1} = \{x_{j_1}^{\epsilon_1}, x_{j_2}^{\epsilon_2}, \ldots, x_{j_s}^{\epsilon_s}\}$. □

A contradictory set of clauses is called *n-wide* if all its axioms have width (= the number of literals) $\geq n$. [ET99] proved that every n-wide tautology has clause space $> n$. In section 4.4 we shall prove PC lower bounds for these tautologies. In the case of the weaker resolution system, we can prove lower bounds for a bigger class of *semiwide* tautologies.

DEFINITION 3.10 (semiwide tautologies). *Suppose that $\mathcal{T}$ is a contradictory set of clauses which is divided into two groups: $\mathcal{T} = \mathcal{P} \dot{\cup} \mathcal{R}$, where $\mathcal{P}$ is satisfiable.*

For $\mathcal{M}$, a proper 1-CNF, we say that $\mathcal{M}$ is $\mathcal{P}$-consistent *iff $\mathcal{P} \cup \mathcal{M}$ is consistent. Equivalently, $\mathcal{M}$ is $\mathcal{P}$-consistent iff it can be extended to a proper 1-CNF which implies all axioms of $\mathcal{P}$ ($\exists \mathcal{M}' \supseteq \mathcal{M}$ $(\mathcal{M}' \models \mathcal{P})$).*

Finally, $\mathcal{T}$ is n-semiwide iff there exists a partition $\mathcal{T} = \mathcal{P} \dot{\cup} \mathcal{R}$ such that $\mathcal{P}$ is satisfiable and for every axiom $C \in \mathcal{R}$, for every $\mathcal{P}$-consistent proper 1-CNF $\mathcal{M}$ with $|\mathcal{M}| < n$, it can be extended to a $\mathcal{P}$-consistent proper 1-CNF $\mathcal{M}' \supseteq \mathcal{M}$ such that $\mathcal{M}' \models C$.

Before we show that n-semiwide tautologies demand clause space at least $n + 1$ to be refuted, we give several natural examples.

It is obvious that every n-wide tautology is also n-semiwide: we let $\mathcal{P} = \emptyset$, and if we fix the values of $n - 1$ variables by a proper 1-CNF $\mathcal{M}$, any clause $C = x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_N^{\epsilon_N}$ with $N \geq n$ can be satisfied by fixing some unassigned variable (i.e., the variable which is not contained in $\mathcal{M}$) x_j. In particular, the COMPLETE-TREE tautology CT_n is n-semiwide.

Another example is the pigeonhole principle with m pigeons and n holes PHP_n^m which states that there is no 1-1 map from $[m]$ to $[n]$, as long as $m > n$. The propositional formulation of this principle has received much consideration in proof complexity and is one of the major test cases for comparing different proof systems. In particular, our $(n + 1)$ lower bound on the clause space of refuting PHP_n^m was independently proved in [Tor99].

We discuss here only the onto-version of this principle which constrains 1-1 maps to be onto. One can easily extend our arguments to other PHP-like principles and $Count_p$.

DEFINITION 3.11 (onto-pigeonhole principle). *$Onto - PHP_n^m$ is the union of the following four groups of axioms:*

(i) $P_i \stackrel{\text{def}}{=} \bigvee_{1\le j\le n} x_{ij}$ $(i \in [m])$;

(ii) $H_j \stackrel{\text{def}}{=} \bigvee_{1\le i\le n} x_{ij}$ $(j \in [n])$;

(iii) $Q_{i_1,i_2;j} \stackrel{\text{def}}{=} \bar{x}_{i_1 j} \vee \bar{x}_{i_2 j}$ $(i_1, i_2 \in [m],\ i_1 \neq i_2;\ j \in [n])$;

(iv) $Q_{i;j_1,j_2} \stackrel{\text{def}}{=} \bar{x}_{ij_1} \vee \bar{x}_{ij_2}$ $(i \in [m];\ j_1, j_2 \in [n],\ j_1 \neq j_2)$.

One can see that $onto - PHP_n^m$ is n-semiwide by taking the partition $\mathcal{P} \dot\cup \mathcal{R}$ with $\mathcal{P} = \{\, Q_{i_1,i_2;j} \mid i_1, i_2, j \,\} \cup \{\, Q_{i;j_1,j_2} \mid i, j_1, j_2 \,\}$ and $\mathcal{R} = \{\, P_i \mid i \,\} \cup \{\, H_j \mid j \,\}$. The proper 1-CNF $\mathcal{M}$ is $\mathcal{P}$-consistent iff it does not put either two pigeons in the same hole (i.e., contains the literals $x_{i_1 j}, x_{i_2 j}$) or one pigeon in two different holes (i.e., contains the literals x_{ij_1}, x_{ij_2}), in other words iff positive literals in $\mathcal{M}$ form a partial matching. Now if $n-1$ variables are fixed, then when we take the axiom P_i we can put the ith pigeon in some unassigned hole (that is, to add the corresponding positive literal). Dually, if we take the axiom H_j we can put some unassigned pigeon in the jth hole. Thus we can always satisfy an axiom from $\mathcal{R}$ with some extended $\mathcal{P}$-consistent proper 1-CNF.

Another principle, GT_n, states that in every transitive directed graph which does not contain cycles of size two, there must exist a source node with no incoming edges. This principle, formulated in [Kri85], was shown to have a proof of polynomial size [Sta96]. Recently, [BG99] used this principle to produce a tautology of polynomial proof size and large minimal proof width.

Quite surprisingly, this very same principle also shows that large clause space complexity does not imply large proof size.

DEFINITION 3.12. *GT_n is the following contradictory set of axioms over $n(n-1)$ variables x_{ij} $(i, j \in [n], i \neq j)$ consisting of three groups:*

(i) $T_{ijk} \stackrel{\text{def}}{=} (x_{ij} \wedge x_{jk}) \to x_{ik}$ $(i, j, k \in [n], i \neq j \neq k)$;

(ii) $C_{ij} \stackrel{\text{def}}{=} \bar{x}_{ij} \vee \bar{x}_{ji}$ $(i, j \in [n], i \neq j)$;

(iii) $S_j \stackrel{\text{def}}{=} \bigvee_{k\neq j} x_{kj}$ $(j \in [n])$.

The first group of axioms says that the graph is transitive. The second group states that there are no cycles of size two. The axiom S_j says that j is not the source node. Clearly, this set of axioms is contradictory. To see that it is $\frac{n}{2}$-semiwide take the partition $\mathcal{T} = \mathcal{P} \dot\cup \mathcal{R}$, where $\mathcal{P}$ consists of the axioms of the first and second groups. Then the proper 1-CNF $\mathcal{M}$ is $\mathcal{P}$-consistent iff it does not contain a cycle of positive literals (i.e., chains like $x_{i_1 i_2}, x_{i_2 i_3}, \ldots, x_{i_k i_1}$) and it does not contain a chain $x_{i_1 i_2}, x_{i_2 i_3}, \ldots, x_{i_{k-1} i_k}$ together with the literal $\bar{x}_{i_1 i_k}$. Now suppose that $\mathcal{M}$ is $\mathcal{P}$-consistent and assigns not more than $\frac{n}{2} - 1$ variables and we are to extend it to satisfy some axiom $S_j \in \mathcal{R}$. We can choose the index $k \neq j$ such that no variables $x_{ki}, x_{ik}, i \in [n]$ are contained in $\mathcal{M}$ and let $\mathcal{M}' = \mathcal{M} \cup \{x_{kj}\}$. Thus GT_n is $\frac{n}{2}$-semiwide.

In the next theorem, which is the main result of this section, we show that semiwide tautologies are hard for resolution in terms of clause space. The intuition of the proof is simple: we show inductively that a configuration of $k < n$ clauses can be satisfied by a restriction that sets $\le k$ variables.

THEOREM 3.13. *For any n-semiwide tautology $\mathcal{T}$,*

$$CSpace^{sem}(\mathcal{T}) > n.$$

Proof. Fix the partition $\mathcal{T} = \mathcal{P} \dot\cup \mathcal{R}$ in accordance with Definition 3.10.

DEFINITION 3.14 (admissible configurations for $\mathcal{T}$). *We call* $\mathcal{M}$ admissible *iff* $\mathcal{M}$ *is a* $\mathcal{P}$*-consistent proper* 1*-CNF and* $|\mathcal{M}| \le n$.

Consider the set $\mathbb{A}$ of all admissible configurations. We claim that the following holds: for each configuration $\mathcal{M}$, derivable in space $\le n$, there exists a configuration $\mathcal{M}^{-1} \in \mathbb{A}$ such that $\mathcal{M}^{-1} \models \mathcal{M}$ and $|\mathcal{M}^{-1}| \le |\mathcal{M}|$. This is obviously enough to prove the theorem since $\mathbb{A}$ does not contain contradictory configurations.

We prove it by induction. The basis is trivial: in the beginning of the derivation the memory contains an empty set. To check the induction step suppose that $\mathcal{M}_t \rightsquigarrow \mathcal{M}_{t+1}$ and that there exists $\mathcal{M}_t^{-1} \in \mathbb{A}$ such that $\mathcal{M}_t^{-1} \models \mathcal{M}_t,\ |\mathcal{M}_t^{-1}| \le |\mathcal{M}_t|$.

Let us consider the three cases corresponding to the possible derivation steps: axiom download, semantic inference step, and memory erasing.

Axiom download ($\mathcal{M}_{t+1} := \mathcal{M}_t \cup \{C\},\ C \in \mathcal{T}$). In this case $|\mathcal{M}_t| \le n-1$ (since there is free space for the new axiom). Thus $|\mathcal{M}_t^{-1}| \le n-1$; hence it can be extended to $\mathcal{P}$-consistent proper 1-CNF $\mathcal{M}_{t+1}^{-1}$ which satisfies C. (If $C \in \mathcal{R}$ this follows from the definition of semiwide tautology, and if $C \in \mathcal{P}$ it is obvious.) Since C is a clause we can satisfy it by fixing just one variable. Thus we can assume without loss of generality (w.l.o.g.) that $|\mathcal{M}_{t+1}^{-1}| \le |\mathcal{M}_t^{-1}| + 1$.

Clearly, $\mathcal{M}_{t+1}^{-1} \in \mathbb{A}$ and $\mathcal{M}_{t+1}^{-1} \models \mathcal{M}_{t+1}$. Also, $|\mathcal{M}_{t+1}^{-1}| \le |\mathcal{M}_t^{-1}| + 1 \le |\mathcal{M}_t| + 1 = |\mathcal{M}_{t+1}|$.

Semantical inference ($\mathcal{M}_{t+1} := \mathcal{M}_t \cup \{C\},\ \mathcal{M}_t \models C$). In this case $\mathcal{M}_t^{-1} \models \mathcal{M}_{t+1}$. By the locality lemma, Lemma 3.9, there exists $\mathcal{M}_{t+1}^{-1} \subseteq \mathcal{M}_t^{-1}$ such that $\mathcal{M}_{t+1}^{-1} \models \mathcal{M}_{t+1}$ and $|\mathcal{M}_{t+1}^{-1}| \le |\mathcal{M}_{t+1}|$.

Memory erasing ($\mathcal{M}_{t+1} := \mathcal{M}_t - \mathcal{M}' \subseteq \mathcal{M}_t$). This is analogous to the case of inference step.

Theorem 3.13 follows. □

So far we have strongly used the existence of axioms of width n to show that a given tautology is semiwide and thus prove a lower bound of n on the clause space. We now go one step further to show that we can transform any semiwide tautology to the following 3-CNF version which requires essentially the same space as the "standard" version.

DEFINITION 3.15 (strong nondeterministic extensions). *For* $C(\vec{x})$ *a clause, a* strong nondeterministic extension of C *is any Boolean function* $f(\vec{x}, \vec{y})$ *such that*

- *if* $C(\vec{\alpha}) = 0$*, then* $f(\vec{\alpha}, \vec{y}) \equiv 0$*;*
- *if* $x_j^\epsilon \in C$*, then there exists an assignment* $\vec{\beta}$ *to* $\vec{y}$ *such that setting* x_j *to* ϵ *and* $\vec{y}$ *to* $\vec{\beta}$ *fixes* f *to* 1*. Formally,*

$$f\left[\epsilon/x_j,\ \vec{\beta}/\vec{y}\right] \equiv 1.$$

Example 3.16. One standard strong nondeterministic extension of a clause $C = x_1 \vee x_2 \vee \cdots \vee x_n$ is the function represented by the following 3-CNF family over $n+2$ clauses and $2n+1$ variables:

$$\{\bar{y}_0\} \cup \{y_{j-1} \vee x_j \vee \bar{y}_j \mid 1 \le j \le n\} \cup \{y_n\}.$$

DEFINITION 3.17 (extended version of $\mathcal{T}$). *An* extended version of the tautology $\mathcal{T}$*, denoted* $\widetilde{\mathcal{T}}$*, is derived by replacing every axiom* C_i *with some CNF set of clauses* $\mathcal{EC}_i$ *representing a strong nondeterministic extension of* C_i *such that distinct* $\mathcal{EC}_i$ *use distinct extension variables* $\vec{y_i} = \langle y_{i1}, y_{i2}, \ldots, y_{ik_i} \rangle$.

THEOREM 3.18. *If $\mathcal{T}$ is n-semiwide, and $\widetilde{\mathcal{T}}$ is some extended version of it, then*

$$CSpace^{sem}(\widetilde{\mathcal{T}}) > n.$$

Proof. As in the proof of Theorem 3.13, we are going to define the set of admissible configurations $\mathbb{A}$. After that the proof will be very similar to that of Theorem 3.13. Let, as before, the partition $\mathcal{T} = \mathcal{P} \dot\cup \mathcal{R}$ be chosen according to Definition 3.10. For every clause $C_i \in \mathcal{T}$ and every $x_j^\epsilon \in C_i$ we *rigidly fix once and for all* an arbitrary assignment $\vec\beta_{ij}$ to the variables $\vec y_i$ such that the restriction which sends x_j to ϵ and sends $\vec y_i$ to $\vec\beta_{ij}$ forces to 1 all clauses from $\mathcal{EC}_i$.

DEFINITION 3.19 (admissible configurations for $\widetilde{\mathcal{T}}$). *We call a proper 1-CNF $\mathcal{M}$ admissible for $\widetilde{\mathcal{T}}$ iff there exists a $\mathcal{T}$-admissible configuration $\widehat{\mathcal{M}}$ (in ordinary variables $\vec x$) such that*

- *for every original variable $x_j^\epsilon \in \mathcal{M}$, we also have $x_j^\epsilon \in \widehat{\mathcal{M}}$;*
- *if $\mathcal{M}$ contains at least one auxiliary variable from $\vec y_i$, then there exists $x_j^\epsilon \in \widehat{\mathcal{M}}$ such that the values of* all *auxiliary variables in $\vec y_i$ belonging to $\mathcal{M}$ are consistent with $\vec\beta_{ij}$.*

Consider the set $\mathbb{A}$ of all configurations admissible for $\widetilde{\mathcal{T}}$. We claim that the following holds: for each configuration $\mathcal{M}$ derivable in space n, there exists a configuration $\mathcal{M}^{-1} \in \mathbb{A}$ such that $\mathcal{M}^{-1} \models \mathcal{M}$ and $|\mathcal{M}^{-1}| \le |\mathcal{M}|$. This is obviously enough to prove the theorem since $\mathbb{A}$ does not contain contradictory configurations.

We prove it by induction. The basis, inference step, and memory erasing can be handled with the help of the locality lemma, Lemma 3.9, as in Theorem 3.13.

Consider the axiom download. Let $\mathcal{M}_{t+1} \leftarrow \mathcal{M}_t \cup \{C\}$, $C \in \mathcal{EC}_i$. By the induction hypothesis there exist configurations $\mathcal{M}_t^{-1}, \widehat{\mathcal{M}}_t^{-1}$ with properties described in Definition 3.19 and such that $\mathcal{M}_t^{-1} \models \mathcal{M}_t$ and $|\mathcal{M}_t^{-1}| \le |\mathcal{M}_t|$.

Case 1. $\mathcal{M}_t^{-1}$ already contains some auxiliary variable from $\mathcal{EC}_i$.

Then we have already assigned in $\widehat{\mathcal{M}}_t^{-1}$ some variable x_j with $x_j^\epsilon \in C_i$ to ϵ such that the values of all auxiliary variables $\vec y_i$ in $\mathcal{M}_t^{-1}$ are consistent with $\vec\beta_{ij}$. Either $x_j^\epsilon \in C$ or $y_{i\ell}^{\beta_{ij\ell}} \in C$ for some $\ell \le k_i$. In the first case we let $\mathcal{M}_{t+1}^{-1} \stackrel{\text{def}}{=} \mathcal{M}_t^{-1} \cup \{x_j^\epsilon\}$, and in the second case we let $\mathcal{M}_{t+1}^{-1} \stackrel{\text{def}}{=} \mathcal{M}_t^{-1} \cup \{y_{i\ell}^{\beta_{ij\ell}}\}$. Also put $\widehat{\mathcal{M}}_{t+1}^{-1} \stackrel{\text{def}}{=} \widehat{\mathcal{M}}_t^{-1}$.

Case 2. $\mathcal{M}_t^{-1}$ does not contain any auxiliary variables from $\mathcal{EC}_i$.

W.l.o.g. we can assume that $|\widehat{\mathcal{M}}_t^{-1}| \le |\mathcal{M}_t^{-1}|$ (simply leave in $\widehat{\mathcal{M}}_t^{-1}$ only those x_j^ϵ which are really used for fulfilling the two conditions in Definition 3.19). Since $\widehat{\mathcal{M}}_t^{-1}$ is $\mathcal{T}$-admissible and $|\widehat{\mathcal{M}}_t^{-1}| \le |\mathcal{M}_t^{-1}| \le |\mathcal{M}_t| < n$, arguing as in the proof of Theorem 3.13, we can find some $x_j^\epsilon \in C_i$ such that $\widehat{\mathcal{M}}_t^{-1} \cup \{x_j^\epsilon\}$ is still $\mathcal{T}$-admissible. Arguing as in Case 1 above, we can extend $\mathcal{M}_t^{-1}$ with either x_j^ϵ itself or with some $y_{i\ell}^{\beta_{ij\ell}}$ to get $\mathcal{M}_{t+1}^{-1}$ with $\mathcal{M}_{t+1}^{-1} \models C$.

Theorem 3.18 follows. □

3.4. Tseitin tautologies. A Tseitin tautology is an unsatisfiable CNF capturing the basic combinatorial principle that, for every graph, the sum of degrees of all vertices is even. These tautologies were originally used by Tseitin [Tse68] to present the first superpolynomial lower bounds on proof size for a certain restricted form of resolution (regular resolution).

The main theorem in this subsection is Theorem 3.24 that presents a linear lower bound on the clause space of Tseitin formulas. This result was independently obtained in [Tor99].

DEFINITION 3.20 (Tseitin formulas). *Fix G, a finite connected graph, with $|V(G)| = n$. $\sigma : V(G) \to \{0,1\}$ is said to have* odd-weight *if $\Sigma_{v \in V(G)} \sigma(v) \equiv 1 \pmod 2$. Denote by $d_G(v)$ the degree of v in G. Fix σ an odd-weight function. Assign a distinct variable x_e to each edge $e \in E(G)$. For $v \in V(G)$ define $PARITY_{v,\sigma} \stackrel{\text{def}}{=} (\bigoplus_{e \ni v} x_e \equiv \sigma(v) (\mathrm{mod}\ 2))$. The* Tseitin formula *of G and σ is*

$$T(G,\sigma) \stackrel{\text{def}}{=} \bigwedge_{v \in V(G)} PARITY_{v,\sigma}.$$

If the maximal degree of G is constant, then the initial size and width of $T(G,\sigma)$ are small as well.

LEMMA 3.21. *If d is the maximal degree of G, then $T(G,\sigma)$ is a d-CNF with at most $n \cdot 2^{d-1}$ clauses and at most $nd/2$ variables.*

We shall need the following lemma from [Urq95].

LEMMA 3.22 (see [Urq95]). *If G is connected, then $T(G,\sigma)$ is contradictory iff σ is an odd-weight function. Moreover, for any $v \in V(G)$ there is an assignment satisfying all axioms from $\{ PARITY_{u,\sigma} \mid u \neq v \}$.*

The space lower bound on Tseitin formulas will be directly connected to the following notion of expansion.

DEFINITION 3.23 (connectivity expansion). *For G, a connected graph on n vertices, let $c(G)$ be the minimal number of edges that one must remove from G in order to obtain a graph in which all connected components have size $\leq n/2$.*

THEOREM 3.24. *For G a graph of maximal degree d,*

$$CSpace^{sem}(T(G,\sigma)) > c(G) - d.$$

Proof. Our invariant will be a specially tailored sequence of restrictions, defined hereby.

DEFINITION 3.25 (admissible configurations for $T(G,\sigma)$). *Suppose that $\mathcal{M}$ is a proper 1-CNF with $|\mathcal{M}| < c(G)$. Let $E(\mathcal{M}) \subseteq E(G)$ be the subset of edges corresponding to the variables of $\mathcal{M}$. Then $G \setminus E(\mathcal{M})$ (the graph G after removing the edges $E(\mathcal{M})$) has a uniquely defined maximal connected component $V_{max}(\mathcal{M})$ with $|V_{max}(\mathcal{M})| > n/2$.*

We call the proper 1-CNF $\mathcal{M}$ with $|\mathcal{M}| < c(G)$ admissible *for $T(G,\sigma)$ if there exists a proper 1-CNF $\mathcal{M}'$ such that $\mathcal{M}' \supseteq \mathcal{M}$ and $\mathcal{M}' \models PARITY_{v,\sigma}$ for any $v \notin V_{max}(\mathcal{M})$. (In other words, $\mathcal{M}$ is consistent with $\{ PARITY_{v,\sigma} \mid v \notin V_{max}(\mathcal{M}) \}$.)*

Remark. It is important to notice the following *monotonicity property*: if $\mathcal{M}$ is admissible and $\mathcal{M}' \subseteq \mathcal{M}$, then $V_{\max}(\mathcal{M}') \supseteq V_{\max}(\mathcal{M})$ and $\mathcal{M}'$ is admissible, too. That is exactly what we need the condition $|\mathcal{M}| < c(G)$ for.

LEMMA 3.26. *Suppose that $\mathcal{M}$ is admissible for $T(G,\sigma)$. Then for any $v_0 \in V_{max}(\mathcal{M})$ there exists a proper 1-CNF $\mathcal{M}' \supseteq \mathcal{M}$ such that for all $v \neq v_0$ $\mathcal{M}' \models PARITY_{v,\sigma}$. (In other words we can extend the assignment $\mathcal{M}$ to satisfy all axioms except that of v_0.)*

Proof. Let us consider the restriction ρ which corresponds to $\mathcal{M}$ ($\rho(x_e) = \epsilon$ iff $x_e^\epsilon \in \mathcal{M}$). If we apply this restriction to the tautology $T(G,\sigma)$ it will be partitioned to the independent formulas: $T^i = \bigwedge_{v \in V_i} PARITY_{v,\sigma'}$ for different connected components V_i of the graph $(G \setminus E(\mathcal{M}))$, where

$$\sigma'(v) \stackrel{\text{def}}{=} \sigma(v) \oplus \bigoplus_{\substack{e \ni v \\ \rho(x_e) \neq *}} \rho(x_e).$$

$V_{\max}(\mathcal{M})$ is the component with maximal size. By the definition of admissible configurations, all T^i are satisfiable for $V_i \neq V_{\max}(\mathcal{M})$. By Lemma 3.22 there exists an assignment to the edges in $V_{\max}(\mathcal{M})$ which satisfies all axioms except $PARITY_{v_0,\sigma'}$. The lemma follows. □

Now let us finish the proof of Theorem 3.24. Let $\mathbb{A}$ be the set of configurations admissible for $T(G,\sigma)$. As usual we claim that for each configuration $\mathcal{M}$, derivable in space $c(G)-d$, there exists a configuration $\mathcal{M}^{-1} \in \mathbb{A}$ such that $\mathcal{M}^{-1} \models \mathcal{M}$ and $|\mathcal{M}^{-1}| \leq |\mathcal{M}|$. We prove it by induction. The basis, inference step, and memory erasing can be handled with the help of the locality lemma, Lemma 3.9, just as in Theorems 3.13 and 3.18.

Consider the axiom download. Let $\mathcal{M}_{t+1} := \mathcal{M}_t \cup \{C\},\ C \in PARITY_{v,\sigma}$ for some vertex v. The proof splits into two cases.

Case 1. $v \notin V_{\max}(\mathcal{M}_t^{-1})$. Since $\mathcal{M}_t^{-1}$ is admissible, there exists $\mathcal{M}_t' \supseteq \mathcal{M}_t^{-1}$ such that for all $v \notin V_{\max}(\mathcal{M}_t^{-1})$, $\mathcal{M}_t' \models PARITY_{v,\sigma}$ (and in particular $\mathcal{M}_t' \models C$). Let $x_e^\epsilon \in \mathcal{M}_t'$ be a literal which forces C to true (i.e., $x_e^\epsilon \in C$). Let $\mathcal{M}_{t+1}^{-1} = \mathcal{M}_t^{-1} \cup \{x_e^\epsilon\}$. It is clear that $V_{\max}(\mathcal{M}_{t+1}^{-1}) = V_{\max}(\mathcal{M}_t^{-1})$ (since $e \ni v$) and $\mathcal{M}_{t+1}^{-1} \in \mathbb{A}$.

Case 2. $v \in V_{\max}(\mathcal{M}_t^{-1})$. Let us add to $E(\mathcal{M}_t^{-1})$ all the edges adjacent to v: $E' = E(\mathcal{M}_t^{-1}) \cup E(v)$. By the induction hypothesis, $|\mathcal{M}_t^{-1}| \leq c(G)-d-1$ (there is one free memory cell for axiom download); hence $|E'| < c(G)$. Let $V'_{\max}$ be the maximal connected component in $G \setminus E'$. By our remark on the monotonicity, $V'_{\max} \subseteq V_{\max}(\mathcal{M}_t^{-1})$. Fix any $v_0 \in V'_{\max}$ and let $\mathcal{M}'$ be the proper extension of $\mathcal{M}_t^{-1}$ from Lemma 3.26 such that for all $u \neq v_0$ $\mathcal{M}' \models PARITY_{u,\sigma}$. Let $x_e^\epsilon \in \mathcal{M}'$ be a literal which forces C to true (i.e., $x_e^\epsilon \in C$). Let $\mathcal{M}_{t+1}^{-1} = \mathcal{M}_t^{-1} \cup \{x_e^\epsilon\}$. It is clear that $V_{\max}(\mathcal{M}_{t+1}^{-1}) \supseteq V'_{\max} \ni v_0$; hence $\mathcal{M}_{t+1}^{-1} \in \mathbb{A}$.

Theorem 3.24 follows. □

If G is an expander, then the clause space of refuting $T(G,\sigma)$ is linear in the input size, and we get the following.

COROLLARY 3.27. *There exist arbitrarily large unsatisfiable* 3*-CNF formulas* $\mathcal{T}$ *with*

$$CSpace^{sem}(\mathcal{T}) = \Omega(|\mathcal{T}|).$$

Proof. Let G be a 3-regular expander with expansion $\epsilon > 0$ (i.e., for all $(V' \subset V)(|V'| \leq n/2 \Rightarrow |E(G) \cap (V' \times (V \setminus V'))| \geq \epsilon|V'|)$). Let σ be an odd-weight function on $V(G)$. Let $\mathcal{T} = T(G,\sigma)$. By Lemma 3.21, $\mathcal{T}$ is an unsatisfiable 3-CNF formula with $O(|V|)$ clauses and variables. On the other hand, it is easy to verify that $c(G) = \Omega(|V|)$. □

4. Clause space in PC. In this section we give definitions of the clause space in the stronger proof system called *polynomial calculus* (PC). We then show that the proof techniques of the previous section are not good enough for PC by proving an upper bound of $\frac{2}{3}n+O(1)$ on the clause space of CT_n. Finally, we generalize our proof systems to *multivalued logic*, since our lower bounds will be presented more naturally in this setting. The lower bound itself appears in the next section.

4.1. PC. In PC we work within a fixed field $\mathbb{F}$. A line in a PC derivation is a polynomial over $\mathbb{F}$, represented as a sum of monomials.[4] With every polynomial

[4]Since all our results apply to any field $\mathbb{F}$, we do not mention it in our future definitions and statements and simply assume $\mathbb{F}$ is some fixed field.

$P(x_1, \ldots, x_n)$ we associate the Boolean function $||P||$, called the *interpretation* of P, which is the characteristic function of the set of its roots in $\{0,1\}^n$, i.e.,

$$||P||(\alpha) = 1 \text{ iff } P(\alpha) = 0.$$

For $\mathcal{P}$ a set of polynomials, $||\mathcal{P}|| \stackrel{\text{def}}{=} \{||P|| : P \in \mathcal{P}\}$.

Notation. Throughout this section we sometimes identify a polynomial P with its interpretation $||P||$ as a Boolean function, and it will be clear from the context which of the two we mean. In particular, for $\mathcal{P}$ a set of polynomials, and $\mathcal{M}$ a set of Boolean functions, we will use the notation $\mathcal{M} \models \mathcal{P}$, ($\mathcal{P} \models \mathcal{M}$, resp.) to denote $\mathcal{M} \models ||\mathcal{P}||$ ($||\mathcal{P}|| \models \mathcal{M}$, resp.).

The natural representation of the clause $\bigvee_{i=1}^n x_i$ as a polynomial is by $\prod_{i=1}^n (1 - x_i)$. Notice that this polynomial has 2^n monomials, and, since we will define the clause space of a polynomial to be the number of monomials appearing in it, this representation makes PC much weaker than resolution. For this reason we use an augmented version of PC that is strictly stronger than resolution, with respect to space as well as size. We denote this augmented system by PCR (*polynomial calculus augmented with resolution*). In this system, we introduce a distinct *variable* for every *literal* and add some axioms forcing x_i and $\bar{x}_i$ to have distinct values in $\{0,1\}$. Thus, a line in PCR is a polynomial over the variable set $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots$, and the following inference rules are used:

DEFAULT AXIOMS. $x(1-x)$ for all variables x (forcing $\{0,1\}$ solutions), and $x + \bar{x} = 1$ for all pairs of distinct variables $x, \bar{x}$ (forcing x to be the negation of $\bar{x}$).

SCALAR ADDITION. Derive $\alpha P_1 + \beta P_2$ from P_1, P_2, for α, β scalars in $\mathbb{F}$, and for P_1, P_2 any polynomials over $\mathbb{F}$.

VARIABLE MULTIPLICATION. Derive $x \cdot P$ from P, for x any variable, and for P any polynomial over $\mathbb{F}$.

A PCR derivation of a polynomial Q from a set of polynomials $\mathcal{P} = \{P_1, \ldots, P_m\}$ is a sequence of configurations $\pi = \{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_s\}$, where $\mathcal{M}_0 = \emptyset$, $\mathcal{M}_s = \{Q\}$, and, for $1 \leq t \leq s$, $\mathcal{M}_t$ must be one of the following:

Axiom download. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup \{P\}$ for some axiom $P \in \mathcal{P}$ or some default axiom P.

Memory erasing. $\mathcal{M}_t := \mathcal{M}_{t-1} - \mathcal{M}'$ for some subset of polynomials $\mathcal{M}' \subseteq \mathcal{M}_{t-1}$.

Inference. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup \{P\}$ for P inferred from some subset $\mathcal{M}' \subseteq \mathcal{M}_{t-1}$ by a single application of scalar addition or variable multiplication rules.

Just like in resolution, our lower bounds will apply to the *semantical* version of PC. In this system, we replace the inference step of the above definition with the following semantical inference step:

Semantical inference. $\mathcal{M}_t := \mathcal{M}_{t-1} \cup \{P\}$ for P such that $||\mathcal{M}_{t-1}|| \models ||P||$.

The clause and variable space of a PCR derivation are the natural analogues of the same measures in resolution. Namely, a *configuration* is a set of polynomials, and its clause space is the number of *distinct* monomials that appear in the polynomials. (Notice that a certain monomial may appear in several polynomials, but we count it only once.) The variable space of a configuration is the sum of degrees of these distinct monomials.

A *refutation of* $\mathcal{P}$ is a derivation of the polynomial 1 from $\mathcal{P}$. The *clause space of a derivation* is the maximal clause space of a configuration in it, and the clause space of $\mathcal{P}$, denoted $MSpace(\mathcal{P})$, is the minimal clause space of a refutation of $\mathcal{P}$, if it exists, and ∞ otherwise. (We use the letter M, for *monomial space*, to distinguish it from resolution clause space.) The variable space of $\mathcal{P}$, denoted $VSpace_{PCR}(\mathcal{P})$, and the semantical clause space of a PCR proof, denoted $MSpace^{sem}(\mathcal{P})$, are analogously defined.

Remark. A polynomial is a linear combination of clauses. It is worth noting that the clause space lower bounds of section 4.4 apply to any sound calculus that uses lines that are *arbitrary* Boolean functions of clauses. (The linearity, however, will be important in section 5 for the *variable* space lower bounds.) The strongest such system is *functional calculus* (FC), which has as lines any Boolean function over clauses and has the single semantical inference rule that allows us to derive any function g from $\mathcal{M}$ whenever $\mathcal{M} \models g$. A definition of this system and its clause space appear in the appendix.

We end this section by noting that PCR is at least as efficient as resolution with respect to variable and clause space. The proof of the following lemma follows from the fact that PCR can efficiently simulate a resolution derivation.

LEMMA 4.1. *For $\mathcal{T}$, a contradictory set of clauses, and $\mathcal{P}$, its natural formulation as a set of polynomials,*

- $MSpace(\mathcal{P}) \leq CSpace(\mathcal{T}) + O(1)$;
- $VSpace_{PCR}(\mathcal{P}) \leq 2 \cdot VSpace_R(\mathcal{T})$.

4.2. PCR upper bounds for CT_n. We now show that $MSpace(CT_n)$ is substantially smaller than $CSpace^{sem}(CT_n)$. This result shows the necessity of using more complicated techniques than those in section 3 to prove PCR lower bounds. (Recall that $CSpace^{sem}(CT_n) = n$.)

THEOREM 4.2. $MSpace(CT_n) \leq 2n/3 + 6$.

Proof.

DEFINITION 4.3. *We say that a PCR proof π has k* temporary memory cells *if whenever the transition $\mathcal{M}_{t-1} \rightsquigarrow \mathcal{M}_t$ is an axiom download, then $MSpace(\mathcal{M}_{t-1}) \leq MSpace(\pi) - k$. (Informally, distinct monomials appearing in the current configuration $\mathcal{M}_t$ are stored in distinct "memory cells." The definition says that whenever we start downloading an axiom, there exist at least k (out of $MSpace(\pi)$) free memory cells.)*

We prove by induction the following claim saying that if we can refute CT_n in small space, we can also refute $CT_{\ell \cdot n}$ in small space.

CLAIM 4.4 (amplification). *If there exists a PCR refutation π of CT_n with*

$$MSpace(\pi) \leq s + k$$

such that π has k temporary cells, then for all integers $\ell \geq 1$

$$MSpace(CT_{\ell \cdot n}) \leq \ell \cdot s + k.$$

Moreover, the corresponding refutation also has k temporary memory cells.

Proof. The proof is by induction on $\ell \geq 1$. For the induction step, suppose π_ℓ is a refutation of $CT_{\ell \cdot n}$ with $MSpace(\pi_\ell) \leq \ell \cdot s + k$ which has k temporary cells. We shall use π_ℓ to refute $CT_{(\ell+1) \cdot n}$. We have only to show how to derive an axiom $A \in CT_{\ell \cdot n}$ from $CT_{(\ell+1) \cdot n}$ using small memory because, if we can derive any axiom of $CT_{\ell \cdot n}$ in small space, we can use π_ℓ directly, replacing each axiom download in π_ℓ with our small space derivation.

Suppose the tth step of π_ℓ is an axiom download of $A \in CT_{\ell \cdot n}$. $A = \bigvee_{j=1}^{\ell n} x_j^{\epsilon_j}$ for some $\vec{\epsilon} = \{\epsilon_1, \ldots, \epsilon_{\ell n}\} \in \{0,1\}^{\ell n}$. By the induction hypothesis, $MSpace(\mathcal{M}_{t-1}) \leq \ell \cdot s$ because $\mathcal{M}_{t-1}$ has k empty memory cells. We use these empty memory cells and additional s memory cells to derive A from all axioms $B \in CT_{(\ell+1)\cdot n}$ that agree with A on all variables of A (i.e., $B = A \vee \bigvee_{j=\ell n+1}^{(\ell+1)n} x_j^{\epsilon_j}$ for some $\epsilon_{\ell n+1}, \ldots, \epsilon_{(\ell+1)n} \in \{0,1\}$). The total memory of the new proof is $MSpace(CT_{\ell \cdot n}) + s$. It also has k temporary memory cells, since it downloads axioms only during the emulation of the proof of CT_n. □

To prove Theorem 4.2 it is sufficient to produce a PCR refutation of CT_3 with clause space six and four temporary memory cells ($s = 2$, $k = 4$). At the beginning of this refutation we infer the configuration $\{x_1, x_2\}$ in the obvious way. After that the proof proceeds as follows (we omit straightforward intermediate transitions):

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightsquigarrow \begin{pmatrix} x_1 \\ x_2 \\ \text{Axiom: } \bar{x}_1\bar{x}_2x_3 \end{pmatrix} \rightsquigarrow \begin{pmatrix} x_1 \\ x_2 \\ \bar{x}_2x_3 \end{pmatrix} \rightsquigarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$
$$\rightsquigarrow \begin{pmatrix} 1-\bar{x}_1 \\ 1-\bar{x}_2 \\ 1-\bar{x}_3 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1-\bar{x}_1 \\ \bar{x}_1-\bar{x}_1\bar{x}_2 \\ \bar{x}_1\bar{x}_2-\bar{x}_1\bar{x}_2\bar{x}_3 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1-\bar{x}_1\bar{x}_2\bar{x}_3 \end{pmatrix}$$
$$\rightsquigarrow \begin{pmatrix} 1-\bar{x}_1\bar{x}_2\bar{x}_3 \\ \text{Axiom: } \bar{x}_1\bar{x}_2\bar{x}_3 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 \end{pmatrix}.$$

All the configurations explicitly displayed here have at most four clauses. Two more temporary clauses are needed to keep intermediate polynomials. □

Remark. For the stronger system FC, we get better upper bounds than those of the previous theorem (namely, CT_n can be refuted in clause space $n/2 + 2$). For a definition of this proof system, and a proof of this and related results, see the appendix.

4.3. PCR over multivalued logic. In this subsection we extend our proof systems to work with multivalued variables. The motivation is the usual one: in many cases, the multivalued logic is a natural vista to view lower bounds for the Boolean case which is our true interest. For example, [RWY97] used R-way (read-once) branching programs to formulate and prove some partial results about the resolution (size) complexity of the weak (i.e., when $m \gg n$) pigeonhole principle PHP_n^m. Crucial to their results are two dual interpretations of PHP_n^m in terms of multivalued logic; here we are interested only in one of them, the *Column Model*. In this model, the pigeonhole principle has the following form: suppose $x_1, \ldots, x_n$ are variables of m-valued logic, where $m > n$ ("$x_j = i$" has the intended meaning "the ith pigeon sits in the jth hole"). Then there exists $i \in [m]$ which is not in the set $\{x_1, \ldots, x_n\}$.

In the proofs of our remaining lower bounds, it will be also very convenient to treat PHP_n^m in this way. In particular, this will allow us to formulate our bounds in terms of a simple, uniform, and concise criterion fulfilled by both CT_n and PHP_n^m (Definition 4.12 below).

We need to generalize some previous definitions.

DEFINITION 4.5. *Let us fix some finite domain D. Instead of Boolean variables, we use* multivalued variables *x_j ranging over the domain D. A* multivalued Boolean function *$f(x_1, \ldots, x_n)$ is a mapping from D^n to $\{0,1\}$, where, as before, we identify* 1 *with True and* 0 *with False. The notions of* a (multivalued) satisfying assignment

$a \in D^n$ and semantical implication *$f_1, \ldots, f_k \models g$ are generalized to the case of multivalued logic straightforwardly.*

Thus, the only remaining thing we still need to define is the set of allowable lines.

DEFINITION 4.6 (multivalued clauses). *Suppose that D is some finite domain. A* multivalued literal *is the formal expression x_j^π, where π is some nonconstant function $\pi : D \to \{0,1\}$.*

A multivalued clause *is a disjunction of multivalued literals corresponding to distinct variables*

$$C = x_{j_1}^{\pi_1} \vee x_{j_2}^{\pi_2} \vee \cdots \vee x_{j_w}^{\pi_w},\ k \neq l \to j_k \neq j_l,$$

with the straightforward interpretation $||C||(\alpha) \stackrel{\text{def}}{=} \pi_1(\alpha_{j_1}) \vee \cdots \vee \pi_w(\alpha_{j_w})$.

The width *of a multivalued clause is the number of multivalued literals in it.*

DEFINITION 4.7. *Denote by x_j^i the multivalued literal $x_j^{\chi_i}$, where χ_i is the characteristic function of i ($\chi_i(i') = 1$ iff $i' = i$).*

We are ready to define multivalued semantical resolution and PCR.

DEFINITION 4.8 (multivalued resolution). Multivalued semantical resolution over domain *D is the system analogous to semantical resolution which works with multivalued clauses instead of usual ones.*

The clause and variable complexity are analogous to that of semantical resolution. Denote by $CSpace_D^{sem}(\mathcal{T})$ the clause space of refuting $\mathcal{T}$ in semantical resolution over domain D.

DEFINITION 4.9 (multivalued PCR). *Suppose that $\mathbb{F}$ is a field.* Multivalued semantical PCR over domain *D is the semantical system which keeps in memory polynomials over $\mathbb{F}$ with literals x_j^π for all possible j, π as their variables. As in the Boolean case, the interpretation is given by the characteristic function of the set of roots:*

$$||P||(\alpha_1, \ldots, \alpha_n) = 1 \quad \textit{iff} \quad P[\pi(\alpha_j)/x_j^\pi] = 0$$

in the field $\mathbb{F}$. The clause and variable complexity are analogous to that of semantical PCR. Denote by $MSpace_D^{sem}(\mathcal{P})$ the clause space of refuting $\mathcal{P}$ in semantical resolution over domain D.

We conclude this subsection with one possible translation from multivalued systems to ordinary ones. This straightforward translation preserves the variable and clause space lower bounds.

DEFINITION 4.10. *Suppose that P is some multivalued proof system over domain D; $\mathcal{T}$ is a contradictory set of axioms. Replace each multivalued variable x_j with the tuple $x_{1j}, x_{2j}, \ldots, x_{mj}$, where $m = |D|$. The intuitive meaning of x_{ij} is "$x_j = i$," i.e., the same as of the multivalued literal x_j^i.*

Let us denote by $PR_D(\mathcal{T})$ the following set of axioms over Boolean variables x_{ij}:

- *$\bar{x}_{i_1 j} \vee \bar{x}_{i_2 j}$ for $i_1 \neq i_2$;*
- *$\phi\left[\bigvee_{i \in \pi^{-1}(1)} x_{ij} / x_j^\pi\right]$ for all $\phi \in \mathcal{T}$.*

It is clear that each refutation of $PR_D(\mathcal{T})$ can be transformed into the refutation of $\mathcal{T}$ without increase in either variables or clauses. Thus we have the following trivial proposition which shows how to use multivalued systems for proving Boolean lower bounds.

PROPOSITION 4.11. *For $\mathcal{T}$, a contradictory set of multivalued clauses over domain D,*

$$CSpace_D^{sem}(\mathcal{T}) \leq CSpace(PR_D(\mathcal{T})).$$

For $\mathcal{P}$, a contradictory set of polynomials over domain D,

$$MSpace_D^{sem}(\mathcal{P}) \leq MSpace(PR_D(\mathcal{P})).$$

4.4. Wide tautologies and lower bounds for PCR. In this section we define a natural class of *wide* tautologies which turn out to be hard for PCR in terms of clause complexity. In section 5 we will also show that wide tautologies are hard for PCR (and hence for resolution as well) in terms of *variable* complexity. The previous upper bound on the clause space of CT_n (Theorem 4.2) shows that selecting *one* variable per clause in the memory, and using this variable to satisfy our clause, will not work for PCR. Surprisingly, we show that selecting *two* variables per clause does the job.

DEFINITION 4.12 (multivalued wide tautology). *A family of multivalued clauses $\mathcal{T}$ over multivalued variables $x_1, \ldots, x_n$ is* wide *iff every clause in it has maximal possible width n.*

One obvious example of a (Boolean) wide family is CT_n. The second example, and our main motivation, is PHP_n^m. Namely, suppose that $|D| = m > n$, and let $\mathcal{T} \stackrel{def}{=} \{ x_1^i \vee x_2^i \vee \cdots \vee x_n^i \mid i \in [m] \}$. Then $PR_D(\mathcal{T}) = PHP_n^m$. By Proposition 4.11, lower bounds for the space complexity of this multivalued version imply lower bounds for the space complexity of the ordinary (Boolean) pigeonhole principle.

Now we show that wide tautologies are hard. The heart of our lower bounds for PCR is the locality lemma, Lemma 4.14.

DEFINITION 4.13 (proper 2-CNF's). *$\mathcal{M}$ is called a* proper 2-CNF *iff*

$$\mathcal{M} = \left\{ x_{j_1^1}^{i_1^1} \vee x_{j_1^2}^{i_1^2},\ x_{j_2^1}^{i_2^1} \vee x_{j_2^2}^{i_2^2}, \ldots, x_{j_k^1}^{i_k^1} \vee x_{j_k^2}^{i_k^2} \right\},$$

where $j_1^1, j_1^2, j_2^1, \ldots, j_k^2$ are pairwise distinct indices and $i_1^1, \ldots, i_k^2 \in [m]$ (not necessarily distinct). (In other words, the proper 2-CNF says that for every $\ell \in [k]$ either $x_{j_\ell^1} = i_\ell^1$ or $x_{j_\ell^2} = i_\ell^2$.)

LEMMA 4.14 (locality lemma for PCR). *Let $\mathcal{M}$ be a proper 2-CNF, and let $\mathcal{M}_1$ be a set of polynomials such that $\mathcal{M} \models \mathcal{M}_1$. Then there exists a proper 2-CNF $\mathcal{M}_1^{-1}$ such that $\mathcal{M}_1^{-1} \models \mathcal{M}_1$ and $|\mathcal{M}_1^{-1}| \leq 2 \cdot MSpace_D^{sem}(\mathcal{M}_1)$.*

Remark. Notice that $\mathcal{M}_1^{-1}$ is not necessarily a proper subformula of $\mathcal{M}$.

Proof. In our proof we will use the following corollary of Hall's matching theorem.

LEMMA 4.15 (Hall's theorem). *For a family of sets $V_1, \ldots, V_k$ (not necessarily distinct), if for all index sets $I \subseteq [k]$*

$$\left| \bigcup_{i \in I} V_i \right| \geq |I|,$$

then the family $V_1, \ldots, V_k$ has a system of distinct representatives. *That is, there exist $v_i \in V_i$ such that $\{v_1, \ldots, v_k\}$ are pairwise distinct.*

COROLLARY 4.16. *For a family of sets $V_1, \ldots, V_k$, if for all index sets $I \subseteq [k]$*

$$\left| \bigcup_{i \in I} V_i \right| \geq 2 \cdot |I|,$$

then we can assign to each V_i two *distinct representatives $v_{i1}, v_{i2} \in V_i$ such that all $2k$ elements $v_{11}, v_{12}, \ldots, v_{k1}, v_{k2}$ are pairwise distinct.*

Proof. Just apply Hall's matching theorem to the system

$$V_1, V_1, V_2, V_2, \ldots, V_k, V_k. \qquad \square$$

Now let us prove Lemma 4.14. Note that $||\mathcal{M}_1|| = g(C_1, \ldots, C_s)$, where $s = MSpace_D^{sem}(\mathcal{M}_1)$, $C_1, \ldots, C_s$ are multivalued clauses, and g is some Boolean function. Let us gradually construct the required proper 2-CNF $\mathcal{M}_1^{-1}$. Suppose w.l.o.g. that $\mathcal{M} = \{x^0_{1,1} \vee x^0_{1,2},\ x^0_{2,1} \vee x^0_{2,2}, \ldots, x^0_{k,1} \vee x^0_{k,2}\}$.

For a clause C denote by $V(C) \subseteq [k]$ the index set of those axioms $x^0_{j,1} \vee x^0_{j,2}$ of $\mathcal{M}$ for which C contains at least one variable $x_{j,1}, x_{j,2}$. (Formally, $V(C) \stackrel{\text{def}}{=} \left\{ j \,\middle|\, \exists \epsilon \in \{1,2\}\ \exists \pi (x^\pi_{j,\epsilon} \in C) \right\}$.) For $\Gamma \subseteq [s]$, let us denote $\bar{\Gamma} = [s] \setminus \Gamma$, $V(\Gamma) = \bigcup_{\gamma \in \Gamma} V(C_\gamma)$. Let Γ be any maximal subset $\Gamma \subseteq [s]$ with the property

$$|V(\Gamma)| \leq 2 \cdot |\Gamma|.$$

Then for any $I \subseteq \bar{\Gamma}$,

$$|V(I) \setminus V(\Gamma)| > 2 \cdot |I|.$$

(Otherwise we could add the set I to Γ, contradicting the maximality of Γ.) Hence, by Corollary 4.16, for any clause C_γ with $\gamma \in \bar{\Gamma}$, we can choose two unique *representative indices* $j_1, j_2 \in V(C_\gamma)$ such that $j_1, j_2 \notin V(\Gamma)$. Let $x_{rep_1(C_\gamma)}, x_{rep_2(C_\gamma)}$ be two corresponding *representative variables* from C_γ which lie in the intersection with axioms numbers j_1, j_2 of $\mathcal{M}$. Denote by $sat_\epsilon(C_\gamma)$ the value of $x_{rep_\epsilon(C_\gamma)}$ which forces C_γ to $True$. Such a value must exist because C_γ is a clause in which $x^\pi_{rep_\epsilon(C_\gamma)}$ appears for some *nonconstant* π (see Definition 4.6).

Let

$$\mathcal{M}_1^{-1} = \left\{ x^0_{j,1} \vee x^0_{j,2} \,|\, j \in V(\Gamma) \right\} \cup \left\{ x^{sat_1(C_\gamma)}_{rep_1(C_\gamma)} \vee x^{sat_2(C_\gamma)}_{rep_2(C_\gamma)} \,\middle|\, \gamma \in \bar{\Gamma} \right\}.$$

Clearly, $\mathcal{M}_1^{-1}$ is a proper 2-CNF. Let us estimate the size of $\mathcal{M}_1^{-1}$:

$$|\mathcal{M}_1^{-1}| = |V(\Gamma)| + |\bar{\Gamma}| \leq 2 \cdot |\Gamma| + |\bar{\Gamma}| \leq 2 \cdot s.$$

Claim 4.17. $\mathcal{M}_1^{-1} \models \mathcal{M}_1$.

Proof. We need to show that every assignment α satisfying $\mathcal{M}_1^{-1}$ satisfies $g(C_1, C_2, \ldots, C_s)$ as well. Suppose that α satisfies all the clauses of $\mathcal{M}_1^{-1}$. Notice that for all $\gamma \in \bar{\Gamma}$ $C_\gamma(\alpha) = 1$. (Since α satisfies $\mathcal{M}_1^{-1}$, it satisfies $x^{sat_1(C_\gamma)}_{rep_1(C_\gamma)} \vee x^{sat_2(C_\gamma)}_{rep_2(C_\gamma)}$, and every one of these representative variables from C_γ forces C_γ to $True$.)

We are going to show that α can be changed in such a way that it will still preserve the value of all clauses $C_1, \ldots, C_s$ (and, hence, of g) but at the same time will satisfy $\mathcal{M}$. For an axiom $x^0_{j,1} \vee x^0_{j,2}$ of $\mathcal{M}$ either $j \in V(\Gamma)$ (in this case the axiom is satisfied since it also appears in $\mathcal{M}_1^{-1}$) or we can choose the variable $x_{j,\epsilon}$ which is *not* a representative variable of any clause C_γ with $\gamma \in \bar{\Gamma}$. (Notice that different clauses cannot have representatives from the same axiom.) In the latter case we just set $x_{j,\epsilon}$ to zero. This does not change the values of clauses $C_1, \ldots, C_s$, but the axioms of $\mathcal{M}$ get satisfied.

Thus we can get the new assignment α' which satisfies $\mathcal{M}$ and such that $C_i(\alpha) = C_i(\alpha')$ for all i. Since $||\mathcal{M}|| \models g(C_1, \ldots, C_s)$, we have

$$g(C_1(\alpha), \ldots, C_s(\alpha)) = g(C_1(\alpha'), \ldots, C_s(\alpha')) = 1.$$

Lemma 4.14 follows. □

We are now ready to prove our main theorem.

THEOREM 4.18. *For any wide tautology $\mathcal{T}$ over n variables with domain D, $MSpace_D^{sem}(\mathcal{T}) \geq \frac{n}{4}$.*

Remark. The same lower bound, with the same proof, applies to the much stronger FC, in which each line is an arbitrary Boolean function and all derivations rule are purely semantical. For a definition of this system and related results see the appendix.

Proof. The proof is quite analogous to the proof of Theorem 3.13. Let us denote by $\mathbb{A}$ the set of all proper 2-CNF's of size $\leq \frac{n}{2}$. As in the previous cases we claim that for any configuration $\mathcal{M}$, achievable in space $\frac{n}{4}$, there exists a configuration $\mathcal{M}^{-1} \in \mathbb{A}$ such that $|\mathcal{M}^{-1}| \leq 2 \cdot MSpace_D^{sem}(\mathcal{M})$ and $\mathcal{M}^{-1} \models \mathcal{M}$. We prove it by induction.

The inference step can be treated with the help of the locality lemma, Lemma 4.14. Namely, if $\mathcal{M}_t \models \mathcal{M}_{t+1}$, first take $\mathcal{M}_t^{-1}$ as $\mathcal{M}_{t+1}^{-1}$, and then shrink it to the required size $2 \cdot MSpace_D^{sem}(\mathcal{M}_{t+1})$ by applying Lemma 4.14.

The axiom download is also straightforward. Suppose that $\mathcal{M}_{t+1} \leftarrow \mathcal{M}_t \cup \{C\}$, $C \in \mathcal{T}$. Take $\mathcal{M}_t^{-1}$ and choose two literals $x_{j_1}^{\pi_1}, x_{j_2}^{\pi_2} \in C$ such that x_{j_1} and x_{j_2} are not contained in $\mathcal{M}_t^{-1}$. Such a pair of literals must exist because C has width n and $|\mathcal{M}_t| < n/4$, and hence $|\mathcal{M}_t^{-1}| \leq n/2$. Then let $\mathcal{M}_{t+1}^{-1} = \mathcal{M}_t^{-1} \cup \{x_{j_1}^{i_1} \vee x_{j_2}^{i_2}\}$ for i_1, i_2 satisfying $\pi_1(i_1) = \pi_2(i_2) = 1$.

Theorem 4.18 follows. □

COROLLARY 4.19. *$MSpace^{sem}(CT_n) \geq MSpace_D^{sem}(CT_n) \geq n/4$.*

As said previously, a lower bound on multivalued logic is also a lower bound on two-valued Boolean logic, and thus we get the following.

COROLLARY 4.20. *For all $m > n$,*

$$MSpace^{sem}(PHP_n^m) \geq MSpace_D^{sem}(PHP_n^m) \geq n/4.$$

5. Variable complexity for PCR. The variable space is a very natural space measure as it is tightly connected to the bit space, which is the actual number of bits needed to write down a memory configuration. If each variable index is written in binary notation, then the bit space is linear in $\log n$ times the variable space. As follows from Theorem 3.5, every (Boolean) proof system which simulates resolution has variable space upper bounded by n^2. In this section we show that any PCR proof (over an arbitrary field) of any wide tautology requires variable space $\Omega(n^2)$. This bound is tight for both CT_n and PHP_n^m.

THEOREM 5.1. *For any wide tautology $\mathcal{T}$ over n variables with domain D and any ground field $\mathbb{F}$, $VSpace_{PCR}^{sem}(\mathcal{T}) = \Omega(n^2)$.*

To understand the intuition of the proof, and to see how heavily it depends on our previous clause space lower bounds, we prove first the following (easy) special case of our theorem.

LEMMA 5.2. *$VSpace_R^{sem}(CT_n) \geq \frac{1}{4}n^2$.*

Proof. By Theorem 3.13, $CSpace^{sem}(CT_n) > n$. Let $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_s\}$ be a semantical resolution refutation of CT_n, and let t be the first time there is some clause $C \in \mathcal{M}_t$ of width $\lceil n/2 \rceil$. Such a t must exist because, by the definition of the resolution rule, the width of a clause may decrease by at most 1 in every step, all axioms have width n, and the empty clause has width 0. Let ρ be the minimal size restriction that sets C to 0. $|\rho| = |C| = \lceil n/2 \rceil$, and it is easy to see that $CT_n|_\rho$ is simply $CT_{\lfloor n/2 \rfloor}$, and $\pi|_\rho$ is a legitimate proof of it. Applying Theorem 3.13 once again, $CSpace^{sem}(CT_{\lfloor n/2 \rfloor}) > \lfloor n/2 \rfloor$, meaning there is some $t' < t$ such that $|\mathcal{M}_{t'}| > \lfloor n/2 \rfloor$.

By the definition of t, all clauses of $\mathcal{M}_{t'}$ have width larger than $\lceil n/2 \rceil$, and the lemma is proved. □

Proof of Theorem 5.1. First we need to define some notation. A term $x_{j_1}^{\pi_1} \cdot \ldots \cdot x_{j_d}^{\pi_d}$ is *multilinear* if all $j_1, \ldots, j_d$ are pairwise distinct. A *multilinear monomial* is an expression $a \cdot t$, where $a \in F^*$ and t is a multilinear term. Finally, a *multilinear polynomial* is a sum of multilinear monomials. Since we are interested only in the semantical version of PCR, and because of the identity $x_j^{\pi} \cdot x_j^{\pi'} = x_j^{(\pi\pi')}$, we will assume w.l.o.g. that all our multivalued terms, monomials, and polynomials are multilinear. Also, throughout the proof of Theorem 5.1 all the terms, monomials, and polynomials are multivalued; thus we omit this word, too. For a term t, let $\mathrm{Supp}(t)$ be the set of all j for which x_j^{π} appears in t for some π. For a polynomial P, $\mathrm{Supp}(P) \stackrel{\mathrm{def}}{=} \bigcup_{t \in P} \mathrm{Supp}(t)$.

There is a natural correspondence between subsets of the set of terms of a polynomial P and its subpolynomials; thus we sometimes write $t \in P$ when the term t is contained in P, and $P_1 \subseteq P$ which means that P_1 is a subpolynomial of P. $|P|$ is the number of terms in P.

For $\rho : \{x_1, \ldots, x_n\} \to D \cup \{*\}$ a restriction and P a polynomial, $\rho(P)$ is the polynomial produced after the substitution of the literals x_j^{π}, $j \in \rho^{-1}(D)$ with $\pi(\rho(x_j))$ and cancellation of terms. Notice that there can be many different polynomials corresponding to the same multivalued function $f : D^n \to \{0, 1\}$. We say that such polynomials are *semantically equivalent* and write $P_1 \sim P_2$. For example, $x \sim 1 - \bar{x}$ in case of Boolean PCR.

Our proof will consist of several stages. The heart of it will be the following construction based on Hall's theorem. (We already used a similar idea in proving Theorem 4.18.)

LEMMA 5.3 (matching lemma). *Suppose that P is a polynomial and t is some term (not necessarily from P). Then there exists a subpolynomial $\Gamma \subseteq P$ and a restriction ρ such that ρ does not assign the variables of $\mathrm{Supp}(t) \cup \mathrm{Supp}(\Gamma)$ and ρ maps all terms of $P \setminus \Gamma$ to zero. (Thus, in particular, $\rho(t) = t$ and $\rho(P) = \Gamma$.) Moreover, $|\mathrm{Supp}(t) \cup \mathrm{Supp}(\Gamma)| + |\rho| \le |P| + \deg t$.*

Proof. Let Γ be any maximal subpolynomial of P with the property

$$|\mathrm{Supp}(t) \cup \mathrm{Supp}(\Gamma)| \le |\Gamma| + \deg t.$$

Denote $V = \mathrm{Supp}(t) \cup \mathrm{Supp}(\Gamma)$. Then for any subset of terms $S \subseteq P \setminus \Gamma$, $|\mathrm{Supp}(S) \setminus V| > |S|$. (Otherwise we could add S to Γ and Γ would not be maximal.) Thus by Hall's theorem there exists a matching

$$\mu : [\text{terms of } P \setminus \Gamma] \to \mathrm{Supp}(P) \setminus V$$

which gives each term from $P \setminus \Gamma$ its *unique representative* from $\mathrm{Supp}(P) \setminus V$. Now we define the restriction ρ as follows. It maps these unique representatives to the values which force the corresponding terms to 0. Thus ρ does not touch the variables from V and $|\mathrm{Supp}(t) \cup \mathrm{Supp}(\Gamma)| + |\rho| \le \deg t + |\Gamma| + |P \setminus \Gamma| = \deg t + |P|$. □

Now let us prove Theorem 5.1. Suppose that we have some PCR refutation $\{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_s\}$ of $\mathcal{T}$. W.l.o.g. we can assume that all polynomials in the memory do not contain nonzero subpolynomials semantically equivalent to zero. (There is no sense in keeping such subpolynomials.) Let us fix the first moment q when some polynomial from $\mathcal{M}_q$ contains a (multilinear!) term with degree less than or equal to $\frac{n}{2}$. Let t be any such term which has the smallest possible degree (in particular, $\deg t \le \frac{n}{2}$), and let $P \in \mathcal{M}_q$ be any polynomial which contains t. Our proof splits into two cases.

Case 1. $\deg t \geq \frac{n}{4}$.

In this case we can assume that $|P| \leq \frac{n}{8}$ since otherwise $VS(P)$ is already greater than $\frac{n^2}{32}$ (recall that t is the smallest degree monomial from $\mathcal{M}_q$), and we are done.

LEMMA 5.4. *Suppose that P is a polynomial with no subpolynomials semantically equivalent to zero and that t is a term of P. Then there exists a restriction ρ which forces P to a nonzero constant c from F^* and assigns at most $|\rho| \leq \deg t + |P|$ variables.*

Proof. We apply our matching lemma, Lemma 5.3, to the polynomial P and the term t. We get the subpolynomial Γ and the restriction ρ which kills all the monomials from $P \setminus \Gamma$.

Notice that $\Gamma \neq \emptyset$ since it must contain t. Thus by our assumption, $\Gamma \not\sim 0$, and we can choose an assignment α to $\mathrm{Supp}(\Gamma)$ which maps it to some constant $c \in F^*$. Let us extend our restriction ρ to the restriction ρ' by α (this is possible because ρ and α assign to disjoint sets of variables). Then $\rho'(P) = c$ and $|\rho'| \leq |\rho| + |\mathrm{Supp}(\Gamma)| \leq |P| + \deg t$. □

The rest of Case 1 is simple. Hitting the first $(q+1)$ lines of the original refutation with the restriction ρ from Lemma 5.4, we get a new valid refutation $\{\rho(\mathcal{M}_0), \rho(\mathcal{M}_1), \ldots, \rho(\mathcal{M}_q)\}$ of the principle $\rho(\mathcal{T})$ (since $c \in \rho(\mathcal{M}_q)$ and $c \neq 0$).

Notice that since $|\rho| \leq \deg t + |P|$ we have $|\rho| \leq \frac{5n}{8}$, so $\rho(\mathcal{T})$ is a wide tautology over $\geq \frac{3n}{8}$ variables. Thus by Theorem 4.18 there exists $j < q$ such that $\rho(\mathcal{M}_j)$ (and, hence, $\mathcal{M}_j$) contains at least $\frac{3n}{32}$ monomials. Each monomial of $\mathcal{M}_j$ had degree $> \frac{n}{2}$ before applying ρ'; therefore $VS(\mathcal{M}_j) \geq \frac{3n^2}{64}$.

Case 2. $\deg t \leq \frac{n}{4}$.

DEFINITION 5.5. *We say that a polynomial P is* d-minimal *iff it does not contain a nonzero subpolynomial semantically equivalent to a polynomial of degree $\leq d$.*

Since $t \in P$, P is not $\frac{n}{4}$-minimal. Let us represent $P = P_0 + P_1$, where P_1 is $\frac{n}{4}$-minimal and $P_0 \sim P_0'$ with $\deg P_0' \leq \frac{n}{4}$. (We can construct such a representation by consecutively moving subpolynomials semantically equivalent to polynomials of degree $\leq \frac{n}{4}$ from P_1 to P_0.)

P does not contain nonzero subpolynomials semantically equivalent to zero; in particular, $P_0 \not\sim 0$. However, there still can be several nonzero polynomials $P_0' \sim P_0$ with $\deg P_0' \leq \frac{n}{4}$. Let P_0' be the polynomial of the smallest degree semantically equivalent to P_0. If there are several such polynomials we choose arbitrarily one with the smallest number of monomials of highest degree. Let s be some maximal-degree term of P_0'.

Now let us apply our matching lemma, Lemma 5.3, to the polynomial P_1 and the term s. It will yield $\Gamma \subseteq P_1$ and the restriction ρ which kills the terms from $P_1 \setminus \Gamma$.

All terms in P_1 are of degree $\geq \frac{n}{4}$; therefore, similarly to Case 1, we can assume that $|P_1| \leq \frac{n}{8}$. Hence, $|\mathrm{Supp}(s) \cup \mathrm{Supp}(\Gamma)| + |\rho| \leq \deg s + |P_1| \leq \frac{3n}{8}$.

Now we use the fact that $\mathcal{M}_q$ is the *first* configuration when a term t of degree $\leq \frac{n}{2}$ appears. It is clear that the step $\mathcal{M}_{q-1} \rightsquigarrow \mathcal{M}_q$ is a semantical inference. (It cannot be an axiom download because all axioms have degree n). Notice that all terms of polynomials from $\mathcal{M}_{q-1}$ have degree greater than $\frac{n}{2}$. Assume also that $CS(\mathcal{M}_{q-1}) \leq \frac{n}{8}$. (Otherwise $VS(\mathcal{M}_{q-1}) \geq \frac{n^2}{16}$.) Now we are going to arrive at a contradiction from all these assumptions.

First we claim that ρ can be extended to a restriction ρ_1 that does not assign the variables in $|\mathrm{Supp}(s) \cup \mathrm{Supp}(\Gamma)|$ and kills (= sets to zero) all the terms of $\mathcal{M}_{q-1}$. To see this notice that ρ has assigned $|\rho|$ variables, and we should keep unassigned $|\mathrm{Supp}(s) \cup \mathrm{Supp}(\Gamma)|$ variables; their sum is $|\rho| + |\mathrm{Supp}(s) \cup \mathrm{Supp}(\Gamma)| \leq |P_1| + \deg s \leq \frac{3n}{8}$,

and we need to kill at most $\frac{n}{8}$ terms in $\mathcal{M}_{q-1}$ of degree $\geq n/2$ each. We consecutively kill terms $t \in \mathcal{M}_{q-1}$ by choosing a free unassigned variable from $\mathrm{Supp}(t) \setminus (\mathrm{Supp}(s) \cup \mathrm{Supp}(\Gamma) \cup \rho^{-1}(D))$.

Since $\mathcal{M}_{q-1} \models P$ we have $\rho_1(P) \sim 0$. Thus $\rho_1(P_0 + P_1) \sim 0$ and $\rho_1(P_0' + P_1) \sim 0$ and $\rho_1(P_0') + \Gamma \sim 0$ (because $\rho(P_1 - \Gamma) = 0$, and ρ_1 does not touch variables from $\mathrm{Supp}(\Gamma)$). Since $\deg(\rho_1(P_0')) \leq \deg(P_0') \leq n/4$ and P_1 is $n/4$-minimal, Γ should in fact be identically zero.

We proved that $\rho_1(P_0') \sim 0$. Let $d = \deg s = \deg(P_0')$. Notice that every term of degree d in $\rho_1(P_0')$ is also contained in P_0' (because a restriction either decreases the degree of a term, or kills it, or does not change it at all). Additionally, $\rho_1(P_0')$ contains s because $\rho_1(s) = s$. Thus we get a polynomial $P_0'' = P_0' - \rho_1(P_0')$ such that $P_0'' \sim P_0$, $\deg(P_0'') \leq d$, and P_0'' contains fewer terms of degree d than P_0'. This contradicts our choice of P_0' as the polynomial of smallest degree with smallest number of monomials of highest degree. Thus this situation cannot take place. Theorem 5.1 follows.

As direct corollaries, we obtain the following tight bounds.

COROLLARY 5.6. $VSpace_R(CT_n) \geq VSpace^{sem}_{PCR}(CT_n) \geq \Omega(n^2)$.

COROLLARY 5.7. $VSpace_R(PHP^m_n) \geq VSpace^{sem}_{PCR}(PHP^m_n) \geq \Omega(n^2)$.

6. Upper bounds for Frege systems. In this section we show that the variable space complexity of CT_n is upper bounded by $O(n)$ for Frege systems. It will imply several nice corollaries and in particular the equivalence of semantical and syntactical versions for Frege proofs. We start by defining Frege proof systems and the variable space measure for them.

DEFINITION 6.1 (Frege proof systems). *A* Frege proof system *works with arbitrary propositional formulas. A* line *in a derivation is an arbitrary formula φ over some complete basis. Every inference rule is specified by a* scheme

$$\frac{A_1, \ldots, A_k}{B},$$

where A_i, B are propositional formulas, and altogether there are only finitely many of them. A formula ψ is derived from formulas $\varphi_1, \ldots, \varphi_k$ using this inference rule if there is a set of substitutions σ of formulas for the variables appearing in the scheme such that $\varphi_i = A_i^\sigma$ for $i = 1, \ldots, k$ and $\psi = B^\sigma$. We use the notation $\varphi_1, \ldots, \varphi_k \vdash \psi$ to denote that ψ was derived from $\varphi_1, \ldots, \varphi_k$ using a single rule.

The notions of a Frege proof (derivation) and refutation are analogous to those of resolution and PC. The notion of a semantical Frege proof is the natural extension of semantical systems to Frege. Finally, we give the definition of variable space of a Frege proof.

DEFINITION 6.2 (Frege variable space). *For $\mathcal{M}$, a configuration, the* variable space of $\mathcal{M}$ *is $VSpace_F(\mathcal{M}) \stackrel{\text{def}}{=} \sum_{\phi \in \mathcal{M}} VSpace_F(\phi)$, where $VSpace_F(\phi)$ is the number of occurrences of variables in ϕ. The* Frege variable space of a set of configurations $\pi = \{\mathcal{M}_0, \ldots, \mathcal{M}_s\}$ *is*

$$VSpace_F(\pi) \stackrel{\text{def}}{=} \max\{VSpace_F(\mathcal{M}_i) : i \in [s]\},$$

and the Frege variable space *of a CNF $\mathcal{T}$ is*

$$VSpace_F(\mathcal{T}) \stackrel{\text{def}}{=} \min\{VSpace_F(\pi)\},$$

where the minimum is taken over all Frege refutations π of $\mathcal{T}$.

To prove our main result we essentially use the compact rule of representing information analogous to Horner's scheme for quick evaluation of the polynomial P at a given point x:

$$P(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \cdots)).$$

Let us begin with the case when the language of our Frege system $\mathcal{F}$ consists of the standard connectives $\neg, \wedge, \vee, \rightarrow$ and the constant $\mathbf{0}$. (At the end of the section we will show how to modify our proofs to embrace the case of Frege systems in other languages.) It is easy to see that the standard simulation of one Frege system by another Frege system in the same language [CR79] also preserves variable space up to a constant multiplicative factor. Therefore, in the following theorem (which is our main result about variable space for Frege) we can unambiguously use the notation $VSpace_F(\mathcal{T})$, and we can assume in its proof that $\mathcal{F}$ contains any prescribed finite set of sound inference rules.

THEOREM 6.3. $VSpace_F(CT_n) = O(n)$.

Proof. We describe an algorithm for refuting CT_n. Let us call a proof *k-linear* if every line in the proof is derived from k axioms and at most one line that is not an axiom. A k-linear proof gives rise to a proof DAG that looks like a line, and hence the name. Clearly, if every formula in a k-linear proof has small variable space, then so does the whole proof. We shall present a 2-linear proof of CT_n such that all intermediate lines have $O(n)$ variable space.

To this end we define the sequence of read-once formulas $\varphi_0, \ldots, \varphi_{2^{(n-1)}-1}$ such that $\varphi_0 = (x_1 \vee x_2 \vee \cdots \vee x_n)$ and $\varphi_{2^{(n-1)}-1} = (x_1 \wedge x_2 \wedge \cdots \wedge x_n)$ and show how to infer φ_{t+1} from φ_t within $O(n)$ variable space using only two initial clauses from CT_n. The line φ_t encodes in linear space the following claim: *Any satisfying assignment for CT_n, when viewed as a number in binary representation, must be greater than $2 \cdot t$.*

Fix $0 \leq t < 2^{n-1}$, and let $a_1 a_2 \ldots a_{n-1}$ be its binary representation. We let

$$\varphi_t \stackrel{\text{def}}{=} x_1 *_1 (x_2 *_2 (x_3 *_3 (\ldots (x_{n-1} *_{n-1} x_n)))),$$

where $*_i = \wedge$ if $a_i = 1$ and $*_i = \vee$ if $a_i = 0$. Notice that, indeed, any assignment satisfying φ_t, when viewed as a binary number, must have value greater than $2 \cdot t$. Notice that all assignments in $\{0,1\}^n$ that satisfy φ_t satisfy φ_{t+1} as well, *except* for the two assignments that are the binary representation of the numbers $2t+1$ and $2t+2$. Thus, since φ_t "almost" implies φ_{t+1}, we have room for hope that we can derive the latter from the former in small space. We now show that this is indeed the case.

In order to show how to infer φ_{t+1} from φ_t and the initial axioms we need to define one more intermediate formula ψ_t. Let $0 \leq m < n$ be the largest index such that $a_m = 0$ (thus, $t = a_1 a_2 \ldots a_{m-1} 011 \ldots 1$), and let

$$\psi_t \stackrel{\text{def}}{=} x_1 *_1 (x_2 *_2 (x_3 *_3 (\ldots (x_{m-1} *_{m-1} x_m)))).$$

In other words, ψ_t is obtained from φ_t by cutting off the maximal suffix consisting entirely of $\wedge$s and encodes that "the satisfying assignment for CT_n must be greater than $2t+1$."

Define two clauses corresponding to the binary representations of the numbers $2t+1, 2t+2$, respectively.

$$A_t = x_1^{a_1} \vee x_2^{a_2} \vee \cdots \vee x_{m-1}^{a_{m-1}} \vee x_m \vee \bar{x}_{m+1} \vee \cdots \vee \bar{x}_n$$

and

$$B_t = x_1^{a_1} \vee x_2^{a_2} \vee \cdots \vee x_{m-1}^{a_{m-1}} \vee \bar{x}_m \vee x_{m+1} \vee \cdots \vee x_n.$$

It is obvious from the semantics of φ_t, ψ_t described above that $\{\varphi_t, A_t\} \models \psi_t$ and $\{\psi_t, B_t\} \models \varphi_{t+1}$. The question is how to produce compact syntactic inferences.

As we noticed before, we can assume w.l.o.g. that $\mathcal{F}$ contains any prescribed finite set of inference rules, and in particular we can assume that $\mathcal{F}$ contains modus ponens. Therefore, it is sufficient to produce inferences of the tautological formulas $(\varphi_t \wedge A_t) \to \psi_t$ and $(\psi_t \wedge B_t) \to \varphi_{t+1}$ that use $O(n)$ variable space. We will consider only the first formula; the second proof is analogous.

For $1 \le \ell \le n$, let $\varphi_t^{(\ell)}, A_t^{(\ell)}, \psi_t^{(\ell)}$ be the suffixes of φ_t, A_t, ψ_t, respectively, that are obtained by crossing out the variables $x_1, \ldots, x_{\ell-1}$. (And we let $\psi_t^{(\ell)} \stackrel{\text{def}}{=} \bot$ if $\ell > m$.) We are going to infer (in linear variable space) all the formulas $(\varphi_t^{(\ell)} \wedge A_t^{(\ell)}) \to \psi_t^{(\ell)}$ by induction on $\ell = n, n-1, \ldots, 1$.

In the base case $\ell = n$ we have $\varphi_t^{(n)} = x_n,\ A_t^{(n)} = \bar{x}_n$, and we get a substitutional instance of the axiom $(A \wedge \bar{A} \to B)$.

In order to infer $(\varphi_t^{(\ell)} \wedge A_t^{(\ell)}) \to \psi_t^{(\ell)}$ from $(\varphi_t^{(\ell+1)} \wedge A_t^{(\ell+1)}) \to \psi_t^{(\ell+1)}$, we use the rule

$$\frac{A \wedge B \to C}{(D \vee A) \wedge (D \vee B) \to (D \vee C)}$$

if $a_\ell = 0$ and the rule

$$\frac{A \wedge B \to C}{(D \wedge A) \wedge (\bar{D} \vee B) \to (D \wedge C)}$$

if $a_\ell = 1$.

We showed how to make the transition from φ_t to φ_{t+1}. At the end we get $\varphi_{2^{(n-1)}-1} = x_1 \wedge x_2 \wedge \cdots \wedge x_n$, which together with $\bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_n$ implies a contradiction. Theorem 6.3 is proved. □

COROLLARY 6.4. *Any tautological formula φ can be inferred from the empty set of axioms $\mathcal{T} = \emptyset$ in variable space $O(VSpace_F(\varphi))$.*

Proof. W.l.o.g. assume that $\{x_1, \ldots, x_n\}$ is the complete list of variables appearing in φ. By induction on the logical complexity of a formula ψ (not necessarily tautological) we produce, for any $\epsilon \in \{0,1\}^n$, an inference of $x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_n^{\epsilon_n} \vee \psi^{\psi(\bar{\epsilon}_1, \ldots, \bar{\epsilon}_n)}(x_1, \ldots, x_n)$ (where, naturally, $\psi^1 \stackrel{\text{def}}{=} \psi$ and $\psi^0 \stackrel{\text{def}}{=} \bar{\psi}$) that has variable space $O(VSpace_F(\psi))$. Since φ is a tautology, this in particular gives, for any $\epsilon \in \{0,1\}^n$, an inference of $x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_n^{\epsilon_n} \vee \varphi(x_1, \ldots, x_n)$ with variable space $O(VSpace_F(\varphi))$. Now we have only to modify the proof of CT_n from Theorem 6.3 by replacing every formula ψ in it with $(\psi \vee \varphi)$ and modifying inference rules accordingly. □

COROLLARY 6.5. *For any tautology $\mathcal{T}$ over n variables*

$$VSpace_F(\mathcal{T}) = O\left(n + \max_{\varphi \in \mathcal{T}} VSpace_F(\varphi)\right).$$

Proof. The proof is similar to the proof of Corollary 6.4. Namely, for every $\epsilon \in \{0,1\}^n$ we can find $\varphi_\epsilon \in \mathcal{T}$ such that $\varphi_\epsilon \to (x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_n^{\epsilon_n})$ is a tautology. When we need the axiom $x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \cdots \vee x_n^{\epsilon_n}$ from CT_n, we download this φ_ϵ, infer $\varphi_\epsilon \to (x_1^{\epsilon_1} \vee x_2^{\epsilon_2} \vee \ldots \vee x_n^{\epsilon_n})$, and apply modus ponens. □

Let $VSpace_F^{sem}(\mathcal{T})$ denote the *semantical* Frege variable space of refuting $\mathcal{T}$.

COROLLARY 6.6. *Semantic and syntactic versions of Frege systems are equivalent in the following variable space model:*

$$VSpace_F^{sem}(\mathcal{T}) \le VSpace_F(\mathcal{T}) \le O(VSpace_F^{sem}(\mathcal{T})).$$

Proof. As in the proof of Theorem 3.7, we show how to emulate the semantic inference π by a syntactic one. The only difference between semantical and syntactical versions is in the inference step. Assume that $\{\varphi_1, \varphi_2, \dots, \varphi_k\} \vdash \psi$. Then we can produce the syntactic proof of the tautology $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k \to \psi$ according to Corollary 6.4 and repeatedly apply modus ponens. □

At the end we briefly discuss how to generalize these results to the case of Frege systems $\mathcal{F}$ in arbitrary complete language L. The problem with the general translation is that the sizes of the resulting formulas may grow very rapidly. However, at least it is not a problem with the logical constant $\mathbf{0}$: it can be trivially replaced with $x \wedge \bar{x}$.

Quite fortunately, the specific language that consists of the remaining connectives $\{\neg, \wedge, \vee, \to\}$ is known to be the weakest in the sense that it can be modeled in any other complete language with only *linear* blow-up in the variable space. More exactly, the following holds.

LEMMA 6.7 (Reckhow [Rec76]). *If $\mathcal{F}$ is a Frege system over any complete language L, then there are L-formulas $NOT(x,z)$, $AND(x,y,z)$, $OR(x,y,z)$, and $IMP(x,y,z)$ such that*

(1) *$NOT(x,z)$ contains one occurrence of x, and $AND(x,y,z)$, $OR(x,y,z)$, and $IMP(x,y,z)$ contain exactly one occurrence of each of x and y;*

(2) *the four formulas represent the Boolean functions $\neg x$, $(x \wedge y)$, $(x \vee y)$, and $(x \to y)$; in particular, the truth values of the formulas are independent of the truth value of z.*

Thus we can rewrite the proof of Theorem 6.3 and its corollaries almost literally, replacing our standard connectives with $NOT(x,z)$, $AND(x,y,z)$, $OR(x,y,z)$, and $IMP(x,y,z)$.

7. Open questions. We conclude this paper with a short list of interesting open questions:

1. Is there any way to capture the notion of propositional space complexity in the uniform framework of first-order theories of bounded arithmetic?
2. Find an unsatisfiable CNF $\mathcal{T}$ in n variables such that $VSpace_R(\mathcal{T}) \ge \omega(n)$ and $\mathcal{T}$ has only polynomially many clauses. (CT_n has exponentially many clauses, and the bound in Corollary 5.7 is linear only in the overall number of variables mn.) We conjecture that $VSpace_R(T(G,\sigma)) \ge \Omega(n^2)$ if G is a 3-regular expander graph and even that

$$VSpace_{PCR}(T(G,\sigma)) \ge \Omega(n^2)$$

when $char(F) \ne 2$.
3. Are there any other interesting fragments of Frege systems, not contained in PCR, for which the notion of variable space makes sense (perhaps cutting planes, depth 2 Frege, etc.)? Can one prove nontrivial lower bounds for these systems?
4. Can we prove the analogue of Theorem 3.7 for variable space? What can be said about the relation between the syntactical and semantical versions of other proof systems with respect to either clause or variable space?

5. Is it possible to prove superconstant clause space lower bounds for PCR-proofs of any bounded fan-in tautology or of $Count_p$? Once more, we conjecture that Corollary 3.27 can be extended to PCR over any field with $char(F) \neq 2$.
6. (See [ET99].) Is it possible to find some strong connection between the clause complexity of a tautology and its minimal proof width for resolution?

Appendix. The purely semantical system called FC works with *arbitrary* Boolean functions, regardless of their syntactical representation complexity. In fact our space complexity for FC defined below will simply minimize over all such representations. Although this system is not natural, the clause space lower bound for the polynomial calculus applies to it as well, and it is a useful tool for proving lower bounds when an abstraction from particulars of a given syntactical system is desirable and instructive.

The line of an FC derivation is an arbitrary Boolean function. The single inference rule is the semantical one, i.e., derive g from $f_1, f_2, \ldots, f_k$ whenever $f_1, f_2, \ldots, f_k \models g$. The definitions of derivations and refutations are analogous to those of resolution and PC.

When defining the clause space for FC, we must overcome the following problem. A line in FC is an *arbitrary* Boolean function f. Clearly, f can be represented by many circuits over some complete Boolean basis, each with a different amount of clauses. The natural way to solve this problem is to define the clause space to be the minimal number of clauses in any such representation.

DEFINITION 7.1 (clause space for FC). *For $\mathcal{M}$ a set of Boolean functions over $x_1, \ldots, x_n$, the* clause space *of $\mathcal{M}$ in FC, denoted $FCSpace(\mathcal{M})$, is the minimal s such that we can choose s clauses with the property that every $f \in \mathcal{M}$ can be represented as a Boolean function over the chosen clauses. Formally,*

$$\begin{aligned} FCSpace(\mathcal{M}) \stackrel{\text{def}}{=} \min \{\, s \mid \exists (C_1(x_1, \ldots, x_n), C_2(x_1, \ldots, x_n), \ldots, \\ C_s(x_1, \ldots, x_n)) \text{ for all } (f(x_1, \ldots, x_n) \in \mathcal{M}) \exists g(y_1, \ldots, y_s) \\ (f \equiv g(C_1, \ldots, C_s)) \,\}, \end{aligned}$$

where C_i are clauses, and g runs over arbitrary Boolean functions in s variables.

We can also define multivalued FC over the domain D in a natural way.

DEFINITION 7.2 (multivalued FC). FC over the domain D *(FC(D)) is the purely semantical system which keeps in memory arbitrary functions $f(x_1, \ldots, x_n) : D^n \to \{0, 1\}$. The inference rule is the semantical one.*

The clause space measure $FCSpace_D(\mathcal{M})$ of a set of such functions $\mathcal{M}$ is the minimal s such that we can choose s multivalued clauses with the property that every $f \in \mathcal{M}$ can be represented as an (ordinary!) Boolean function over the chosen multivalued clauses.

We now prove our main theorems for FC. The first is a better upper bound on clause space of CT_n, which is proved using the method of Theorem 4.2.

THEOREM 7.3. $FCSpace(CT_n) \leq n/2 + 2$.

Proof. By Claim 4.4 (that applies to FC just as well), we need analyze only the base case and show that $FCSpace(CT_2) \leq 2$ ($s = k = 1$):

$$\begin{pmatrix} \text{Axiom: } x_1 \vee x_2 \\ \text{Axiom: } x_1 \vee \bar{x}_2 \end{pmatrix} \rightsquigarrow \begin{pmatrix} x_1 \end{pmatrix} \rightsquigarrow \begin{pmatrix} x_1 \\ \text{Axiom: } \bar{x}_1 \vee x_2 \end{pmatrix}$$
$$\rightsquigarrow \begin{pmatrix} \neg(\bar{x}_1 \vee \bar{x}_2) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \neg(\bar{x}_1 \vee \bar{x}_2) \\ \text{Axiom: } \bar{x}_1 \vee \bar{x}_2 \end{pmatrix} \rightsquigarrow \begin{pmatrix} \bot \end{pmatrix}. \qquad \square$$

Although FC is much stronger than PCR, it turns out that Theorem 4.18 applies equally well to the FC.

THEOREM 7.4. *For any wide tautology $\mathcal{T}$ over n variables with domain D, $FCSpace(\mathcal{T}) \geq \frac{n}{4}$.*

Proof. The proof is identical to the proof of Theorem 4.18. (Notice that in the proof of Lemma 4.14 we used the fact $s = MSpace_D^{sem}(\mathcal{M}_1)$ *only* to write down the representation $||\mathcal{M}_1|| = g(C_1, \ldots, C_s)$ for an *unspecified* Boolean function g, and this transition works perfectly well in the context of FC). □

Acknowledgment. We are grateful to Jan Krajíček for several useful remarks.

REFERENCES

[AL86] R. AHARONI AND N. LINIAL, *Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas*, J. Combin. Theory Ser. A, 43 (1986), pp. 196–204.

[BP98] P. BEAME AND T. PITASSI, *Propositional proof complexity: Past, present and future*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 65 (1998), pp. 66–89.

[BG99] M.L. BONET AND N. GALESI, *A study of proof search algorithms for resolution and polynomial calculus*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 422–431.

[CEI96] M. CLEGG, J. EDMONDS, AND R. IMPAGLIAZZO, *Using the Groebner basis algorithm to find proofs of unsatisfiability*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 174–183.

[CR79] S.A. COOK AND R. RECKHOW, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.

[ET99] J.L. ESTEBAN AND J. TORÁN, *Space bounds for resolution*, in Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science, 1999, pp. 530–539.

[IPS97] R. IMPAGLIAZZO, P. PUDLÁK, AND J. SGALL, *Lower bounds for the polynomial calculus and the Groebner basis algorithm*, Comput. Complexity, 8 (1999), pp. 127–144.

[Koz77] D. KOZEN, *Lower bounds for natural proof systems*, in Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 254–266.

[Kra95] J. KRAJÍČEK, *Bounded Arithmetic, Propositional Logic and Complexity Theory*, Cambridge University Press, Cambridge, UK, 1995.

[Kri85] B. KRISHNAMURTHY, *Short proofs for tricky formulas*, Acta Inform., 22 (1985), pp. 253–275.

[RWY97] A. RAZBOROV, A. WIGDERSON, AND A. YAO, *Read-once branching programs, rectangular proofs of the pigeonhole principle and the transversal calculus*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 739–748.

[Rec76] R.A. RECKHOW, *On the Lengths of Proofs in the Propositional Calculus*, Ph.D. thesis, Technical Report 87, Department of Computer Science, University of Toronto, 1976.

[Sta96] G. STALMARK, *Short resolution proofs for a sequence of tricky formulas*, Acta Inform., 33 (1996), pp. 277–280.

[Tar] M. TARSI, *private communication.*

[Tor99] J. TORÁN, *Lower bounds for space in resolution*, in Proceedings of Computer Science Logic, Lecture Notes in Comput. Sci. 1683, Springer, Berlin, 1999, pp. 362–373.

[Tse68] G.S. TSEITIN, *On the complexity of derivation in propositional calculus*, in Studies in Constructive Mathematics and Mathematical Logic, Part 2, Consultants Bureau, New York, London, 1968, pp. 115–125.

[Urq95] A. URQUHART, *The complexity of propositional proofs*, Bull. Symbolic Logic, 1 (1995), pp. 425–467.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1212–1228

AN ENUMERATIVE GEOMETRY FRAMEWORK FOR ALGORITHMIC LINE PROBLEMS IN $\mathbb{R}^3$*

THORSTEN THEOBALD†

Abstract. We investigate the enumerative geometry aspects of algorithmic line problems when the admissible bodies are balls or polytopes. For this purpose, we study the common tangent lines/transversals to k balls of arbitrary radii and $4-k$ lines in $\mathbb{R}^3$. In particular, we compute tight upper bounds for the maximum number of real common tangents/transversals in these cases. Our results extend the results of Macdonald, Pach, and Theobald who investigated common tangents to four unit balls in $\mathbb{R}^3$ [*Discrete Comput. Geom.*, 26 (2001), pp. 1–17].

Key words. tangents, balls, transversals, lines, enumerative geometry, real solutions, computational geometry

AMS subject classifications. 14N10, 68U05, 51M30, 14P99, 52C45

PII. S009753970038050X

1. Introduction.

Algorithmic questions involving lines in $\mathbb{R}^3$ belong to the fundamental problems in computational geometry [36, 26], computer graphics [28], and robotics [33]. As an initial reference example from computational geometry, consider the problem of determining which bodies of a given scene cannot be seen from *any* viewpoint outside of the scene. From the geometric point of view, this leads to the problem of determining the *common tangents to four given bodies* in $\mathbb{R}^3$ (cf. section 2). Other algorithmic tasks leading to the same geometric core problem include computing smallest enclosing cylinders [32], computing geometric permutations/stabbing lines [27, 2], controlling a laser beam in manufacturing [26], or solving placement problems in geometric modeling [10, 17].

If the bodies are polytopes, the common tangents are common transversals of edges [27]; so, in fact, the main geometric task is to compute the common transversals to four given lines in $\mathbb{R}^3$. This geometric problem has been well known for many years (see, e.g., [16]). In particular, if a configuration has only finitely many common transversals, then this number is bounded by 2; and it is well known how to characterize the configurations with infinitely many common transversals.

On the other hand, the following theorem in [21] shows that this situation completely changes if the bodies under investigation are unit balls (see also [35, 23]).

PROPOSITION 1. *Four unit balls in* $\mathbb{R}^3$ *have at most* 12 *common tangent lines unless their centers are located on the same line. Furthermore, there exists a configuration with* 12 *tangents; i.e., the upper bound is tight.*

Essentially, this means that algebraically this tangent problem is of degree 12. Note that due to this high degree, proving the characterization of the configurations with infinitely many common tangents is a highly nontrivial task.

However, concerning the class of tangent problems to four given bodies, Proposition 1 solves only one particular case. In the present paper, we develop techniques to analyze a substantially larger class of variants. In particular, we aim at filling the gaps between the two extreme situations mentioned before by considering common

*Received by the editors November 6, 2000; accepted for publication (in revised form) January 7, 2002; published electronically April 26, 2002.
http://www.siam.org/journals/sicomp/31-4/38050.html
†Zentrum Mathematik, Technische Universität München, D–80290 München, Germany (theobald@ma.tum.de).

TABLE 1

Summary of results and references of known results. For the case of four balls of general radii we are able to provide a formulation with Bézout bound 12 *(which improves the results from* [17] *substantially; see section* 4*).*

	Upper bound # solutions		# Real solutions of our construction		Characterization of degenerate instances	
4 lines	2	(well known)	2	(well known)	yes	(well known)
3 lines, 1 ball	4		4		yes	
2 lines, 2 balls	8		8		–	
1 line, 3 balls	12		12		–	
4 unit balls	12	[21]	12	[21]	yes	[21]
4 balls	12	([17])	12	[21]	–	

tangents/transversals to k balls and $4-k$ lines, $k \in \{0,\dots,4\}$. For convenience of notation, we consider a transversal of a line as a tangent to the line. Our investigations do not only clarify the exact growth in algebraic degree from 2 to 12 but also provide effective means to tackle these questions when the symmetry (in the sense of identical bodies) is lost. From the algorithmic point of view, these problems of common tangents immediately arise in the mentioned applications when the class of admissible bodies in the scene consists of both balls and polytopes (see section 2).

As the main contribution of this paper, we compute *tight* upper bounds for the number of common tangents to k balls and $4-k$ lines in the finite case, $k \in \{0,\dots,4\}$. Here, *tightness* refers to the following (quite strong) sense of real algebraic geometry (cf. [34]): On the one hand, for each k we bound the number of solutions by algebraic methods, say, by some number m. Then, on the other hand, we provide a construction which indeed leads to m solutions in *real* space $\mathbb{R}^3$ (which would not be possible if any polynomial formulation contained some complex solutions or solutions at infinity).

The general difficulty of proving tight bounds of this kind may be seen by the following two aspects. For the classical enumerative geometry problem of conics tangent to five given conics (dating back to Steiner in 1847) the existence problem of 3264 *real* solutions had not been solved until a few years ago (see [30] and [14, sect. 7.2]). Furthermore, as pointed out in [34], there are nearly no criteria or general techniques for tackling these type of questions. For these reasons, it is even more remarkable that in *all* (!) of the situations there exists a construction matching the upper bound.

Table 1 summarizes our results and provides references of known results. It shows the upper bounds for the number of solutions and the matching numbers of real solutions in our constructions. The last column shows that only in a few cases are we able to explicitly characterize the configurations with an infinite number of common tangents. Namely, besides the already existing results for four lines and four unit balls, we add the characterization for three lines and one ball. In the entries with a "–" we cannot give such a characterization and will discuss this issue at the end of the paper.

Let us point out that the proofs of these results are of quite different flavors. For $k \in \{1,2\}$, the upper bounds immediately follow from Bézout's theorem. Whereas for $k=1$ it is easy to give a construction matching this bound, the construction for $k=2$ is quite involved. In particular, for $k=2$ we apply tools from algebraic geometry and computer algebra (e.g., standard bases) to prove correctness of the construction. However, proving the tight upper bound for three balls and one line is completely different. Here, the Bézout bound in our formulation will be 16 instead of 12. In order to find a better bound for the number of real solutions, we have to analyze the

underlying algebraic geometry of the problem in detail. Finally, in the proof for four balls of general radii we use elementary geometry to find a formulation with Bézout bound 12. Altogether, we think that this variety of techniques can serve to provide many ideas when tackling related problems.

This paper is structured as follows. In section 2, we establish the connection between the algorithmic problems and the geometric tangent problems. Then, after providing some algebraic background on Plücker coordinates in section 3, we prove the necessary results for Table 1 in section 4. We conclude the paper with a short discussion of the remaining open questions.

2. Motivation and algorithmic background. The problem under investigation represents the algebraic core problem within several algorithmic applications mentioned in the introduction. Exemplarily, we describe two of them.

Partial visibility. Consider the following problem from ray-tracing with moving viewpoints. Here, we want to compute information on the viewpoint positions where the visibility topology of the scene changes. As a special case, this includes tackling the following core problem of partial visibility.

A set $B \subset \mathbb{R}^n$ (say, $n \in \{2,3\}$) is called a *(convex) body* if it is bounded, closed, convex, and contains an inner point. Now we consider a scene consisting of a set $\mathcal{B}$ of (not necessarily disjoint) bodies from a specific class $\mathcal{X}$ in $\mathbb{R}^n$. ($\mathcal{X}$ might be the set of all balls or the set of all polytopes.) A body $B \in \mathcal{B}$ is called *partially visible from a viewpoint* v if there exists a line segment connecting v and B not intersecting with the interior of any other body in $\mathcal{B}$. A body $B \in \mathcal{B}$ is called *partially visible* if B can be seen from some viewpoint "outside" of the scene, i.e., if there exists a ray starting at B not intersecting with the interior of any other body in $\mathcal{B}$. We call such a ray a *visibility ray* for B. Bodies which are not partially visible can be immediately removed from the scene, which reduces the complexity of the visualization process. In case of dense crystals whose atoms are visualized as sufficiently large balls, the reduction in complexity may be quite substantial.

In the two-dimensional case, checking partial visibility of a body B can be reduced to a finite number of geometric problems as follows (cf. the treatment of stabbing lines in [12]). Without loss of generality let $|\mathcal{B}| \geq 2$. If there exists a visibility ray for B, then we can continuously transform (i.e., translate and rotate) the visibility ray until we reach a situation where the underlying line is tangent to at least two of the bodies. (One of them might be B itself.) Hence, it suffices to compute the set of all common tangent lines to a pair of bodies in $\mathcal{B}$ and check whether one of these lines contains a visibility ray. For any pair of bodies, the number of common tangent lines is at most four (which is a very special case of the results in [6, 19] on the number of common supporting hyperplanes in general dimension).

In the three-dimensional case we can essentially proceed analogously. Since a line in $\mathbb{R}^3$ has four degrees of freedom, the core problem is to compute the common tangents to four bodies in $\mathbb{R}^3$ (cf. [27, 2]). However, in the three-dimensional case, there are also some special cases where we can transform a visibility ray only to a situation with two or three bodies, or where a configuration with four bodies has an infinite number of common tangents.

For a polytope P, any tangent to P intersects an edge of P. Hence, if $\mathcal{X}$ contains balls and polytopes, we have to compute common tangents/transversals to k lines and $4-k$ balls, $0 \leq k \leq 4$. An algorithmic treatment of the situations with infinitely many common tangents (depending on the class $\mathcal{X}$ of bodies) requires an *a priori* characterization of the configurations with infinitely many common tangents.

In contrast to some other problems in computational geometry, characterizing these situations cannot be neglected (say, by applying perturbation techniques [11]), since the large algebraic degree involved makes it highly nontrivial to guarantee a correct perturbation.

Envelopes. Let $\mathcal{B}$ be a collection of n convex bodies in $\mathbb{R}^3$. A line l is called a *line transversal* of $\mathcal{B}$ if it intersects every member of $\mathcal{B}$. The set of line transversals of $\mathcal{B}$ can be represented as the region enclosed between an upper and a lower envelope as follows (see [7, 1, 2]). These representations are important in the design of data structures supporting ray shooting queries (i.e., seeking the first body, if any, met by a query ray) [1].

If we exclude lines parallel to the yz-plane, a line l in $\mathbb{R}^3$ can be uniquely represented by its projections on the xy- and xz-planes: $y = \sigma_1 x + \sigma_2$, $z = \sigma_3 x + \sigma_4$. Hence, a line can be represented by the quadruple $(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \in \mathbb{R}^4$.

Let B be a convex body in $\mathbb{R}^3$. For fixed $\sigma_1, \sigma_2, \sigma_3$, the set of lines $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ that intersect B is obtained by translating a line in the z-direction between two extreme values, $(\sigma_1, \sigma_2, \sigma_3, \phi_B^-(\sigma_1, \sigma_2, \sigma_3))$ and $(\sigma_1, \sigma_2, \sigma_3, \phi_B^+(\sigma_1, \sigma_2, \sigma_3))$, which represent lines tangent to B from below and from above, respectively. Hence, the set of line transversals to $\mathcal{B}$ can be represented as

$$\{(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \ : \ \max_{B \in \mathcal{B}} \phi_B^-(\sigma_1, \sigma_2, \sigma_3) \leq \sigma_4 \leq \min_{B \in \mathcal{B}} \phi_B^+(\sigma_1, \sigma_2, \sigma_3),$$

which is a region enclosed between a lower envelope and an upper envelope in $\mathbb{R}^4$. If the elements of $\mathcal{B}$ are balls or polytopes, then the set of line transversals defines a semialgebraic set in $\mathbb{R}^4$ (see [2]). Assuming general position, the vertices (= zero-dimensional faces) of the boundary of this region correspond to lines which are tangent to four of the bodies in $\mathcal{B}$. Similar to the first scenario, any implementation of this basic step has to cope with the enumerative questions treated in the present paper.

The role of an algebraic oracle. In both of these algorithmic scenarios, the problem is reduced to the core problem of finding the common tangents/transversals to k lines and $4 - k$ balls. In literature, core problems of this type are considered to be problems of *constant description complexity* (see, e.g., [2]). Often, it is assumed that one has access to an *algebraic oracle* computing the necessary tangents, and the algorithm is formulated in terms of that oracle. From this point of view, our analysis can be seen as the necessary mathematical investigations on how to build this algebraic oracle.

In particular, any implementation of this algebraic oracle or any interface to a black box subroutine establishing that oracle has to cope with the enumerative questions. From the viewpoint of data structures it is always useful and sometimes even necessary to know a good (i.e., tight) upper bound on the number of these tangent lines. From the viewpoint of program verification, knowing a tight upper bound on the number of tangent lines offers the possibility of strong and valuable consistency checks within a program (in particular with regard to the necessary numerical subroutines; cf. section 5). Finally, from the viewpoint of efficiency, understanding the geometry of the basic problem helps to find the right polynomial formulations for the underlying numerical algorithms.

3. Plücker coordinates. In several of the proofs, we use the well-known Plücker coordinates of lines in projective space $\mathbb{P}^3$ (see, e.g., [16, 8]). Let $x = (x_0, x_1, x_2, x_3)^T$, $y = (y_0, y_1, y_2, y_3)^T \in \mathbb{P}^3$ be two different points on a line l. Then l can be represented (of course not uniquely) by the 4×2-matrix L whose two columns are x and y. The

Plücker vector $p = (p_{01}, p_{02}, p_{03}, p_{12}, p_{13}, p_{23})^T \in \mathbb{P}^5$ of the line is defined by the determinants of the 2×2-submatrices of L, i.e., $p_{ij} = x_i y_j - x_j y_i$. It is well known that the set of vectors in $\mathbb{P}^5$ satisfying the Plücker relation

$$p_{01}p_{23} - p_{02}p_{13} + p_{03}p_{12} = 0 \tag{1}$$

is in one-to-one correspondence with the set of lines in $\mathbb{P}^3$. A line l intersects with a line l' in $\mathbb{P}^3$ if and only if their Plücker vectors p and p' satisfy

$$p_{01}p'_{23} - p_{02}p'_{13} + p_{03}p'_{12} + p_{12}p'_{03} - p_{13}p'_{02} + p_{23}p'_{01} = 0. \tag{2}$$

In order to characterize lines tangent to balls we consider tangent lines to arbitrary quadrics in $\mathbb{P}^3$. Throughout the presentation, we will identify a quadric surface in $\mathbb{P}^3$ with its symmetric 4×4-representation matrix. For example, the sphere with radius r and center $(c_1, c_2, c_3)^T \in \mathbb{R}^3$, in $\mathbb{P}^3$ described by $(x_1 - c_1x_0)^2 + (x_2 - c_2x_0)^2 + (x_3 - c_3x_0)^2 = r^2x_0^2$, is identified with the matrix

$$\begin{pmatrix} c_1^2 + c_2^2 + c_3^2 - r^2 & -c_1 & -c_2 & -c_3 \\ -c_1 & 1 & 0 & 0 \\ -c_2 & 0 & 1 & 0 \\ -c_3 & 0 & 0 & 1 \end{pmatrix}.$$

LEMMA 2. *Let L be a 4×2-matrix representing the line $l \subset \mathbb{P}^3$. l is tangent to a quadric Q in $\mathbb{P}^3$ if and only if the 2×2-matrix L^TQL is singular.*

Proof. If we denote the two columns of L by x and y, then the line l consists of all points

$$\left\{ z = (z_0, z_1, z_2, z_3)^T \,:\, z = \lambda x + \mu y,\, (\lambda, \mu)^T \in \mathbb{R}^2 \setminus \{(0,0)^T\} \right\}.$$

By definition, l is tangent to Q if and only if this line intersects the quadric exactly once (namely, with multiplicity 2) or if it is contained in the quadric. The homogeneous quadratic equation

$$(\lambda x + \mu y)^T Q (\lambda x + \mu y) = 0$$

can be made affine by setting $\mu = 1$. Since the discriminant of this affine quadratic equation in λ is

$$(2x^TQy)^2 - 4(x^TQx)(y^TQy) = -4 \det(L^TQL),$$

the statement follows immediately. ∎

In order to transfer this condition to Plücker coordinates, we introduce the operator

$$\wedge^2 : \mathbb{R}^{m,n} \to \mathbb{R}^{\binom{m}{2},\binom{n}{2}}$$

(cf. [35]). The row and column indices of the resulting matrix are subsets of cardinality 2 of $\{1, \ldots, m\}$ and $\{1, \ldots, n\}$, respectively. For $I \subset \{1, \ldots, m\}$ and $J \subset \{1, \ldots, n\}$, with $|I| = |J| = 2$,

$$(\wedge^2 A)_{I,J} := \det A_{[I,J]},$$

where $A_{[I,J]}$ denotes the 2×2-submatrix of the given matrix A with row indices I and column indices J. Let l be a line in $\mathbb{P}^3$ and L be a 4×2-matrix representing l.

By interpreting the 6×1-matrix $\wedge^2 L$ as a vector in $\mathbb{P}^5$, we observe $\wedge^2 L = p_l$, where p_l is the Plücker vector of l.

LEMMA 3. *A line $l \subset \mathbb{P}^3$ is tangent to a quadric Q if and only if the Plücker vector p_l of l satisfies*

$$p_l^T(\wedge^2 Q)p_l = 0. \tag{3}$$

Proof. Let L be a 4×2-matrix whose two columns contain different points of l. The Cauchy–Binet formula from multilinear algebra (see, e.g., [22]) implies

$$\begin{aligned}\det(L^T Q L) &= (\wedge^2 L^T)(\wedge^2 Q)(\wedge^2 L)\\ &= (\wedge^2 L)^T(\wedge^2 Q)(\wedge^2 L).\end{aligned}$$

Now the claim follows from Lemma 2. ∎

For a sphere with radius r and center $(c_1, c_2, c_3)^T \in \mathbb{R}^3$ the quadratic form $p_l^T(\wedge^2 Q)p_l$ results in

$$\begin{pmatrix} p_{01}\\ p_{02}\\ p_{03}\\ p_{12}\\ p_{13}\\ p_{23}\end{pmatrix}^T \begin{pmatrix} c_2^2+c_3^2-r^2 & -c_1c_2 & -c_1c_3 & c_2 & c_3 & 0\\ -c_1c_2 & c_1^2+c_3^2-r^2 & -c_2c_3 & -c_1 & 0 & c_3\\ -c_1c_3 & -c_2c_3 & c_1^2+c_2^2-r^2 & 0 & -c_1 & -c_2\\ c_2 & -c_1 & 0 & 1 & 0 & 0\\ c_3 & 0 & -c_1 & 0 & 1 & 0\\ 0 & c_3 & -c_2 & 0 & 0 & 1\end{pmatrix} \begin{pmatrix} p_{01}\\ p_{02}\\ p_{03}\\ p_{12}\\ p_{13}\\ p_{23}\end{pmatrix}. \tag{4}$$

4. Proofs and constructions. We show the following theorem.

THEOREM 4. *Given $4-k$ lines and k balls in $\mathbb{R}^3$, $0 \le k \le 4$. If there exist only finitely many common tangent lines to these four bodies, then the number of these tangents is bounded by*

$$\begin{cases} 2 & \text{if } k = 0,\\ 4 & \text{if } k = 1,\\ 8 & \text{if } k = 2,\\ 12 & \text{if } k \in \{3, 4\}.\end{cases}$$

These bounds are tight; i.e., for each k there exists a configuration where the number of different real tangent lines matches the stated number. The bounds are tight even if the balls are unit balls.

For brevity, we denote the maximum numbers of tangent lines in the five situations by N_k, $k \in \{0, \dots, 4\}$. Before proving the statements in the following lemmas, let us recall the following version of Bézout's theorem (see, e.g., [8, p. 91]).

THEOREM 5 (Bézout). *Let $f_1, \dots, f_n$ be homogeneous polynomials in $x_0, \dots, x_n$ of degrees $d_1, \dots, d_n > 0$. If $f_1, \dots, f_n$ have a finite number of common zeros in projective n-space $\mathbb{P}^n$, then the number of zeros (counted with multiplicity) is bounded by $d_1 \cdot d_2 \cdots d_n$.*

Note that the upper bounds $N_0 \le 2$, $N_1 \le 4$, $N_2 \le 8$ immediately follow from Bézout's theorem. Namely, since the common tangent lines to three lines and one ball can be formulated by three linear equations of the form (2), one equation of the form (3) as well as the Plücker relation (1) in the six homogeneous variables $p_{01}, \dots, p_{23}$, we obtain $N_1 \le 4$. Analogously, we obtain $N_0 \le 2$, $N_2 \le 8$.

Further, note that the common transversals to four given lines in three-dimensional space are a well-studied problem in enumerative geometry, and it is well known that

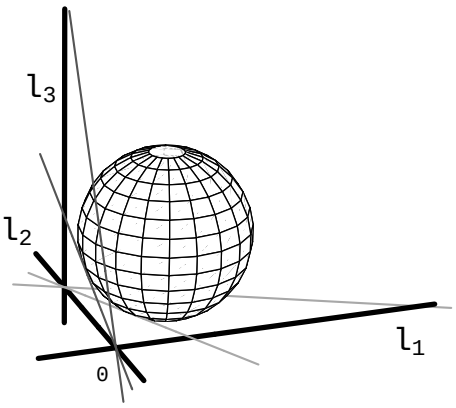


FIG. 1. *The figure shows a configuration with three lines l_1, l_2, l_3, and one ball of radius* 11/10*, leading to four common tangent lines. The two tangent lines in the x_1x_2-plane are drawn in light grey, whereas the two tangent lines in the x_2x_3-plane are drawn in dark grey.*

the upper bound of 2 can actually be achieved in real space $\mathbb{R}^3$ (see, e.g., [16]); hence $N_0 = 2$. The number of common transversals is finite if and only if the Plücker vectors of the four given lines are linearly independent.

In the following, let $B(c, r)$ denote the (closed) ball with center c and radius r.

LEMMA 6. $N_1 = 4$.

Proof. Since $N_1 \leq 4$, it suffices to give a construction with three lines and one ball, leading to four common tangents. Let l_1 be the x_1-axis, l_2 be the x_2-axis, and l_3 be parallel to the x_3-axis and passing through $(0, 2, 0)^T$ (see Figure 1); hence $l_1 \cap l_2 = \{(0, 0, 0)^T\}$ and $l_2 \cap l_3 = \{(0, 2, 0)^T\}$.

Each line intersecting the three lines l_1, l_2, and l_3 is located in the x_1x_2-plane (in which case it passes through $(0, 2, 0)^T$) or is located in the x_2x_3-plane (in which case it passes through the origin). For $1 < r < \sqrt{2}$ the ball $B((1, 1, 1)^T, r)$ intersects both the x_1x_2-plane and the x_2x_3-plane but does not intersect with any of the lines l_1, l_2, l_3. Hence, since there are two tangents to the ball passing through the origin and lying in the x_1x_2-plane, and since there are two tangents to the ball passing through $(0, 2, 0)^T$ and lying in the x_1x_3-plane, there are four common tangents altogether. Figure 1 shows a configuration with $1 < r = 11/10 < \sqrt{2}$. We remark that by appropriate scaling, the ball can be transformed into a unit ball. Furthermore, by slightly perturbing the configuration, the lines can be made pairwise skew. □

To complete the entries for three lines and one ball in Table 1, it remains to characterize the configurations with infinitely many common tangent lines. If the three lines are not pairwise skew, then all common tangent lines lie in the same plane or pass through a point of intersection. Since the resulting characterization can be easily established, we can assume that the three lines are pairwise skew.

It is well known that the common transversals of three pairwise skew lines define a hyperboloid (see, e.g., [31, 3]). By applying a translation and a rotation, the hyperboloid can be transformed into

$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} - \frac{x_3^2}{c^2} = 1 \qquad \text{with } a, b, c > 0. \tag{5}$$

This transformation changes the center of the ball into some new center $(p_1, p_2, p_3)^T \in \mathbb{R}^3$. Now the characterization of infinitely many common tangent lines is given by the following lemma.

LEMMA 7. *Let l_1, l_2, l_3 be three pairwise skew lines whose common transversals generate a hyperboloid of the form (5), and let B_4 be a ball with center $(p_1, p_2, p_3)^T$ and radius $r > 0$. Then there exist infinitely many common tangents to l_1, l_2, l_3, B_4 if and only if $p_1 = p_2 = 0$, $a = b$, and in the x_1x_3-plane the circle $x_1^2 + (x_3 - p_3)^2 = r^2$ is a tangent circle to both branches of the hyperbola $x_1^2/a^2 - x_3^2/c^2 = 1$.*

Proof. The hyperboloid (5) can be parametrized by one of the two sets of generating lines. In particular, this hyperboloid is generated by the set of lines

$$\left\{ (x_1, x_2, 0)^T + \lambda \left(-\frac{a}{bc} x_2, \frac{b}{ac} x_1, 1 \right)^T : \lambda \in \mathbb{R} \right\}, \tag{6}$$

$$\text{where } \frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} = 1 \tag{7}$$

(see, e.g., [18]). In order to characterize those lines which are tangent to the ball, we can apply Lemma 3 to the lines (6) and obtain a polynomial equation in x_1, x_2 of degree at most 4. After bringing the terms of even degree in x_1 to the left side and the terms of odd degree in x_1 to the right side, squaring the equation yields a new equation in which every term is of even degree in x_1. Now we can use (7) to eliminate x_2 and obtain a polynomial equation of degree at most 8 in x_1. Since a univariate polynomial with infinitely many common zeros is the zero polynomial, this polynomial formulation in a single variable implies that if the hyperboloid contains infinitely many tangent lines to the ball, then *all* lines in the parametrization are tangent lines to the ball.

Since the intersection of the hyperboloid with any plane parallel to the x_1x_2-axis is symmetric with respect to the origin, a necessary condition for infinitely many common tangents is $p_1 = p_2 = 0$. In this situation, a configuration with infinitely many common tangents further implies $a = b$. Hence, since $p_1 = p_2 = 0$ and $a = b$, both the hyperboloid and the ball are rotational symmetric with respect to the x_3-axis, and it suffices to consider the section through the x_1x_3-plane. In this section, the circle $x_1^2 + (x_3 - p_3)^2 = r^2$ must be a tangent circle to both branches of the hyperbola $x_1^2/a^2 - x_3^2/c^2 = 1$.

If, conversely, $p_1 = p_2 = 0$, $a = b$, and in the x_1x_3-plane the circle $x_1^2 + (x_3 - p_3)^2 = r^2$ is a tangent circle to the hyperbola $x_1^2/a^2 - x_3^2/c^2 = 1$, then the rotational symmetry implies that every line in the hyperboloid $x_1^2/a^2 + x_2^2/b^2 - x_3^2/c^2 = 1$ is tangent to the ball B_4. Hence, there are infinitely many common tangents. ∎

LEMMA 8. $N_2 = 8$.

Proof. Since $N_2 \leq 8$, it suffices to give a construction with two lines and two balls of the same radius, leading to eight common tangent lines. We start from the following configuration with six different common tangent lines. The two balls are symmetrically located on the x_1-axis: $c_3 = (\gamma, 0, 0)^T$, $c_4 = (-\gamma, 0, 0)^T$; the radius r will be specified below. The lines l_1 and l_2 are chosen in a plane $x_2 = \beta$ for some $\beta > 0$ such that the lines intersect in $(0, \beta, 0)^T$. Hence, every common transversal of the two lines either lies in the plane $x_2 = \beta$ or passes through the point $(0, \beta, 0)^T$. If the two balls intersect with each other, and $\beta < r$, and $(0, \beta, 0)^T$ is not contained in the union of the balls $B(c_3, r)$, $B(c_4, r)$, then there are exactly six different common tangents (see Figure 2): two tangents pass through $(0, \beta, 0)^T$ and lie in the plane $x_1 = 0$; two tangents lie in the plane $x_2 = \beta$ and are parallel to the x_1-axis; and two tangents lie in the plane $x_2 = \beta$ and pass through $(0, \beta, 0)^T$. For the following considerations it is quite useful to have a succinct description of the last two tangents and also to work with integer coefficients for β, γ, and r. In particular, we will force

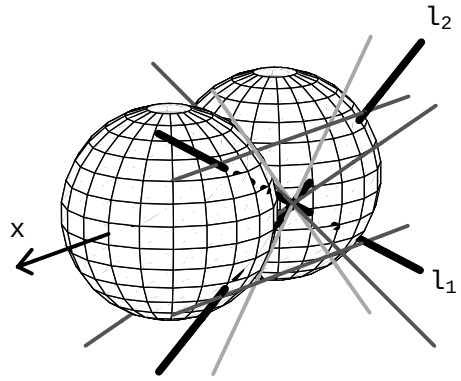


FIG. 2. *The figure shows a construction with two lines and two balls, leading to six different tangent lines. The two tangents lying in the plane $x_2 = \beta$ and passing through $(0, \beta, 0)^T$ are drawn in light grey. The other four tangents are drawn in dark grey.*

the two tangents in the plane $x_2 = \beta$ and passing through $(0, \beta, 0)^T$ to be of the form $(0, \beta, 0)^T + \lambda(1, 0, \pm 1)^T$. In order to obtain these tangents, β, γ, and r have to satisfy $\beta^2 + \gamma^2/2 = r^2$ and $r > \gamma$. An appropriate choice is $\beta = 7$, $\gamma = 8$, and $r = 9$, so that the tangents of the last type are

$$t_1 := \left\{(0,7,0)^T + \lambda(1,0,1)^T \,:\, \lambda \in \mathbb{R}\right\} \text{ and } t_2 := \left\{(0,7,0)^T + \lambda(1,0,-1)^T \,:\, \lambda \in \mathbb{R}\right\}.$$

Now the key observation is that these two tangents have multiplicity 2. In order to prove this we consider the system of Plücker equations stemming from (2) and (4). Independent of the specific choice of lines l_1, l_2 with the above properties, the common transversals of l_1 and l_2 are given by the common zeros of the two linear, homogeneous polynomials

$$\begin{aligned} f_1 &= -7p_{03} + p_{23}, \\ f_2 &= 7p_{01} + p_{12}. \end{aligned}$$

The quadratic equations resulting from the balls $B(c_3, r)$ and $B(c_4, r)$ are

$$\begin{aligned} f_3 &= -81p_{01}^2 - 17p_{02}^2 - 17p_{03}^2 - 16p_{02}p_{12} + p_{12}^2 - 16p_{03}p_{13} + p_{13}^2 + p_{23}^2, \\ f_4 &= -81p_{01}^2 - 17p_{02}^2 - 17p_{03}^2 + 16p_{02}p_{12} + p_{12}^2 + 16p_{03}p_{13} + p_{13}^2 + p_{23}^2. \end{aligned}$$

Furthermore, let $f_5 = p_{01}p_{23} - p_{02}p_{13} + p_{03}p_{12}$ be the polynomial of the Plücker relation (1).

The tangent t_1 has Plücker coordinate $(1, 0, 1, -7, 0, 7)^T$. In order to compute the multiplicity of this tangent, we follow the method and the notation in [9, sect. 4.4]. First we pass to an affine version of the polynomials by adding the polynomial $f_6 = p_{01} - 1$; this forces $p_{01} = 1$ in any common zero of the system. Then we move the point t_1 to the origin by applying the linear variable transformation

$$(p_{01}, p_{02}, p_{03}, p_{12}, p_{13}, p_{23})^T = (q_{01}, q_{02}, q_{03}, q_{12}, q_{13}, q_{23})^T + (1, 0, 1, -7, 0, 7)^T.$$

The local intersection multiplicity μ can be computed as the vector space dimension of the quotient ring

$$\mu = \dim R_l / I_l,$$

where $R_l := \mathbb{C}[q_{01}, \ldots, q_{23}]_{\langle q_{01},\ldots,q_{23}\rangle}$ is the local ring whose elements are the rational functions in $q_{01}, \ldots, q_{23}$ with nonvanishing denominator at 0. I_l is the ideal defined by $f_1, \ldots, f_6$ in the local ring R_l.

In order to compute μ, we use the fact that in case of finite dimension

$$\dim R_l/I_l = \dim R_l/\langle \mathrm{LT}(I_l)\rangle,$$

where $\langle \mathrm{LT}(I_l)\rangle$ denotes the ideal generated by the leading terms of I_l (see, e.g., [9, Chap. 4, Cor. 4.5]). This dimension can be easily extracted from a standard basis of I_l. (For the convenience of the reader, a short review of standard bases can be found in the appendix.) Since by our choice of β, γ, and r all coefficients are integers, we can apply a computer algebra package (e.g., `Singular` [15]) to compute a standard basis $\{h_1, \ldots, h_6\}$ of the ideal I_l with respect to antigraded reverse lexicographical order:

$$\begin{aligned}
h_1 &= q_{01},\\
h_2 &= 112q_{02} + 34q_{03} + 14q_{12} - 16q_{13},\\
h_3 &= 14q_{03} + q_{12},\\
h_4 &= q_{12},\\
h_5 &= 64q_{23},\\
h_6 &= 112q_{13}^2.
\end{aligned}$$

Hence, the leading monomials of $h_1, \ldots, h_6$ with respect to antigraded reverse lexicographical order are q_{01}, q_{02}, q_{03}, q_{12}, q_{23}, q_{13}^2. The desired multiplicity μ is the cardinality of the set of cosets $\{1 + I_l, q_{13} + I_l\}$, which implies $\mu = 2$. By symmetry, the tangent t_2 has multiplicity 2 as well.

Now we choose one particular configuration of the presented class, namely the one with $l_1 := t_1$ and $l_2 := t_2$. By perturbing this configuration, the two double tangent lines will split into four different tangent lines: first, we slightly increase the x_2-coordinate of the line l_2 so that the resulting line l_2' becomes $(0, \beta', 0)^T + \lambda(1, 0, -1)^T$ for some $\beta' > \beta$. In this process, the double tangent t_1 splits into two tangents t_1^a and t_1^b intersecting l_1 and l_2' in different orders; i.e., one of the tangents t_1^a, t_1^b touches l_1, l_2, B_3, and B_4 in the order (B_3, l_1, l_2, B_4) and one of them in the order (B_3, l_2, l_1, B_4). However, the tangent t_2 is still a double zero of the system of polynomials, since the parallel lines t_2 and l_2' intersect in the plane at infinity of $\mathbb{P}^3$.

Similarly, we can make the double tangent t_2 split into two tangents by slightly decreasing the x_2-coordinate of the line l_1; denote the resulting line by l_1'. Figure 3 shows the configuration for l_1' passing through the points $(0, 6.5, 0)^T$, $(2, 6.5, 2)^T$ and l_2' passing through the points $(0, 7.5, 0)^T$, $(2, 7.5, -2)^T$. □

For N_3 the situation is more involved. The Bézout bound gives 16, but, in fact, the number of real common tangents is bounded by 12. Our proof is based on some algebraic-geometric investigations of the common tangents to four unit balls by Macdonald [20]. By appropriately applying these considerations to the situation with three balls and one line, it will turn out that there are always two solutions at infinity with multiplicity at least 2. For the general background on the algebraic and geometric concepts used in the subsequent proofs, easily accessible introductions can be found in [25, 29].

We start with the following observation in [35]. The sphere with center $(c_1, c_2, c_3)^T \in \mathbb{R}^3$ and radius r has the homogeneous equation in $\mathbb{P}^3$

$$(x_1 - c_1x_0)^2 + (x_2 - c_2x_0)^2 + (x_3 - c_3x_0)^2 = r^2x_0^2.$$

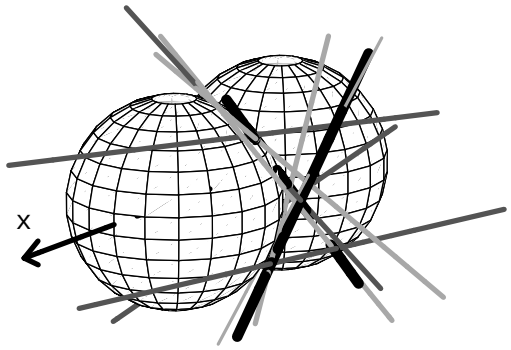


FIG. 3. *Construction with two lines and two balls, leading to eight common tangent lines.*

In the plane at infinity $x_0 = 0$, this gives the equation

$$x_1^2 + x_2^2 + x_3^2 = 0,$$

which is independent of the center and the radius. Let ω denote this conic section in the plane at infinity. Later in the proof, we will work in the space of lines in $\mathbb{P}^3$. In that situation, we will have to consider those tangents through any point $z \in \omega$ in the plane at infinity rather than z itself. For this reason, we provide a characterization of these tangents.

LEMMA 9. *Let $z = (0, \zeta_1, \zeta_2, \zeta_3)^T \in \omega$. The tangent to the conic ω at z which lies in the plane at infinity has Plücker coordinate*

$$(p_{01}, p_{02}, p_{03}, p_{12}, p_{13}, p_{23})^T = (0, 0, 0, \zeta_3, -\zeta_2, \zeta_1)^T.$$

In particular, the tangent contains the points $(0, -\zeta_2, \zeta_1, 0)^T$, $(0, \zeta_3, 0, -\zeta_1)^T$, and $(0, 0, -\zeta_3, \zeta_2)^T$.

Proof. Since $\zeta_0 = 0$ we can compute in projective plane $\mathbb{P}^2$; so let $\overline{z} = (\zeta_1, \zeta_2, \zeta_3)^T$. The conic section

$$x^T A x = 0 \qquad \text{with } A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is regular in $\overline{z}$ with tangent $\{y = (y_1, y_2, y_3)^T \in \mathbb{P}^2 : \overline{z}^T A y = 0\}$. In particular, $(-\zeta_2, \zeta_1, 0)^T$, $(\zeta_3, 0, -\zeta_1)^T$, $(0, -\zeta_3, \zeta_2)^T$, and $\overline{z}$ itself lie on this tangent. Now any two of these points can be used to compute the Plücker coordinate of the tangent line. □

Consider a configuration with a line l_1 and three spheres Q_2, Q_3, and Q_4 in $\mathbb{R}^3$. The idea for proving the solutions at infinity is to transfer the geometry of ω to the space of lines in $\mathbb{P}^3$. More precisely, let t be a tangent to ω at z in the plane at infinity. Since the quadrics $\wedge^2 Q_2, \wedge^2 Q_3, \wedge^2 Q_4 \in \mathbb{P}^5$ characterize the tangents to Q_2, Q_3, Q_4, the Plücker vector p_t of t is contained in $\wedge^2 Q_2$, $\wedge^2 Q_3$, and $\wedge^2 Q_4$. Let Ω denote the quadric in $\mathbb{P}^5$ defined by the Plücker equation (1). Since t is a line in $\mathbb{P}^3$, t is also contained in Ω. We will show that the tangent hyperplanes to the quadrics $\wedge^2 Q_2$, $\wedge^2 Q_3$, $\wedge^2 Q_4$, Ω at p_t contain a common subspace of dimension 2. In connection with

the linear form defined by the transversals of the line l_1, this will prove the multiplicity of at least 2.

Let us investigate the spheres Q_2, Q_3, Q_4 first. For $i \in \{2,3,4\}$, we are looking for lines whose Plücker vectors lie in the tangent hyperplane of $\wedge^2 Q_i$ at p_t. The geometric concept behind this relation is *polarity*. Recall that the polar plane of a point $a \in \mathbb{P}^n$ with respect to an arbitrary quadric Q is defined by

$$\{y \in \mathbb{P}^n \,:\, a^T Q y = 0\}.$$

If $a \in Q$, then the polar hyperplane is a tangent hyperplane. The polar line of a line $l \in \mathbb{P}^3$ is defined by

$$\{y \in \mathbb{P}^3 \,:\, a^T Q y = 0 \text{ for all } a \in l\}.$$

The following lemma establishes a connection between the tangent hyperplanes to $\wedge^2 Q$ and the concept of polarity for a quadric Q.

LEMMA 10. *Let t be a tangent line to a quadric $Q \subset \mathbb{P}^3$, and let the point $a \in \mathbb{P}^3$ be contained in the polar line of t. Then, for any line l containing a, the Plücker vector p_l of l is contained in the tangent hyperplane to $\wedge^2 Q$ at p_t, i.e., $p_t^T(\wedge^2 Q)p_l = 0$.*

Proof. Let T be a representation of t by a 4×2-matrix as described in section 3. Further, let b be a point on l with $b \neq a$, and let $L = (a, b)$ be a representation of l by a 4×2-matrix. Since a is contained in the polar line of t, we have $T^T Q a = (0,0)^T$. Hence, by reasoning as in Lemma 3, we can conclude

$$p_t^T(\wedge^2 Q)p_l = \det(T^T Q L) = 0. \qquad \square$$

In particular, the following version of a well-known relationship (see, e.g., [25]) shows that the precondition of Lemma 10 is satisfied if $a = t \cap Q$.

LEMMA 11. *If t is tangent to a quadric Q at some point a, then a is contained in the polar line of t.*

Proof. Let $y \neq a$ be a point on t. Since t lies on the polar plane (namely, the tangent plane) of a with respect to Q, we have $a^T Q y = 0$. Since also $a^T Q a = 0$, a lies on the polar line of t with respect to Q. $\square$

Finally, we are ready to prove the upper bound for N_3.

LEMMA 12. $N_3 \leq 12$.

Proof. Let L_1 be the hyperplane (2) in $\mathbb{P}^5$ characterizing the transversals of the line l_1; that is, any point on L_1 which satisfies the Plücker relation is the Plücker coordinate of a transversal to l_1. Let $\wedge^2 Q_2, \wedge^2 Q_3, \wedge^2 Q_4$ be the quadrics (4) characterizing the tangents to the three balls. Further, let $z = (0, \zeta_1, \zeta_2, \zeta_3)^T \in \omega$, and let $\pi \subset \Omega \subset \mathbb{P}^5$ be the set of Plücker vectors whose corresponding lines in $\mathbb{P}^3$ pass through z. π can be written as the image of the projective mapping $h : \mathbb{P}^3 \to \Omega \subset \mathbb{P}^5$,

$$h(y_0, y_1, y_2, y_3) = \wedge^2 \begin{pmatrix} 0 & y_0 \\ \zeta_1 & y_1 \\ \zeta_2 & y_2 \\ \zeta_3 & y_3 \end{pmatrix}.$$

Since h is linear, it follows that π is a two-dimensional plane in $\mathbb{P}^5$ with $\pi \subset \Omega$.

Let t be the tangent to ω at z in the plane at infinity. By Lemmas 11 and 10, π is contained in the tangent hyperplane to $\wedge^2 Q_i$ at p_t, $2 \leq i \leq 4$.

In order to show that π is also contained in the tangent hyperplane to Ω at p_t, let y be a point different from z, and let l be a line through z and y. Then, by Lemma 9, the Plücker vectors p_t and p_l satisfy

$$\begin{aligned} p_t^T \Omega p_l &= (0,0,0,\zeta_3,-\zeta_2,\zeta_1) \cdot \frac{1}{2} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} -\zeta_1 y_0 \\ -\zeta_2 y_0 \\ -\zeta_3 y_0 \\ \zeta_1 y_2 - \zeta_2 y_1 \\ \zeta_1 y_3 - \zeta_3 y_1 \\ \zeta_2 y_3 - \zeta_3 y_2 \end{pmatrix} \\ &= -\frac{1}{2} y_0 (\zeta_1^2 + \zeta_2^2 + \zeta_3^2) \\ &= 0\,. \end{aligned}$$

Hence, the four tangent hyperplanes of $\wedge^2 Q_2$, $\wedge^2 Q_3$, $\wedge^2 Q_4$, Ω at p_t contain a common subspace of dimension at least 2. By Lemma 9, the tangents to the conic ω lie on a conic $\overline{\omega}$, namely on

$$p_{12}^2 + p_{13}^2 + p_{23}^2 = 0,$$

in the two-dimensional subspace of $\mathbb{P}^5$ given by $p_{01} = p_{02} = p_{03} = 0$. The restriction of the hyperplane L_1 to the subspace $p_{01} = p_{02} = p_{03} = 0$ defines a one-dimensional subspace $\overline{L_1}$. Since $\overline{L_1}$ is one-dimensional, it intersects with $\overline{\omega}$ at two points $b_1, b_2 \in \mathbb{P}^5$ in the plane $p_{01} = p_{02} = p_{03} = 0$. Further, since b_1 and b_2 satisfy the Plücker relation, they are Plücker vectors of some tangents t_1 and t_2 to ω. Altogether, the five tangent hyperplanes of $\wedge^2 Q_2$, $\wedge^2 Q_3$, $\wedge^2 Q_4$, Ω, L_1 at b_1 and b_2 contain a common subspace of dimension at least 1. Hence, the tangent hyperplanes are not independent, which implies that the multiplicity of intersection in b_1 and b_2 is at least 2 (see, e.g., [24, p. 115]). □

In order to show that $N_3 = 12$ it remains to give a construction with one line l_1 and three balls B_2, B_3, B_4 of the same radius r, leading to 12 common tangents. Let l_1 be the x_3-axis, and let the centers c_2, c_3, c_4 of the balls constitute an equilateral triangle with edge length 1 in the x_1x_2-plane, say $c_2 = (\sqrt{3}/3, 0, 0)^T$, $c_3 = (-\sqrt{3}/6, 1/2, 0)^T$, $c_4 = (-\sqrt{3}/6, -1/2, 0)^T$ (see Figure 4). For $1/2 < r < \sqrt{3}/3$, the balls are non-disjoint, and none of them contains the origin.

Let t be a line which intersects l_1, and let H be the plane containing t and l_1. The three cuts $H \cap B_1$, $H \cap B_2$, and $H \cap B_3$ are discs (maybe degenerated to single points or empty sets). Unless H is equidistant to two of the centers, one of these discs is strictly contained in one of the other two. Hence, any common tangent to the line and the three balls lies in one of the three planes which contain the x_3-axis and which are equidistant to two of the centers.

For example, one of these planes is the x_1x_3-plane, which is equidistant to c_2 and c_3. The section through this plane contains two disjoint discs: one representing the (identical) intersections of the plane with B_2 and B_3, and the second one because of B_1. These two discs are separated by the line l_1. Hence, in this plane there are four common tangents. Altogether, since there are three planes of this kind, we have 12 common tangents.

Finally, it remains to analyze the common tangents to four balls (with arbitrary radii) in $\mathbb{R}^3$. Of course, this problem can also be formulated in Plücker coordinates. However, since the solutions of these equations have a common component at infinity

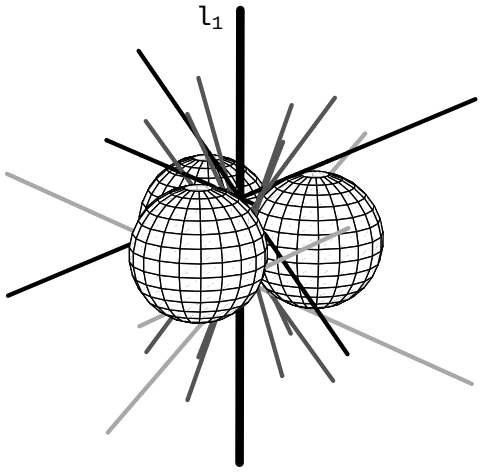


FIG. 4. *Construction with one line and three balls, leading to* 12 *tangents.*

[35], we prefer to compute the number of tangents by an elementary approach. Recently, in [17] the common tangents to four balls have been formulated by polynomial equations with Bézout number 24. We improve this result by giving a polynomial formulation with Bézout number 12; this is optimal by Proposition 1.

The idea for obtaining the system with Bézout bound 12 is to generalize the approach for unit balls in [21]. Note that in the proof we will always refer to the generic case. For this reason—in contrast to Proposition 1—the proof does not provide a precise characterization of the cases with infinitely many common tangent lines.

LEMMA 13. $N_4 \leq 12$.

Proof. Let $c_1, \ldots, c_4$ be affinely independent, and, without loss of generality, let r_4 be the smallest of the radii. We consider functions $\rho_i : [0, r_4] \to \mathbb{R}$ with $\rho_i(0) = 0$, $\rho_i(r_4) = r_i$. Let $\rho_4(t) = t$, while $\rho_i(t)$ for $1 \leq i \leq 3$ will be specified below. First we describe the set of lines which are tangent to the balls $B(c_i, \rho_i(t))$ for $t > 0$.

A line l will be specified by its homogeneous direction vector $s = (s_1, s_2, s_3)^T$ and its closest point p to the origin.

The line l has distance $\rho_i(t)$ from some point c_i if and only if the line $l - p$ (which passes through the origin) has distance $\rho_i(t)$ from $c_i - p$, i.e., if and only if

$$((c_i - p) \times s)^2 = \rho_i^2(t) s^2.$$

Introducing the moment vector $m := p \times s$ and applying Lagrange's identity gives

$$(c_i \times s)^2 - 2\langle c_i, p \rangle s^2 + m^2 - \rho_i^2(t) s^2 = 0. \tag{8}$$

Choosing c_4 to be at the origin and subtracting (8) for index 4 from this equation for index $i \in \{1, 2, 3\}$ yields linear equations in p:

$$\langle c_i, p \rangle = \frac{1}{2s^2}(c_i \times s)^2 - \frac{1}{2}(\rho_i^2(t) - t^2), \qquad 1 \leq i \leq 3. \tag{9}$$

Setting $M := (c_1, c_2, c_3)^T$, we obtain the vector equation

$$p = \frac{1}{2s^2} M^{-1} \begin{pmatrix} (c_1 \times s)^2 \\ (c_2 \times s)^2 \\ (c_3 \times s)^2 \end{pmatrix} - \frac{1}{2} M^{-1} \begin{pmatrix} \rho_1^2(t) - t^2 \\ \rho_2^2(t) - t^2 \\ \rho_3^2(t) - t^2 \end{pmatrix}. \tag{10}$$

Now the key idea is that if we choose parametrizations $\rho_i(t)$ with $\rho_i^2(t) - t^2 = C_i$ for some constants $C_i \in \mathbb{R}$, $1 \le i \le 3$, then the vector p is uniquely determined by the direction vector s. Furthermore, the conditions $\rho_i(r_4) = r_i$ imply $C_i = r_i^2 - r_4^2$; hence, $\rho_i^2(t) = t^2 + (r_i^2 - r_4^2)$. By Cramer's rule,

$$M^{-1} = \frac{1}{6V}(c_2 \times c_3, c_3 \times c_1, c_1 \times c_2),$$

where $V := \det(c_1, c_2, c_3)/6$ denotes the oriented volume of the tetrahedron $c_1c_2c_3c_4$. By introducing the normal vectors

$$n_1 := (c_2 \times c_3)/2, \quad n_2 := (c_3 \times c_1)/2, \quad n_3 := (c_1 \times c_2)/2,$$

and substituting (10) into $\langle p, s\rangle = 0$, we can eliminate p and obtain a homogeneous cubic condition for the direction vector s:

$$\sum_{i=1}^{3} \left((c_i \times s)^2 + s^2(r_i^2 - r_4^2)\right) \langle n_i, s\rangle = 0.$$

Any solution s of this equation is the direction vector of a line with distances $\rho_i(t)$ from the four centers for *some* parameter t. Substituting the radius condition $||p|| = r_4$ into (10) gives an equation of degree 4. Since $\rho_i(r_4) = r_i$, $1 \le i \le 4$, any common solution of the cubic and the quartic equation gives a common tangent to the four balls $B(c_i, r_i)$. By Bézout's theorem, the formulation of the tangent problem by a cubic and a quartic equation implies $N_4 \le 12$. □

5. Conclusion and open questions. We have investigated the enumerative geometry questions for the common tangents to four bodies in $\mathbb{R}^3$ when the bodies are balls or polytopes. These results reflect the algebraic complexity inherent in the mentioned applications. In other words, whenever we want to focus on exact computations for the visibility or envelope problems described in section 2, we have to cope with solving systems of polynomial equations of the stated degrees.

The main open problem is to complete the characterization of the degenerate instances in Table 1. For example, in the case of four balls with arbitrary radii there are some obvious situations with infinitely many common tangent lines: whenever the four centers are collinear and the four balls are inscribed in the same hyperboloid H. We conjecture that there does not exist any configuration with four balls of arbitrary radii, noncollinear centers, and infinitely many common tangent lines. However, we were not able to prove this.

From the practical point of view, actually computing the numerical values of the solutions (which has, e.g., been done in finding the constructions given in this paper) requires either multidimensional numerical methods such as homotopy methods or combinations of symbolic techniques with univariate polynomial solvers. (For an introduction into all these techniques see [9].) Since generally these techniques are still computationally expensive, it is important to apply the most appropriate polynomial formulations of the concrete problems. From this point of view, our results provide optimal formulations. Finally, let us mention that there are many research efforts in improving the efficiency of the two mentioned numerical polynomial solving techniques. In particular, for recent improvements and the state of the art of the first technique see [37], and with regard to the second technique see [4, 5, 13].

Appendix: Standard bases. We review the definitions of a standard basis, starting from Gröbner basis theory (see [9]). The theory of Gröbner bases provides computational methods to find "nice" generators for an ideal I in a polynomial ring $\mathbb{C}[x_1, \ldots, x_n]$. The theory of *standard bases* extends this theory for ideals in local rings. More precisely, let $R_l := \mathbb{C}[x_1, \ldots, x_n]_{\langle x_1, \ldots, x_n \rangle}$ be the set of rational functions f/g in $x_1, \ldots, x_n$ with $g(0, \ldots, 0) \neq 0$. R_l defines a *local* ring; i.e., it contains exactly one maximal ideal. Since the algebraic-geometric definitions of intersection multiplicities are related to the concept of local rings, standard bases provide a powerful tool to effectively compute intersection multiplicities.

From the various possible term orders, we restrict ourselves to considering the *antigraded reverse lexicographical order* (arevlex). For $\alpha, \beta \in \mathbb{N}_0^n$, we have $x^\alpha >_{arevlex} x^\beta$ if and only if

$$\sum_{i=1}^{n} \alpha_i < \sum_{i=1}^{n} \beta_i$$

or

$$\sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \beta_i \quad \text{and} \quad x^\alpha >_{revlex} x^\beta,$$

where $>_{revlex}$ denotes the reverse lexicographical order of Gröbner basis theory. For any polynomial f, the leading term of f, denoted $\mathrm{LT}(f)$, is the maximal term of f with regard to the arevlex-order.

For an ideal I in R_l, the *set of leading terms of* I, abbreviated $\mathrm{LT}(I)$, is the set of leading terms of elements of I.

A *standard basis* of I is a set $\{g_1, \ldots, g_t\} \subset I$ such that $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_t) \rangle$. Given a set of polynomial generators of I, a standard basis of I can be effectively computed by variants of the Buchberger algorithm.

Acknowledgments. The author would like to thank Abhi Dattasharma for his useful comments and the anonymous referees for their suggestions and for pointing out the issue of envelopes.

REFERENCES

[1] P.K. Agarwal, B. Aronov, and M. Sharir, *Computing envelopes in four dimensions with applications*, SIAM J. Comput., 26 (1997), pp. 1714–1732.

[2] P.K. Agarwal, B. Aronov, and M Sharir, *Line transversals of balls and smallest enclosing cylinders in three dimensions*, Discrete Comput. Geom., 21 (1999), pp. 373–388.

[3] A. Beutelspacher and U. Rosenbaum, *Projective Geometry: From Foundations to Applications*, Cambridge University Press, Cambridge, UK, 1998.

[4] D.A. Bini, *Numerical computation of polynomial zeros by means of Aberth's method*, Numer. Algorithms, 13 (1996), pp. 179–200.

[5] D.A. Bini and G. Fiorentino, *Design, analysis, and implementation of a multiprecision polynomial rootfinder*, Numer. Algorithms, 23 (2000), pp. 127–173.

[6] S.E. Cappell, J.E. Goodman, J. Pach, R. Pollack, M. Sharir, and R. Wenger, *Common tangents and common transversals*, Adv. Math., 106 (1994), pp. 198–215.

[7] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, and J. Stolfi, *Lines in space: Combinatorics and algorithms*, Algorithmica, 15 (1996), pp. 428–447.

[8] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, 2nd ed., Springer-Verlag, New York, 1996.

[9] D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, Grad. Texts Math. 185, Springer-Verlag, New York, 1998.

[10] C. Durand, *Symbolic and Numerical Techniques for Constraint Solving*, Ph.D. thesis, Purdue University, West Lafayette, IN, 1998.
[11] H. Edelsbrunner and E.P. Mücke, *Simulation of simplicity: A technique to cope with degeneracy*, ACM Trans. Graph., 9 (1990), pp. 43–72.
[12] H. Edelsbrunner and M. Sharir, *The maximum number of ways to stab n convex nonintersecting sets in the plane is* $2n-2$, Discrete Comput. Geom., 5 (1990), pp. 35–42.
[13] S. Fortune, *Polynomial root finding using iterated eigenvalue computation*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, London, ON, Canada, 2001, pp. 121–128.
[14] W. Fulton, *Introduction to Intersection Theory in Algebraic Geometry*, 2nd ed., CBMS Reg. Conf. Ser. Math. 54, AMS, Providence, RI, 1996.
[15] G.-M. Greuel, G. Pfister, and H. Schönemann, *Singular Version* 2.0. *A Computer Algebra System for Polynomial Computations*, Centre for Computer Algebra, University of Kaiserslautern, Kaiserslautern, Germany, 2001; http://www.singular.uni-kl.de .
[16] W. Hodge and D. Pedoe, *Methods of Algebraic Geometry*, Vol. 2, Cambridge University Press, Cambridge, UK, 1952.
[17] C.M. Hoffmann and B. Yuan, *On spatial constraint solving approaches*, in Proc. 3rd Workshop on Automated Deduction in Geometry (Zürich), Lecture Notes in Comput. Sci. 2061, Springer-Verlag, Berlin, 2000, pp. 1–15.
[18] H. Knörrer, *Geometrie*, Vieweg, Braunschweig, 1996.
[19] T. Lewis, B. von Hohenbalken, and V. Klee, *Common supports as fixed points*, Geom. Dedicata, 60 (1996), pp. 277–281.
[20] I.G. Macdonald, *private communication*.
[21] I.G. Macdonald, J. Pach, and T. Theobald, *Common tangents to four unit balls in* $\mathbb{R}^3$, Discrete Comput. Geom., 26 (2001), pp. 1–17.
[22] M. Marcus, *Finite Dimensional Multilinear Algebra, Part* I, Marcel Dekker, New York, 1973.
[23] G. Megyesi, *Lines tangent to four unit spheres with affinely dependent centres*, Discrete Comput. Geom., 26 (2001), pp. 493–497.
[24] J. Milnor, *Singular Points of Complex Hypersurfaces*, Ann. of Math. Stud. 61, Princeton University Press, Princeton, NJ, 1968.
[25] D. Pedoe, *A Course of Geometry for Colleges and Universities*, Cambridge University Press, London, 1970.
[26] M. Pellegrini, *Ray shooting and lines in spaces*, in The CRC Handbook of Discrete and Computational Geometry, J.E. Goodman and J. O'Rourke, eds., CRC Press, Boca Raton, FL, 1997, pp. 599–614.
[27] M. Pellegrini and P.W. Shor, *Finding stabbing lines in* 3*-space*, Discrete Comput. Geom., 8 (1992), pp. 191–208.
[28] M. Penna and R. Patterson, *Projective Geometry and Its Applications to Computer Graphics*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
[29] M. Reid, *Undergraduate Algebraic Geometry*, London Math. Soc. Stud. Texts 12, Cambridge University Press, Cambridge, UK, 1988.
[30] F. Ronga, A. Tognoli, and T. Vust, *The number of conics tangent to* 5 *given conics: The real case*, Rev. Mat. Univ. Complut. Madrid, 10 (1997), pp. 391–421.
[31] G. Salmon, *A Treatise on the Analytic Geometry of Three Dimensions*, Vol. 1: German translation by W. Fiedler, Teubner-Verlag, Leipzig, 1863.
[32] E. Schömer, J. Sellen, M. Teichmann, and C. Yap, *Smallest enclosing cylinders*, Algorithmica, 27 (2000), pp. 170–186.
[33] J.M. Selig, *Geometrical Methods in Robotics*, Springer-Verlag, New York, 1996.
[34] F. Sottile, *Enumerative geometry for the real Grassmannian of lines in projective space*, Duke Math. J., 87 (1997), pp. 59–85.
[35] F. Sottile, *From enumerative geometry to solving systems of polynomial equations with Macaulay* 2, in Computations in Algebraic Geometry with Macaulay 2, Algorithms Comput. Math. 8, D. Eisenbud, D. Grayson, M. Stillman, and B. Sturmfels, eds., Springer-Verlag, New York, 2001, pp. 101–129.
[36] J. Stolfi, *Oriented Projective Geometry*, Academic Press, Boston, 1991.
[37] J. Verschelde, *PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Software, 25 (1999), pp. 251–276.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1229–1254

QUANTUM CIRCUITS THAT CAN BE SIMULATED CLASSICALLY IN POLYNOMIAL TIME*

LESLIE G. VALIANT†

Abstract. A model of quantum computation based on unitary matrix operations was introduced by Feynman and Deutsch. It has been asked whether the power of this model exceeds that of classical Turing machines. We show here that a significant class of these quantum computations can be simulated classically in polynomial time. In particular we show that two-bit operations characterized by 4×4 matrices in which the sixteen entries obey a set of five polynomial relations can be composed according to certain rules to yield a class of circuits that can be simulated classically in polynomial time. This contrasts with the known universality of two-bit operations and demonstrates that efficient quantum computation of restricted classes is reconcilable with the Polynomial Time Turing Hypothesis. The techniques introduced bring the quantum computational model within the realm of algebraic complexity theory. In a manner consistent with one view of quantum physics, the wave function is simulated deterministically, and randomization arises only in the course of making measurements. The results generalize the quantum model in that they do not require the matrices to be unitary. In a different direction these techniques also yield deterministic polynomial time algorithms for the decision and parity problems for certain classes of read-twice Boolean formulae. All our results are based on the use of gates that are defined in terms of their graph matching properties.

Key words. quantum computation, Turing Hypothesis, matchgates, polynomial time simulation

AMS subject classifications. 81P68, 68Q10, 68Q15

PII. S0097539700377025

1. Background.

The now classical theory of computational complexity is based on the computational model proposed by Turing [33] augmented in two ways: On the one hand, random operations are often allowed in addition to the originally proposed deterministic ones [24]. On the other hand, the number of computational steps is restricted to be at most polynomial in the input length, which allows such striking phenomena as NP-completeness to be expressed [10]. Taken together, the resulting theory has provided strong circumstantial evidence for the robustness of the resulting model [16]. The following statement is a variant of how this robustness is sometimes expressed and might be called the Polynomial Time Turing Hypothesis:

> *Any physical computing device can be simulated by a randomizing Turing machine that, as the input instances vary, takes a number of steps that grows as at most some fixed polynomial in the quantity $T + S + E$, where T, S, and E are the time, space, and energy used by the computing device.*

While no generally accepted counterexample is known, this hypothesis has, and can be expected to continue to, come under repeated challenge. The fundamental sources of challenge are functions that arise in accepted laws of physics but which are

*Received by the editors August 21, 2000; accepted for publication (in revised form) December 8, 2001; published electronically April 26, 2002. This research was supported in part by grants NSF-CCR-95-04436, NSF-CCR-98-77049, ARO-DAAL-03092-G-0115, and by the National Security Agency (NSA) and Advanced Research and Development Activity (ARDA) under Army Research Office (ARO) contract DAAD19-01-1-0506. A preliminary version of this paper appeared in *Proceedings of the 33rd ACM Symposium on Theory of Computing*, ACM Press, New York, 2001, pp. 114–123.
http://www.siam.org/journals/sicomp/31-4/37702.html

†Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (valiant@deas.harvard.edu).

not known to be polynomial time computable classically. Such a challenge has been recognized since the late 1970s for the matrix permanent (i.e., the determinant with all positive signs). On the one hand, the wave function for indistinguishable bosons can be expressed as a permanent (e.g., [29]). On the other hand, this function is known to be $\#P$-complete [34], or, in other words, as hard as counting the number of solutions of NP-complete problems and therefore at least as hard as the decision question for NP-complete problems. An approach to resolving this particular challenge is proposed by Troyansky and Tishby [32]. The second such source, the one that has received the most attention and is the subject of this paper, is the class of functions that can be computed as compositions of fixed size unitary operations and first discussed in detail by Deutsch [12]. A third source of challenge is the Jones polynomial, as discussed by Freedman [19].

Each of these sources poses a challenge in implying that among the following three statements at least one must be true: (a) the Polynomial Time Turing Hypothesis is false, (b) some physical reason excludes the possibility of a physical device evaluating appropriate instances of the function, or (c) the supposed hard function can be computed classically in polynomial time by a yet undiscovered algorithm. Identifying the statement or statements in such a list that are true would clearly add to our knowledge and offers a valid scientific goal. This paper is concerned with an instance of (c).

The unitary matrix model can be traced to Benioff [2], who observed that classical binary Boolean operations could be formulated as fixed size unitary matrix transformations. Subsequently, Feynman [17] and Deutsch [12] suggested that unitary transformations as realized by quantum devices offer a model of computation that may violate the Polynomial Time Turing Hypothesis and asked whether it did. Bernstein and Vazirani [3] showed how uniformity over all input sizes could be incorporated in the quantum computation model and hence that quantum polynomial time class BQP could be defined. While it is not generally believed that this class contains all of #P or even NP, Shor [30] was able to show that two particular problems, namely integer factorization and discrete logarithms, which are not known to be polynomial time computable, do lie in BQP.

In this model of quantum computation a gate operating on k Boolean bits is represented as a $2^k \times 2^k$ unitary matrix. A unitary matrix U is any matrix with complex number entries such that $UU^{*t} = I$, where I is the identity matrix, U^* is the complex conjugate of U, and U^{*t} is the transpose of U^*. We note, however, that our constructions are all based on general matrix properties in which unitarity plays *no* essential part, and, in that sense, our treatment generalizes the quantum computational model (cf. [18]).

Some basic results on this quantum computational model can be found in [12, 13, 14, 15, 27]. Most notably, it is known that there exist two-bit gates which, in conjunction with arbitrary one-bit gates, can approximate arbitrary unitary matrices. On the other hand, it is easy to see that circuits composed of one-bit gates alone can be simulated deterministically in polynomial time since it is possible to follow the evolution of each bit independently. A more interesting class, where only a discrete set of gates is allowed but following the evolution of a small set of basis operators is sufficient, is attributed to Knill [20].

Our approach can be viewed as a branch of algebraic complexity theory [7, 8, 31, 34] in which computation gates are simulated by graph matching properties. In [34, 35] this approach is developed in the context of the matrix permanent. In the

current paper we use encodings by matrix functions that, in contrast, are known to be polynomial time computable. In particular we use the following three functions: the Pfaffian, the decision problem for matchings, and the parity problem for matchings.

2. Pfaffians. Here we shall describe some standard graph-theoretic notions and their relation to the Pfaffian of a matrix. In the subsequent section we shall introduce a more general notion, that of a Pfaffian Sum. In later sections we shall go on to show that certain quantum circuits can be represented as graphs in such a way that the Pfaffian Sum of the adjacency matrix of the graph corresponds to the basic computational properties of the corresponding circuit. Further, the probability of the circuit evolving from one state to another will be expressible as a Pfaffian Sum.

A weighted undirected graph, or simply a *graph*, G is a triple (V, E, W), where V is a set of *vertices* each represented by a distinct positive integer, E is a set of *edges* or unordered pairs (i, j) of the vertices $i, j \in V$, and W is the set of *weights*, each weight $w(i, j)$ corresponding to the edge $(i, j) \in E$. For example, $V = \{1, 2, 3\}$, $E = \{(1, 2), (2, 3), (1, 3)\}$, $w(1, 2) = w(2, 3) = w(1, 3) = 2$ is the complete graph on three vertices in which every edge has weight 2.

An $n \times n$ matrix C is *skew-symmetric* if for all i, j $(1 \leq i, j \leq n)$ $C(i, j) = -C(j, i)$. The *matrix of the graph* $G = (V, E, W)$ where $V = \{1, 2, \ldots, n\}$ is the $n \times n$ matrix $M(G)$ where the (i, j) entry $M(G)(i, j)$ is defined to equal

(i) $w(i, j)$ if $i < j$,
(ii) $-w(i, j)$ if $i > j$, and
(iii) 0 otherwise.

In the more general case that $V = \{k_1, k_2, \ldots, k_n\}$ where $k_1 < k_2 < \cdots < k_n$, weight $w(k_i, k_j)$ replaces $w(i, j)$ in (i) and (ii) in this definition. For brevity we shall abbreviate $M(G)$ by G whenever it is clear that a matrix is intended.

The Pfaffian of an $n \times n$ skew-symmetric matrix C is defined to be zero if n is odd, one if $n = 0$, and if n is even with $n = 2k$ and $k > 0$ then it is defined as

$$\mathrm{Pf}(C) = \sum_{\pi} \epsilon_\pi w(i_1, i_2) w(i_3, i_4) \ldots w(i_{2k-1}, i_{2k}),$$

where

(i) $\pi = [i_1, i_2, i_3, \ldots, i_{2k}]$ is a permutation on $[1, 2, \ldots, n]$,
(ii) summation is over all such permutations π where further
$i_1 < i_2, i_3 < i_4, \cdots, i_{2k-1} < i_{2k}$, and
$i_1 < i_3 < i_5 < \cdots < i_{2k-1}$, and
(iii) $\epsilon_\pi \in \{-1, 1\}$ is the sign of the permutation; i.e., it is -1 or $+1$ according to whether the number of transpositions or swaps of pairs of distinct elements i_j, i_k needed to reorder π to the identity permutation is odd or even. (An equivalent definition in this context is that it is the sign or parity of the number of overlapping pairs, where a pair of edges (i_{2r-1}, i_{2r}), (i_{2s-1}, i_{2s}) is *overlapping* iff $i_{2r-1} < i_{2s-1} < i_{2r} < i_{2s}$ or $i_{2s-1} < i_{2r-1} < i_{2s} < i_{2r}$. Note that it is implicit here that $i_{2r-1} < i_{2r}$ and $i_{2s-1} < i_{2s}$.)

A *matching* $E^* \subseteq E$ of G is a set of edges such that if (i, j), (r, s) are distinct edges in E^*, then i, j, r, s are all distinct vertices. In a graph with an even number $2k$ of nodes a matching E^* is *perfect* if it contains k edges. (In other words every $i \in V$ is an endpoint of, or is *saturated* by, some edge in E^*.)

We shall use the following graph-theoretic interpretation of the Pfaffian: If C is the matrix of the graph G, then there is a one-on-one correspondence between monomials in the Pfaffian and perfect matchings in G. The monomial $w(i_1, i_2)$

$w(i_3, i_4) \ldots w(i_{2k-1}, i_{2k})$ in $\mathrm{Pf}(G)$ corresponds to the perfect matching $\{(i_1, i_2), (i_3, i_4), \ldots, (i_{2k-1}, i_{2k})\}$ in G. The coefficient ϵ_π of this monomial is the parity of the number of overlapping pairs of edges, in the sense defined above.

The following fact, due to Cayley [9] (see also [5, Theorem 9.5.2]) relates the Pfaffian to the determinant.

THEOREM 1. *For any skew-symmetric matrix C*

$$\mathrm{Det}\,(C) = (\mathrm{Pf}\,(C))^2.$$

It is known that for matrices with elements from any field the complexity of computing the determinant is to within a constant multiplicative factor the same as that of matrix multiplication [31]. Further, for $n \times n$ matrices this cost is known to be upper bounded by $O(n^\alpha)$, where $\alpha < 2.38$ [11]. It follows that the square of the Pfaffian is polynomial time computable, and for complex matrices so is the square of the modulus $|\mathrm{Pf}(C)|^2 = |(\mathrm{Pf}(C))^2|$. Efficient parallel algorithms for these functions also follow from similar results for the determinant [4].

For an $n \times n$ matrix C we call a set $A = \{i_1, i_2, \ldots, i_r\} \subseteq \{1, 2, \ldots, n\}$ an *index set*. Further, we denote by $C[i_1, i_2, \ldots, i_r]$ or $C[A]$ the $(n-r) \times (n-r)$ matrix obtained by deleting the i_jth row and column from C for all the r values of j $(1 \leq j \leq r)$.

The following fact is useful for proving negative results about how the Pfaffians of different submatrices of a matrix constrain each other.

THEOREM 2. *For any skew-symmetric $n \times n$ matrix C and any distinct $i < j < k < l \in \{1, 2, \ldots, n\}$*

$$\begin{aligned}\mathrm{Pf}(C)\mathrm{Pf}(C[i,j,k,l]) &- \mathrm{Pf}(C[i,j])\mathrm{Pf}(C[k,l]) \\ &+ \mathrm{Pf}(C[i,k])\mathrm{Pf}(C[j,l]) - \mathrm{Pf}(C[i,l])\mathrm{Pf}(C[j,k]) = 0\end{aligned}$$

and

$$\begin{aligned}-\mathrm{Pf}(C[j,k,l])\mathrm{Pf}(C[i]) &+ \mathrm{Pf}(C[i,k,l])\mathrm{Pf}(C[j]) \\ &- \mathrm{Pf}(C[i,j,l])\mathrm{Pf}(C[k]) + \mathrm{Pf}(C[i,j,k])\mathrm{Pf}(C[l]) = 0.\end{aligned}$$

Proof. These can be derived from the Grassmann–Plücker identity for Pfaffians [36]. □

3. Pfaffian Sums. We start by defining the notion of a Pfaffian Sum, which expresses the Pfaffian summed over exponentially many minors of a matrix. In graph-theoretic terms the Pfaffian Sum generalizes the Pfaffian in that it has terms not only for the perfect matchings but for matchings of all sizes. This generalization is necessary where we wish to simulate the more general quantum gates. The main result in this section is that the Pfaffian Sum can be expressed in terms of the standard Pfaffian, and its square is therefore also computable in polynomial time.

DEFINITION. *The* Pfaffian Sum *of an $n \times n$ skew-symmetric matrix C is a polynomial over indeterminates $\lambda_1 \ldots \lambda_n$ such that*

$$\mathrm{PfS}(C) = \sum_A \left(\prod_{i \in A} \lambda_i \right) \mathrm{Pf}(C[A]).$$

Summation here is over the various 2^n principal minors obtained from C by deleting some subset A of the indices, including the empty set. In this paper we shall need only the instances in which each λ_i is fixed to be 0 or 1. The i for which $\lambda_i = 0$

can be thought of as the *unomittable* indices and those with $\lambda_i = 1$ as the *omittable* indices. Then for this $(0,1)$-*case* the Pfaffian Sum is simply the sum of the $\mathrm{Pf}(C[A])$ over those A that contain only omittable indices. This is the only case that we shall use.

We note that one can similarly define a *Determinant Sum*

$$\mathrm{DetS}(C) = \sum_A \left(\prod_{i \in A} \lambda_i \right) \mathrm{Det}\,(C[A]).$$

It is easy to verify that for any matrix C

$$\mathrm{DetS}(C) = \mathrm{Det}\,(C + \Delta),$$

where $\Delta(i,j) = 0$ if $i \neq j$, and $\Delta(i,i) = \lambda_i$ for every i. Thus a summed determinant is as easy to compute as a single determinant. We shall now show that the same holds for the Pfaffian. We define the $n \times n$ matrix $\Lambda^{(n)}$ as follows:

$$\Lambda^{(n)}(i,j) = \begin{cases} (-1)^{j-i+1}\lambda_i\lambda_j & \text{if} \quad i < j, \\ (-1)^{i-j}\lambda_i\lambda_j & \text{if} \quad i > j, \\ 0 & \text{if} \quad i = j. \end{cases}$$

Also for an $n \times n$ matrix C we define C^+ to be the $(n+1) \times (n+1)$ matrix of which the first n rows and columns equal C itself, and the $(n+1)$st row and column entries are all zero.

Pfaffian Sum Theorem. *For an $n \times n$ skew-symmetric matrix C*

$$\mathrm{PfS}(C) = \begin{cases} \mathrm{Pf}(C + \Lambda^{(n)}) & \textit{if} \quad n \quad \textit{is even,} \\ \mathrm{Pf}(C^+ + \Lambda^{(n+1)}) & \textit{with } \lambda_{n+1} \textit{ set to } 1, \quad \textit{if } n \quad \textit{is odd.} \end{cases}$$

Lieb [25] gave a proof for the instance of this relationship in which the λ_i are equal for all i, and n is even. One can derive our result from that instance. We shall, however, give a direct proof of this fact that makes the associated combinatorial structures more explicit.

Lemma 1. *If matrix C' is obtained from skew-symmetric matrix C by first interchanging the elements of rows i and j, and then interchanging the elements of columns i and j, then* $\mathrm{Pf}(C) = -\mathrm{Pf}(C')$.

Proof. The specified matrix operation clearly leaves the determinant function unchanged. By Theorem 1 it follows that the operation either leaves the Pfaffian invariant or multiplies it by -1. Assume without loss of generality that $i < j$. Then each monomial in $\mathrm{Pf}(C)$ that contains $C(i,j)$ as a factor will have an identical monomial in $\mathrm{Pf}(C')$ except that the latter will contain $C'(i,j)$, where $C'(i,j) = -C'(j,i) = -C(i,j)$. Therefore at least some terms in $\mathrm{Pf}(C)$ and $\mathrm{Pf}(C')$ have opposite signs, and it follows that they all must have opposite signs. □

Lemma 2. *If skew-symmetric matrices C, C' are identical except that for some $i \in \{1, 2, \ldots, n\}$ for all j $C(i,j) = -C'(i,j)$ and $C(j,i) = -C'(j,i)$, then* $\mathrm{Pf}(C) = -\mathrm{Pf}(C')$.

Proof. The monomials of $\mathrm{Pf}(C)$, $\mathrm{Pf}(C')$ can be paired so that they are identical except that they contain one factor $C(k,i)$ or $C'(k,i)$, respectively, or one factor of $C(i,k)$ or $C'(i,k)$, respectively, for some k. In either case the two factors will have opposite signs by assumption, and hence so will the monomials. □

LEMMA 3. *Let $\Lambda^{(n)}$ be the $n \times n$ skew-symmetric matrix defined above with n even. Then*

$$\mathrm{Pf}(\Lambda^{(n)}) = \lambda_1 \ldots \lambda_n.$$

Proof. We prove this by induction on even values of n. Clearly $\mathrm{Pf}(\Lambda^{(2)}) = \lambda_1\lambda_2$. Now assume that the result holds for some even $n-2$. Consider the monomials in the expansion for $\Lambda^{(n)}$. In all the monomials that contain the element $\Lambda^{(n)}(n-1,n) = \lambda_{n-1}\lambda_n$ the complementary factors clearly sum to $\Lambda^{(n-2)}$, which by induction equals $\lambda_1 \ldots \lambda_{n-2}$. It is therefore sufficient to observe that all the other monomials in $\Lambda^{(n)}$ cancel in pairs: Any such other monomial contains a product $\Lambda^{(n)}\ (i,n-1)\Lambda^{(n)}(j,n)$ and has a companion monomial that is identical except that this product is replaced by $\Lambda^{(n)}(i,n)\Lambda^{(n)}(j,n-1)$, which is equal in value but gives rise to the opposite sign because of the extra transposition $(n-1,n)$. □

Proof of theorem. First we assume that n is even. For some fixed $A \subseteq \{1,\ldots,n\}$ consider the monomial set N of $\mathrm{Pf}(C + \Lambda^{(n)})$ that involve λ elements in all rows and columns indexed by A, and involve C elements in the remaining rows and columns. We claim that the monomials in N sum to

$$\mathrm{Pf}(C[A])\mathrm{Pf}(\Lambda^{(A)}), \tag{1}$$

where $\Lambda^{(A)}$ is the $\Lambda^{|A|}$ matrix in which for each λ_i there has been substituted the indeterminate λ_j, where j is the ith smallest element of A. By applying Lemma 3 to the second term here and summing over all choices of A, the theorem then follows.

To establish this claim we consider a sequence of σ transpositions or swaps on the indices (i.e., row and column numbers) of the original matrix that results in the indices A migrating to $n-|A|+1$, $n-|A|+2,\ldots,n$, and an accompanying process of sign changes to the elements described below. We claim that the following hold for every A:

(a) The sequence of transpositions and sign changes to the elements causes all monomials in N to remain unchanged in sign in the Pfaffian.
(b) The matrix that is the outcome of the process has the C entries in the first $n-|A|$ rows/columns and these contribute a multiplicand of $\mathrm{Pf}(C([A])$ (with positive sign) to the sum of N.
(c) The matrix that is the outcome of the process has the λ entries in the last $|A|$ rows/columns and these contribute a multiplicand of $\mathrm{Pf}(\Lambda^{(A)})$ to the sum of N.

The conjunction of these three claims clearly implies that the quantity (1) above equals the original sum of N as needed.

For (a) we note that by Lemma 1 the number of sign changes to the Pfaffian, and hence the monomials in N, caused by σ transpositions is $(-1)^\sigma$, and we shall observe below that the effect of the sign changes to be specified to the elements is a compensatory $(-1)^\sigma$ factor. Claim (b) is true by the definition of $\mathrm{Pf}(C[A])$. With regard to (c) we first observe that while the elements $\Lambda^{(n)}(i,j)$ and hence $\Lambda^{(A)}(i,j)$ alternate in sign as j varies and i is fixed, this is not necessarily the case for the pattern of the λ entries that contribute to N in the matrix *before* the transpositions. For example, entries in adjacent columns after the transpositions will have the same sign if the indices of the columns before the transpositions differ by an even number. It will be useful to consider the particular sequence of transpositions where the largest indices in A are moved up to fill the positions $n, n-1,\ldots,n-|A|+1$, in this order, and each one is moved by a succession of $(i,i+1)$ transpositions. Clearly the number

of such transpositions has the same parity as the number of indices that need to be moved by an odd number of such neighbor transpositions. However, the indices j that need an odd number of such transpositions to reach their destination are exactly those for which the signs of the entries are different from those of the entries of $\Lambda^{(n)}$ in the destination row/column. What we consider, therefore, is the process in which for just these indices we will change the sign of all the entries in that row and column in the course of moving it. The effect of these sign changes to the elements in a row and column is, by Lemma 2, to change the sign of the Pfaffian. We conclude that this process of transpositions and sign changes would result in the λ_i entries having moved to the $|A|$ highest indices, forming a $\Lambda^{(A)}$ matrix there, as required for claim (c), and that the effect of the sign changes on the monomial set N in the Pfaffian is a compensatory $(-1)^\sigma$ factor as required for claim (a).

Now we consider the case that n is odd. Since $n+1$ is now even it is clear from the previous analysis that $\mathrm{Pf}(C^+ + \Lambda^{(n+1)})$ will again equal the Pfaffian Sum, but now summation can be restricted to those A that contain n since otherwise the $\mathrm{Pf}(C^+[A])$ contribution vanishes. Hence this Pfaffian sum is equal to that of the original C if λ_{n+1} is set to 1. □

4. Matchgates. We shall simulate each quantum or other gate by what we call a matchgate.

A *matchgate* Γ is a quadruple (G, X, Y, T), where G is a graph (V, E, W), $X \subseteq V$ is a set of *input* vertices, $Y \subseteq V$ is a set of *output* vertices, and $T \subseteq V$ is a set of *omittable* vertices such that (i) X, Y, and T are all disjoint, and (ii) for all $i \in T$, if $j \in X$, then $j < i$ and if $j \in Y$, then $j > i$.

The matchings we consider will be those that saturate all the unomittable nodes, i.e., $V - T$, and also some, possibly empty, subset of T. Whenever we refer to the Pfaffian Sum of a matchgate fragment, such as G' below, we shall assume the substitutions $\lambda_i = 1$ if $i \in T$ and $\lambda_i = 0$ otherwise.

We call $X \cup Y$ the *external* nodes. For $Z \subseteq X \cup Y$ we define the *character* $\chi(\Gamma, Z)$ of Γ with respect to Z to be the product

$$\mu(\Gamma, Z)\mathrm{PfS}(G'),$$

where (a) $G' = (V - Z, E', W')$, where further E' is the restriction of E to edges with both endpoints in $V - Z$, and W' is the corresponding restriction of W, and (b) the *modifier* $\mu(\Gamma, Z) \in \{-1, 1\}$ counts the parity of the number of overlaps between matching edges in E' and external edges. The external edges are the edges that link each matchgate to the rest of the circuit. We consider there to exist one external edge from each node in $X \cap Z$ and from each node in $Y \cap Z$. The other endpoint of each of the former is some node of lower index than any in V and of each of the latter is some node of index higher than any in V. Figure 1 gives an illustrative example.

The character $\chi(\Gamma, Z)$, therefore, takes into account overlaps between the internal edges of matchgate Γ and the external edges that link its external nodes to the rest of the circuit. The significance of condition (ii) in the definition of matchgates is that it guarantees that the modifier $\mu(\Gamma, Z)$ is always well defined: for any fixed Z the external edges that arise are uniquely defined, but it has to be guaranteed that the parity of the overlap of any one such external edge with *every* matching of E' that saturates all the unomittable nodes is the same. Condition (ii) achieves this by not allowing an omittable node in the gate to be numbered intermediate between the endpoints of an external edge. (That case might produce different overlap parity for the given external edge and the various internal matchings depending on whether the

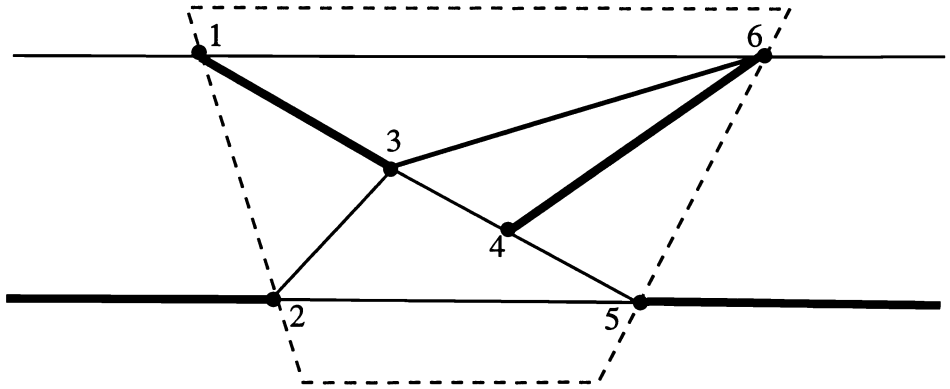


FIG. 1. *Illustration of a matchgate Γ with inputs $X = \{1, 2\}$ and outputs $Y = \{5, 6\}$, with the external nodes $Z = \{2, 5\}$ matched. The character $\chi(\Gamma, Z)$ is the product of the modifier $\mu(\Gamma, Z)$ and of the Pfaffian Sum of the matchgate after nodes Z have been removed. Now $\mu(\Gamma, Z) = 1$ since the externally matched edges (i.e., those from nodes 2 and 5) overlap exactly twice with the internally matched edges, whether the edges $\{1, 3\}$ and $\{4, 6\}$ as shown in this example, or any alternative ones, such as $\{1, 6\}, \{3, 4\}$.*

omittable node was in the matching.) To verify this note that if for $i \in X \cap Z$ there are r nodes $j < i$ where $j \in V - Z$, then the parity of the overlap of the external edge from i with the internal edges is the parity of r.

We define the character matrix $\chi(\Gamma)$ of $\Gamma = (G, X, Y, T)$ as a $2^{|X|} \times 2^{|Y|}$ matrix where the rows represent the subsets of the inputs X, and the columns the subsets of the outputs Y, and the entries the corresponding values of $\chi(\Gamma, Z)$ for the various choices of Z. Matchgates with $|X| = |Y| = k$ can then be regarded as square matrix transformations defined by the character matrix. For example, $k = 1$ corresponds to one-bit 2×2 matrix transformations and $k = 2$ corresponds to two-bit 4×4 transformations. In all cases we need to specify a correspondence between subsets of X and the rows of the matrix and another correspondence between subsets of Y and the columns of the matrix. We insist that there is the following consistency between the row and column orderings. We assume that there is a bijection f that maps the elements of X to elements of Y such that if the ith row corresponds to subset X' of X and the ith column to subset Y' of Y, then Y' is the image of X' under f. The *character matrix* $\chi(\Gamma)$ is therefore defined as a matrix of the elements $\chi(\Gamma, Z)$ with some such consistent row/column ordering.

The character matrix defines a linear transformation from linear combinations of the $2^{|X|}$ components $\underline{x} \in \{0, 1\}^{|X|}$ to linear combinations of the $2^{|Y|}$ components $\underline{y} \in \{0, 1\}^{|Y|}$. The character matrix can also be thought of as determining the nondeterministic steps allowed by a matchgate. If vector $\underline{x} \in \{0, 1\}^{|X|}$ is specified for the bits of X, then for each possible $\underline{y} \in \{0, 1\}^{|Y|}$ the corresponding element in the $2^{|I|} \times 2^{|Y|}$ character matrix gives the *value* to be associated with nondeterministic output $\underline{y}$ for input $\underline{x}$. The overall computations of the circuits composed of these matchgates are like counting Turing machines [34] or quantum computations [3, 13] in that values may be multiplied over the steps of individual nondeterministic computation branches and then added over all possible branches to obtain the numerical value of the computation.

First we shall show that all one-bit or 2×2 matrix transformations can be simulated by matchgates.

PROPOSITION 1. *For any field F and any 2×2 matrix B of elements from F there exists a four node matchgate Γ with weights drawn from F such that $\chi(\Gamma) = B$.*

Proof. Consider the matchgate (G, X, Y, T), where G is the complete graph on nodes $V = \{1, 2, 3, 4\}$, $X = \{1\}$, $Y = \{4\}$, $T = \{2\}$, and W is as variously specified below.

We shall identify row 1 of the matrix with $\emptyset$ and row 2 with $\{1\}$. Similarly we identify column 1 with $\emptyset$ and column 2 with $\{4\}$. Let us abbreviate $B(1,1) = a$, $B(1,2) = b$, $B(2,1) = c$, and $B(2,2) = d$. Now it is immediate from the definitions that the modifier is always 1 since in this case external edges cannot overlap with internal ones. The character is simply the (0,1)-Pfaffian Sum for the gate with the external nodes corresponding to the variables set to 1 removed and with T as the omittable nodes. In particular,

$\chi(\Gamma, \emptyset) = w(2,3)w(1,4) - w(1,3)w(2,4) + w(1,2)w(3,4)$,
$\chi(\Gamma, \{1\}) = w(3,4)$,
$\chi(\Gamma, \{4\}) = w(1,3)$, and
$\chi(\Gamma, \{1,4\}) = w(2,3)$.

Each of these statements can be verified by inspection. For example, in the second of these, since one of the four nodes, namely node 1, is matched externally, and there is one omittable node, namely 2, in the matchgate, the only matching is the one that involves the remaining two nodes, namely $\{3, 4\}$.

First suppose that at least one of b, c, or d is nonzero. Then by fixing $w(1,3) = b$, $w(3,4) = c$, $w(2,3) = d$, and setting the remaining weights so that $-bw(2,4) + cw(1,2) + dw(1,4) = a$, the four components of B can be set to the arbitrary values a, b, c, and d as desired.

In the special case that $b = c = d = 0$ we consider the same matchgate but make $T = \emptyset$. Then setting $w(2,3) = 0$ ensures that $b = c = d = 0$. The expression for $\chi(\Gamma, \emptyset)$ is unchanged and therefore by setting the remaining weights one can set $B(1,1)$ to the arbitrary value of a. □

To model two-bit or 4×4 matrices we consider two-bit matchgates, i.e., where $|X| = 2$, $|Y| = 2$. The characters of these can be viewed as above, with the rows representing subsets of X and the columns subsets of Y.

PROPOSITION 2. *For any field F and any 4×4 matrix B of elements from F where all the off diagonal elements are zero and $B(1,1)B(4,4) = B(2,2)B(3,3)$ there exists a matchgate Γ with weights drawn from F such that $\chi(\Gamma) = B$.*

Proof. Consider $V = \{1, 2, 3, 4, 5, 6\}$, $X = \{1, 2\}$, $Y = \{5, 6\}$, and $T = \emptyset$. The rows of the matrices have ordering $\emptyset$, $\{1\}$, $\{2\}$, $\{1, 2\}$, and the columns have ordering $\emptyset$, $\{6\}$, $\{5\}$, $\{5, 6\}$.

First suppose that $B(4,4) \neq 0$. Then let the only nonzero weights be $w(1,6) = B(3,3)/B(4,4)$, $w(2,5) = B(2,2)/B(4,4)$, and $w(3,4) = B(4,4)$. It can be verified that this matchgate has the required character.

If $B(4,4) = 0$, then we have various special cases: (a) $w(1,6)$, $w(2,3)$, $w(4,5) \neq 0$ solves the case $B(1,1) \neq 0$, $B(2,2) \neq 0$, $B(3,3) = 0$, $B(4,4) = 0$. (b) $w(2,5)$, $w(1,3)$, $w(4,6) \neq 0$ solves the case $B(1,1) \neq 0$, $B(2,2) = 0$, $B(3,3) \neq 0$, $B(4,4) = 0$. (c) $w(2,3)$, $w(4,5) \neq 0$ solves the case $B(1,1) = 0$, $B(2,2) \neq 0$, $B(3,3) = 0$, $B(4,4) = 0$. (d) $w(1,3)$, $w(4,6) \neq 0$ solves the case $B(1,1) = 0$, $B(2,2) = 0$, $B(3,3) \neq 0$, $B(4,4) = 0$. To solve the remaining case of $B(1,1) \neq 0$, $B(2,2) = 0$, $B(3,3) = 0$, $B(4,4) = 0$ we can consider the eight node matchgate with $X = \{1, 2\}$, $Y = \{7, 8\}$,

and nonzero edges $w(1,3)$, $w(2,4)$, $w(5,7)$, and $w(6,8)$. □

We now turn to more general 4×4 matrices.

PROPOSITION 3. *For any field F and any 4×4 matrix B of elements from F satisfying the nine equations $B(1,2) = B(1,3) = B(2,1) = B(2,4) = B(3,1) = B(3,4) = B(4,2) = B(4,3) = 0$, and $B(1,1)B(4,4) - B(2,2)B(3,3) = B(1,4)B(4,1) - B(2,3)B(3,2)$, there exists a matchgate Γ with weights drawn from F such that $\chi(\Gamma) = B$, provided $B(4,4) \neq 0$.*

Proof. Consider $V = \{1,2,3,4,5,6\}$, $X = \{1,2\}$, $Y = \{5,6\}$, $T = \emptyset$, and row/column ordering as in Proposition 2. We let $t = 1/B(4,4)$ and consider the following set of nonzero weights: $w(1,6) = tB(3,3)$, $w(2,5) = tB(2,2)$, $w(1,2) = tB(1,4)$, $w(5,6) = tB(4,1)$, $w(1,5) = -tB(3,2)$, $w(2,6) = -tB(2,3)$, $w(3,4) = B(4,4)$. □

Now consider the following set of what we call the *matchgate identities* for 4×4 character matrices:

$$
\begin{aligned}
&B(1,1)B(4,4) - B(2,2)B(3,3) - B(1,4)B(4,1) + B(2,3)B(3,2) = 0\\
&B(2,1)B(4,4) - B(2,2)B(4,3) - B(4,1)B(2,4) + B(2,3)B(4,2) = 0\\
&B(3,1)B(4,4) + B(3,3)B(4,2) - B(4,1)B(3,4) - B(3,2)B(4,3) = 0\\
&B(1,3)B(4,4) + B(3,3)B(2,4) - B(1,4)B(4,3) - B(2,3)B(3,4) = 0\\
&B(1,2)B(4,4) - B(2,2)B(3,4) - B(1,4)B(4,2) + B(3,2)B(2,4) = 0
\end{aligned}
$$

PROPOSITION 4. *For any field F and any set of values of the 11 entries of the 4×4 matrix B other than $R = \{B(1,1), B(1,2), B(1,3), B(2,1), B(3,1)\}$ such that $B(4,4) \neq 0$, there exists a matchgate Γ with weights drawn from F such that $\chi(\Gamma)$ equals B in the 11 entries other than R, and in each of the R entries $\chi(\Gamma)$ equals various polynomials in terms of these other 11 entries and $1/B(4,4)$. In particular, $B = \chi(\Gamma)$ satisfies the five matchgate identities.*

Proof. Consider the matchgate of Proposition 3 but modify it by making $T = \{4\}$ and adding four more nonzero edge weights: $w(1,3) = -B(3,4)$, $w(2,3) = B(2,4)$, $w(3,5) = B(4,2)$, $w(3,6) = -B(4,3)$. This is illustrated in Figure 2.

By inspection, for any $Z \subseteq X \cup Y$ one can write down $\chi(\Gamma, Z)$: If $\mid Z \mid$ is even, then all the matchings that contribute to $\chi(\Gamma, Z)$ saturate the omittable node 4 and do so necessarily with edge $\{3,4\}$. Hence this case reverts back to Proposition 3 since the newly added edges are all incident to node 3 and can, therefore, play no part. Hence in the corresponding eight positions in B we can regard seven as being arbitrary, and the eighth, namely $B(1,1)$, as being constrained by the first matchgate relation in terms of the other seven exactly, as in Proposition 3, since $B[4,4] \neq 0$.

We now consider the eight entries of $\chi(\Gamma, Z)$, where $|Z|$ is odd. By inspecting Figure 1 and recalling the definition of $\chi(\Gamma, Z)$ we obtain immediately that

$$
\begin{aligned}
&\chi(\Gamma, \{2,5,6\}) = B(3,4),\\
&\chi(\Gamma, \{1,5,6\}) = B(2,4),\\
&\chi(\Gamma, \{1,2,6\}) = B(4,2),\\
&\chi(\Gamma, \{1,2,5\}) = B(4,3),\\
&\chi(\Gamma, \{1\}) = tB(2,2)B(4,3) + tB(4,1)B(2,4) - tB(2,3)B(4,2),\\
&\chi(\Gamma, \{2\}) = -tB(3,3)B(4,2) + tB(4,1)B(3,4) + tB(3,2)B(4,3),\\
&\chi(\Gamma, \{5\}) = -tB(3,3)B(2,4) + tB(1,4)B(4,3) + tB(2,3)B(3,4),\\
&\chi(\Gamma, \{6\}) = tB(2,2)B(3,4) + tB(1,4)B(4,2) - tB(3,2)B(2,4).
\end{aligned}
$$

The first equality states simply that the arbitrary chosen value $B(3,4)$ is in fact the value of the character $\chi(\Gamma, \{2,5,6\})$ for the given matchgate, exactly as desired.

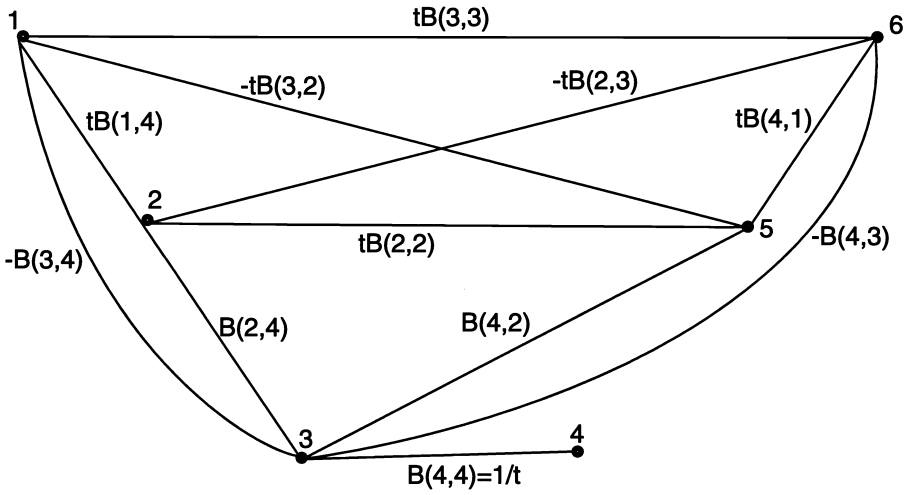


FIG. 2. *A matchgate with inputs* $\{1,2\}$ *and outputs* $\{5,6\}$*, designed to have character that equals the given matrix* B *in* 11 *of the* 16 *entries, as required by Proposition* 4. *The quantity* t *denotes* $1/B(4,4)$.

The same holds for $B(2,4), B(4,2)$, and $B(4,3)$. Hence, in addition to the seven entries in the character that we said could be fixed arbitrarily, here are a further four.

The last four statements describe the remaining four entries of the character in terms of the 11 arbitrarily fixed, and of $B(4,4)$, which, as we saw, is constrained by the first matchgate identity. These four statements assert nothing other than that if $B(2,1), B(3,1), B(1,3)$, and $B(1,2)$ do equal the corresponding character entries of the specified matchgate, then these quantities obey the last four matchgate identities, respectively. □

A related question is to characterize the subclass of the feasible character matrices that are also unitary. For matrices in which the only nonzero entries are those appearing in the first matchgate identity this is easy. We note that in that case restricting the matrix to rows/columns $\{1,4\}$ imposes a unitary constraint on the four remaining elements, as does the restriction to rows/columns $\{2,3\}$. However, unitary 2×2 matrices can be written as [5]

$$e^{i\alpha} \begin{bmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{bmatrix} \begin{bmatrix} \cos\gamma/2 & -\sin\gamma/2 \\ \sin\gamma/2 & \cos\gamma/2 \end{bmatrix} \begin{bmatrix} e^{-i\delta/2} & 0 \\ 0 & e^{i\delta/2} \end{bmatrix}, \tag{2}$$

where $\alpha, \beta, \gamma, \delta$ are real valued. In other words the 2×2 matrix with elements $B(1,1), B(4,1), B(1,4), B(4,4)$ can be written in this form with appropriate $\alpha_1, \beta_2, \gamma_1, \delta_1$, as can the 2×2 with elements $B(2,2), B(2,3), B(3,2), B(3,3)$ for appropriate $\alpha_2, \beta_2, \gamma_2, \delta_2$. However, the first matchgate identity is equivalent to saying that the difference of the determinants of these two 2×2 matrices is zero. However, the determinant of the expression (2) is $e^{i\alpha}$. It follows that, except for the restriction that $e^{i\alpha_1} - e^{i\alpha_2} = 0$, and $B(4,4) \neq 0$, the space of matchgate character matrices is spanned as the values of $\alpha_1, \beta_1, \gamma_1, \delta_1, \alpha_2, \beta_2, \gamma_2$, and δ_2 range over the reals.

THEOREM 3. *Given any matchgate Γ there exists another matchgate Γ' that has the same character as Γ and has an even number of nodes, exactly one of which is omittable.*

Proof. Suppose $\Gamma = (G, X, Y, T)$, where $G = (V, E)$. We can assume that $\mid V \mid$ is odd, since otherwise we can add an omittable node with no incident edges. Suppose $X = \{i_1, \ldots, i_r\}$ and $Y = \{j_1, \ldots, j_s\}$. Then $\Gamma' = (G', X', Y', T')$, where $G' = (V', E')$ and $\mid V' \mid = \mid V \mid +1$ is obtained as follows via a graph G'':

(i) G'' is isomorphic to G except that it has an extra omittable node with index j_1 with no incident edges. The nodes $1, \ldots, j_1 - 1$ in G'' correspond to the same indices in G, while nodes $j_1 + 1, \ldots, \mid V \mid +1$ correspond to $j_1, \ldots, \mid V \mid$ in G, respectively.

(ii) G' is the graph whose matrix is the sum of the matrix of G'' plus $\Lambda^{(n)}$ as defined in the Pfaffian Sum theorem, where $n = \mid V' \mid = \mid V \mid + 1$. Here $\lambda_i = 1$ iff i is omittable in Γ or if $i = j_1$, and otherwise $\lambda_i = 0$.

(iii) $\Gamma' = (G', X', Y', T')$, where $X' = \{i_1, \ldots, i_r\}$, $Y' = \{j_1 + 1, \ldots, j_s + 1\}$, and $T' = \{j_1\}$.

Now for any $Z \subseteq X \cup Y$ there is a $Z' \subseteq X' \cup Y'$ for the corresponding set of nodes in G'. The theorem asserts for all such corresponding pairs Z, Z' that

$$\mu(\Gamma, Z)\mathrm{PfS}(G - Z) = \mu(\Gamma', Z')\mathrm{PfS}(G' - Z').$$

Now $\mu(\Gamma, Z) = \mu(\Gamma', Z')$ clearly. Also it is immediate that $\mathrm{PfS}(G - Z) = \mathrm{PfS}(G'' - Z')$ since the only difference between $G - Z$ and $G'' - Z'$ is that the latter graph contains an additional omittable node with no incident edges. It, therefore, remains to verify that $\mathrm{PfS}(G'' - Z') = \mathrm{PfS}(G' - Z')$. Now $\mid V' - Z' \mid$ is either even or odd, and we claim that in either case the necessary equality is an instance of the Pfaffian Sum theorem. The important observation is that all the omittable nodes in G'' have indices k in the range $i_r < k \leq j_1$. Hence we can think of $G' - Z'$ as being obtained from $G'' - Z'$ by the addition of a Λ matrix for the indices $i_r + 1, i_r + 2, \ldots, j_1$, the entries being zero where any row or column is outside this range. It follows that independent of the choice of Z' we can appeal directly to the Pfaffian Sum theorem to verify the equality that $\mathrm{PfS}(G'' - Z') = \mathrm{PfS}(G' - Z')$: In the case that $\mid Z' \mid$, and hence $\mid G'' - Z' \mid$ also, is even the claim follows by the first case of the Pfaffian Sum theorem, since then $\mathrm{PfS}(G'' - Z') = \mathrm{Pf}(G' - Z') = \mathrm{PfS}(G' - Z')$. Where $\mid Z' \mid$, and hence $\mid G'' - Z' \mid$ also, is odd we note that the contributions to $\mathrm{PfS}(G'' - Z')$ all have node j_1 omitted (since it has no incident edges). Now since the added Λ matrix has nonzero entries for (k, l) only where $k, l \leq j_1$ the effect of adding Λ to $G'' - Z' - \{j_1\}$ is, again by the first case of the Pfaffian Sum theorem, to make $\mathrm{PfS}(G'' - Z')$ equal $\mathrm{Pf}(G' - Z' - \{j_1\}) = \mathrm{PfS}(G' - Z')$. □

The following is proved in [36] and shows that Proposition 4 is essentially optimal. The only gap is that Proposition 4 assumes that $B[4, 4] \neq 0$.

THEOREM 4. *Suppose that for a matchgate Γ with inputs $X = \{i, j\}$ and outputs $\{k, l\}$ with $i < j < k < l$, B is the character matrix for row ordering $\phi, \{i\}, \{j\}, \{i, j\}$ and column ordering $\phi, \{l\}, \{k\}, \{k, l\}$. Then B obeys the five matchgate identities.*

We conclude this section by observing that there is an alternative approach to designing matchgates with desired characters. In this approach one considers matrices that differ from the identity by a multiple of an infinitesimal quantity ϵ. Such matrices have been used to investigate universal quantum gates [14, 15]. In particular, we define the class $\{J_{\alpha\beta} \mid 1 \leq \alpha, \beta \leq 4\}$ of 4×4 matrices with $\{0, 1\}$ entries as follows:

$$J_{\alpha\beta}(i, j) = \delta_{i\alpha}\delta_{j\beta}, \qquad 1 \leq \alpha, \beta, i, j \leq 4,$$

where $\delta_{rs} = 1$ iff $r = s$, and $\delta_{rs} = 0$ otherwise. Clearly each $J_{\alpha\beta}$ has a unit entry at (α, β) and a zero entry elsewhere.

We now define three vector spaces of 4×4 matrices where the a_i are complex numbers:

X1: $a_1 J_{11} + a_2 J_{22} + a_3 J_{33} + (a_2 + a_3 - a_1) J_{44}$.

X2: $J + a_4 J_{23} + a_5 J_{32} + a_6 J_{14} + a_7 J_{41}$, where $J \in$ X1.

X3: $J + a_8(J_{21} + J_{43}) + a_9(J_{12} + J_{34}) + a_{10}(J_{31} - J_{42}) + a_{11}(J_{13} - J_{24})$, where $J \in$ X2.

PROPOSITION 5. *There is a set of* 11 *matchgates whose characters are* $I + \epsilon B_i$ *for* $i = 1, 2, \ldots, 11$, *where* $B_1, \ldots, B_{11}$ *span* X3.

Proof. We enumerate 11 matrices B_i for $i = 1, 2, \ldots, 11$ that span the space X3. In each case we show how to construct a matchgate with character $I + \epsilon B_i$, where ϵ is an infinitesimal indeterminate. In our listing we first describe the matrix B_i such that the character of the matchgate is $I + \epsilon B_i$, and then list all the nonzero weights in the matchgate.

The first six all have $V = \{1, 2, 3, 4\}$, $X = \{1, 2\}$, $Y = \{3, 4\}$, and $T = \emptyset$. The rows of the matrices have ordering $\emptyset$, $\{1\}$, $\{2\}$, $\{1, 2\}$, and the columns have ordering $\emptyset$, $\{4\}$, $\{3\}$, $\{3, 4\}$.

(i) $J_{11} + J_{22}$: $w(1,4) = 1, \quad w(2,3) = 1 + \epsilon$.

(ii) $J_{11} + J_{33}$: $w(1,4) = 1 + \epsilon, \quad w(2,3) = 1$.

(iii) $J_{11} + J_{22} + J_{32}$: $w(1,4) = 1, \quad w(2,3) = 1 + \epsilon, \quad w(1,3) = -\epsilon$.

(iv) $J_{11} + J_{22} + J_{23}$: $w(1,4) = 1, \quad w(2,3) = 1 + \epsilon, \quad w(2,4) = -\epsilon$.

(v) $J_{11} + J_{22} + J_{14}$: $w(1,4) = 1, \quad w(2,3) = 1 + \epsilon, \; w(1,2) = -\epsilon$.

(vi) $J_{11} + J_{22} + J_{41}$: $w(1,4) = 1, \; w(2,3) = 1 + \epsilon, \; w(3,4) = -\epsilon$.

The seventh matchgate has $V = \{1, 2, 3, 4, 5, 6\}$, $X = \{1, 2\}$, $Y = \{5, 6\}$, and $T = \emptyset$. The rows of the matrices have ordering $\emptyset$, $\{1\}$, $\{2\}$, $\{1, 2\}$, and the columns have ordering $\emptyset$, $\{6\}$, $\{5\}$, $\{5, 6\}$.

(vii) $J_{11} + J_{22} + J_{33} + J_{44}$: $w(1,6) = 1, \quad w(2,5) = 1, \quad w(3,4) = \epsilon$.

The last four matchgates are defined as the seventh except now $T = \{4\}$:

(viii) $J_{21} + J_{43}$: $w(1,6) = w(2,5) = w(3,4) = 1, \quad w(3,6) = -\epsilon$.

(ix) $J_{12} + J_{34}$: $w(1,6) = w(2,5) = w(3,4) = 1, \quad w(1,3) = -\epsilon$.

(x) $J_{31} - J_{42}$: $w(1,6) = w(2,5) = w(3,4) = 1, \quad w(3,5) = -\epsilon$.

(xi) $J_{13} - J_{24}$: $w(1,6) = w(2,5) = w(3,4) = 1, \quad w(2,3) = -\epsilon$.

It is easily verified that in each case the matchgate described has character $I + \epsilon J$, where J is the claimed matrix. Further, it is clear that these 11 matrices span the 11-dimensional space X3. □

5. Matchcircuits. We now discuss how matchgates can be combined in large numbers to form circuits. In the first instance the global properties of circuits that we seek are of the same nature as those of individual gates: we define input and output nodes for a circuit and wish to establish the character matrix of the circuit from the character matrices of the constituents and the particulars of how the gates are connected together. These latter particulars need special attention if the characters of the constituents are to be composed so that no untoward sign effects arise. The only difference between a matchgate and a matchcircuit is that modifiers are assumed to be $+1$ for the latter since we do not consider a circuit to be externally connected.

One important purpose of combining matchgates is to derive matchgates with new characters. A most basic operation is that of putting two two-bit gates in sequence, as illustrated in Figure 3. We shall assume that for G_1 and G_2 the ordering in the character matrix is $\phi, \{1\}, \{2\}, \{1, 2\}$ for the inputs and $\phi, \{6\}, \{5\}, \{5, 6\}$ for

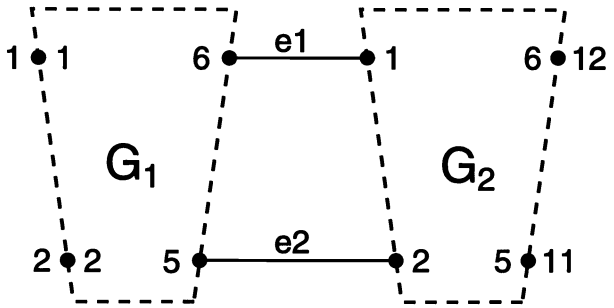


FIG. 3. *A composition of two six node matchgates, G_1 and G_2, that forms a* 12 *node matchgate G. The original labeling $\{1,2,5,6\}$ of the external nodes of G_1, G_2, as well as the new labeling $\{1,2,11,12\}$ of the external nodes of G are shown. In the composition all the* 12 *nodes are renumbered $\{1,\ldots,12\}$ from left to right in the diagram.*

the outputs. For G we assume these orderings to be the same for the inputs and $\phi, \{12\}, \{11\}, \{11,12\}$ for the outputs.

The important point now is that if gate G is formed from constituent gate G_1 appended by gate G_2 as shown, and if the character matrices of G_1 and G_2 are B_1 and B_2, respectively, then the character matrix of G will be $B = B_1B_2$ where matrix multiplication is implied. To see this consider, for example, the entry $B(2,3)$ which corresponds to input $\{1\}$ and output $\{11\}$ of the composite gate G. In the matrix product $B = B_1B_2$ the corresponding value is

$$B_1(2,1)B_2(1,3) + B_1(2,2)B_2(2,3) + B_1(2,3)B_2(3,3) + B_1(2,4)B_2(4,3).$$

It is easy to verify that the four terms correspond, respectively, to sets of matchings in G that (i) contain neither edge $e1$ nor edge $e2$, (ii) have $e1$ but not $e2$, (iii) have $e2$ but not $e1$, and (iv) have both $e1$ and $e2$. In the Pfaffian expression for G these terms all have positive signs since the potential overlap between $e1$ and $e2$ is always even, and their overlap with the internal edges of G_1 and G_2 are already accounted for in the characters of G_1 and G_2 through the modifier.

In this way if, for example, B_1 and B_2 are $I + \epsilon J_1$ and $I + \epsilon J_2$, then their composition will have character $I + (J_1 + J_2)\epsilon + O(\epsilon^2)$. Clearly if we also have a matchgate for $I + \epsilon J_1$, then we can obtain one for $I + a\epsilon J_1$, where a is any complex number by replacing ϵ in the matchgate weights by $a\epsilon$. It follows that if we have matchgates for a set of infinitesimals, such as the 11 in Proposition 5, then we can construct a matchgate for any linear combination of the infinitesimals, up to a $O(\epsilon^2)$ term.

It also follows that if we set $\epsilon = 1/n$, then if we compose n copies of the gate for $I + \epsilon J = e^{\epsilon J} + O(\epsilon^2)$, then we obtain a gate for $e^J + O(1/n)$, as can be seen from the relation

$$(e^{A/n} + O(1/n^2))^n = e^A + O(1/n)$$

which holds for all matrices A that are independent of n. Note that e^A is defined here as the power series $1 + A + A^2/2! + \ldots$. This n-fold composition or *iteration* offers an alternative approach to constructing matchgates with desired characters.

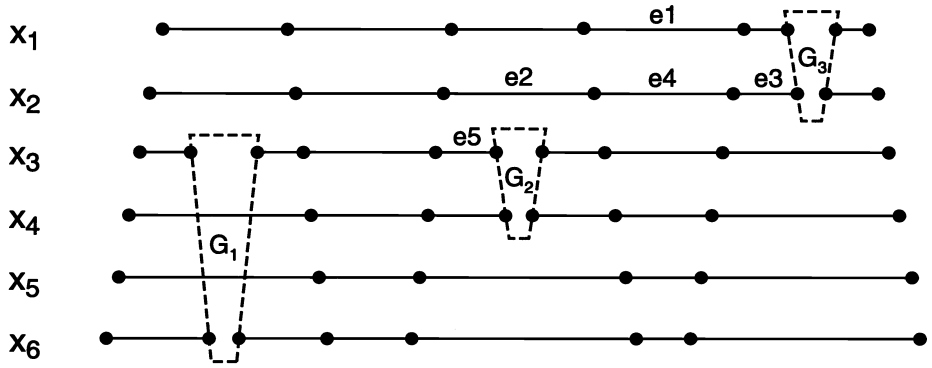


FIG. 4. *An example of a circuit composed of gates G_1, G_2, and G_3 acting on bits $\{x_3, x_6\}$, $\{x_3, x_4\}$ and $\{x_1, x_2\}$, respectively. In this example $e3$ and $e5$ are external edges, $e2$ is a parallel edge, $e1$ and $e4$ are connecting edges, and the internal edges within G_1, G_2, and G_3 are not shown. The nodes in the overall circuit are numbered so as to be increasing from left to right. The input and output node sets X, Y are the leftmost and rightmost six nodes, respectively.*

More generally let $\Gamma_1, \ldots, \Gamma_m$ be a set of matchgates $\Gamma_i = (G_i, X_i, Y_i, T_i)$, where $G_i = (V_i, E_i, W_i)$ and $|X_i| = |Y_i|$. A *matchcircuit* $\Gamma = (G, X, Y, T)$ with $G = (V, E, W)$ is a composition of $\Gamma_1, \ldots, \Gamma_m$ that can be obtained in the following way: (1) The Γ_i are first reordered as necessary. (2) Each external node of Γ_i has one external edge. An output node of Γ_i can be linked to an input node of Γ_j if $i < j$, via a *linking chain* of an odd number of edges all of weight 1, and in that case that output node and that input node are considered as members of a set D. (3) Nodes in X_i or Y_i that are not in D are the endpoints of linking chains of odd length, the other endpoints of which are considered as the members of X or Y, the input and output nodes of the circuit, respectively. An odd length chain between a node in X and a node in Y represents a bit on which the circuit does not act. (4) T is the union of the T_i. (5) The node set is now renumbered as necessary so that for each i all the nodes in V_i are *contiguous* in V (i.e., if j, k are nodes originating from V_i, and $j < l < k$ for some node l in V, then l originates from a node in V_i and is not a node on a linking chain or from a different V_i), the nodes X have smaller indices than any other node, and Y have larger indices.

Figure 4 shows an example of a matchcircuit with $|X| = |Y| = 6$ and with three matchgates. When we construct a matchcircuit from matchgates we have in mind the construction shown. Note that if $n = |X| = |Y|$, then for each gate $2n$ new nodes are introduced with a relative numbering implied by the left to right ordering shown.

MATCHCIRCUIT THEOREM. *If matchcircuit Γ is a composition of matchgates $\Gamma_1, \ldots, \Gamma_m$, then*

$$\mathrm{PfS}(G) = \sum_S \epsilon_S \prod_{1 \le i \le m} \mathrm{PfS}(G_i - S_i),$$

where (1) *S is a mapping that determines for each node in D whether it is to be matched by an edge internal or external to the matchgate from which it originated,* (2) *S_i is the set of external nodes in Γ_i that are assigned by S to be matched by an external edge, and* (3) *$\epsilon_S \in \{-1, 1\}$ (and depends on the choice of S and on the nature*

of the composition Γ).

Proof. The different choices of the assignment S partition the matchings of G. In order to establish the theorem it suffices to show that the matchings of G that respect S contribute to $\mathrm{PfS}(G)$ a term

$$\prod_{1\le i\le m} \mathrm{PfS}(G_i - S_i) \tag{3}$$

up to a multiplicative factor of 1 or -1.

It is clear that for any sets $U_1 \subseteq T_1$, $U_2 \subseteq T_2, \ldots, U_m \subseteq T_m$, and $U = U_1 \cup U_2 \cup \cdots \cup U_m$ each matching in $\mathrm{PfS}(G)$ that omits U corresponds one to one to the union of matchings from $\Gamma_1, \ldots, \Gamma_m$ that omit $U_1, \ldots, U_m$, respectively. Hence the monomials in $\mathrm{PfS}(G)$ that are consistent with S are identical to the monomials in (3) above. It remains therefore only to establish that either all the corresponding pairs of monomials have equal signs (and hence $\epsilon_S = 1$) or they all have unequal signs (and hence $\epsilon_S = -1$).

For all the matchings in G that are consistent with a single S the external edges (and other chain edges) matched are identical. Hence the matchings for the fixed S differ only as far as the edges that are entirely internal to the various Γ_i separately. Within any one such Γ_i the signs of the various terms in $\mathrm{PfS}(G_i - S_i)$ are whatever they are. When they form subterms of $\mathrm{PfS}(G)$ their signs are not affected relative to each other. This is because for any two edges from distinct gates, say edge (u, v) of E_i and (w, x) of E_j, either $u < v < w < x$ or $w < x < u < v$ and therefore there is no overlap in either case. It follows that the contributions from the various $\mathrm{PfS}(G_i - S_i)$ simply multiply together.

It remains to observe that the external and other linking edges generate no overlap. This is because the overlap between the external edges and those within gates is, as previously observed, fixed by virtue of condition (ii) in the definition of matchgates. The overlap among the external edges and linking chain edges is clearly fixed also since there is just one matching being considered for these. It remains to observe that the fixed matching in the linking edges that are not external (i.e., not directly incident to a gate) have no overlap with internal gate edges by virtue of condition (5) in the definition of matchcircuits that imposes an adequate constraint on their numbering. □

We shall think of a matchcircuit as acting on a sequence of bits $x_1, \ldots, x_n$ and of a k-bit matchgate as acting on a subset of these n bits. We say that a matchcircuit is in *standard description* if when the character matrix of a gate is described to be B it is the case that any one diagonal element of B refers to exactly the same subset of $\{x_1, \ldots, x_n\}$ for the inputs as for the outputs. Thus if, for example, $B(2, 2)$ refers to an input subset of $\{1\}$ and an output subset of $\{6\}$, as in Proposition 2, then a standard description would insist that output 6 of this gate be the same x_i bit as input 1 of this gate. Clearly, a circuit not in standard description can be put into standard description at the expense only of reordering the outputs (and renaming the bits operated on inside the circuit.)

The following Main theorem gives some general conditions that are sufficient to guarantee that the above Matchcircuit theorem can be applied with $\epsilon_S = 1$ in all cases. This in turn ensures that when the matchgates are composed into a matchcircuit, the character matrix of the circuit is the matrix product of the character matrices of the individual gates when these are regarded as acting on all n bits. This follows from the argument used earlier in the section to show that the composition of the two

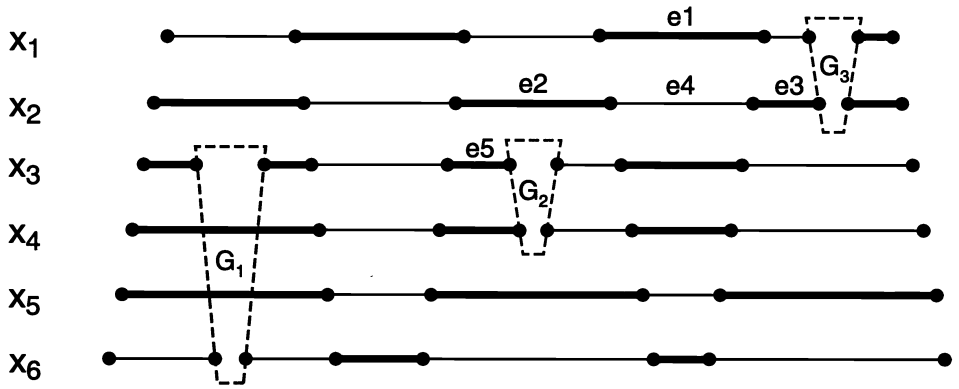


FIG. 5. *The matching for the matchcircuit shown in Figure* 4 *when* G_1, G_2, *and* G_3 *are realizing mappings* J_{22}, J_{41}, *and* J_{34}, *respectively. The matched edges outside of* G_1, G_2, *and* G_3 *are shown in bold. In this instance the circuit realizes the mapping* $(0,1,1,1,1,0) \to (1,1,0,0,1,0)$, *where an entry is* 1 *or* 0 *according to whether the node in* X *or* Y *corresponding to it is saturated by an edge in the matching.*

matchgates shown in Figure 3 corresponds to the multiplication of their character matrices.

MAIN THEOREM. *For any matchcircuit composed of two-bit matchgates any entry in its character matrix can be expressed as a Pfaffian Sum of the circuit with the corresponding external nodes removed if the circuit when described in standard form is composed of* (a) *matchgates with any character* B *such that* $B(i,j) = 0$ *for all* i,j *such that* $i \neq j$, (b) *matchgates operating on any two bits with indices differing by one (e.g.,* x_i, x_{i+1}*) that have character* B *in which the only nonzero entries are among* $B(1,1)$, $B(2,2)$, $B(3,3)$, $B(4,4)$, $B(1,4)$, $B(4,1)$, $B(2,3)$, $B(3,2)$, *and* (c) *any matchgate operating on bits* x_1 *and* x_2.

Proof. Figure 4 shows a circuit with three gates G_1, G_2, and G_3. They illustrate the cases (a), (b), and (c), respectively, since G_2 acts on adjacent gates x_3, x_4, and G_3 acts on the end bits x_1, x_2. Also shown is a systematic way of adding chains of edges to join gate outputs to gate inputs so that in the composition the Pfaffians multiply and add with positive sign.

Figure 5 illustrates the following specific nondeterministic branch of a computation of the above circuit: The input is $(x_1, x_2, x_3, x_4, x_5, x_6) = (0,1,1,1,1,0)$. Before the composition all of the gates G_1, G_2, and G_3 have external node sets $\{1,2,7,8\}$, row ordering $\emptyset$, $\{1\}$, $\{2\}$, $\{1,2\}$, and column ordering $\emptyset$, $\{8\}$, $\{7\}$, $\{7,8\}$. In the illustration gate G_1 is realizing the identity mapping from input $x_3 = 1, x_6 = 0$ to output $x_3 = 1, x_6 = 0$. It is placed in the circuit so that input 1 and output 8 of the gate act on bit x_3, and input 2 and output 7 act on bit x_6. It can be thought of, therefore, as internally realizing the character entry $(2,2)$ or the mapping J_{22}. Similarly gate G_2 realizes mapping J_{41} and is placed in the circuit so that input 1 and output 8 of the gate act on bit x_3, and input 2 and output 7 act on bit x_4. Gate G_3 realizes mapping J_{34} and is placed in the circuit so that input 1 and output 8 of the gate act on bit x_1, and input 2 and output 7 act on bit x_2. By saying here that the various gates realize various mappings we are in fact specifying a particular choice of S in the statement of the Matchcircuit theorem. Equivalently we are specifying particular

subsets of matchings in all the gates.

What we need to show is that for any such choice of S that corresponds to nonzero character entries for all the gates, whether of types (a), (b), or (c) in the statement of the Main theorem, the value of ϵ_S in the Matchcircuit theorem is always $+1$. To see this we classify the edges in the circuit into categories: ones that are *internal* to a gate, ones that are *external* edges (those that connect the external nodes of a gate to the outside, e.g., $e3$, $e5$ in Figure 4), those that are *parallel* to gates (e.g., $e2$), which are defined to be those above or below the gates in Figure 4, and *connecting* edges (e.g., $e1$, $e4$), which are sets of n edges that connect the external/parallel edges of one gate to the external/parallel edges of another. Clearly there are three sources of overlapping edges that might affect ϵ_S: overlap between internal and external edges, overlap between external/parallel edges and connecting edges, and overlap among the external/parallel edges involved in the same gate. The first source is already taken into account in the definition of the character of a matchgate. The second contributes $+1$ since any particular connecting edge (e.g., $e4$) forces the external/parallel edges at its two ends ($e2$, $e3$) to be either both matched or neither matched, and hence the overlaps of these latter edges with any connecting edge (e.g., $e1$) above the connecting edge ($e4$) in question will occur in pairs and hence cancel.

The remaining case is that of overlaps among the external/parallel edges of the same gate, and these can be only between an edge externally connected to a gate (e.g., $e5$) and a parallel edge above it (e.g., $e2$). In case (c) since the gate acts on x_1, x_2, there are no parallel edges above the gate, and hence there is no contribution to ϵ_S. In case (b) since the gate has an even number of external edges that are in the matching and their overlaps are with parallel edges above, these contributions will be in even pairs and will also cancel. Finally, in case (a) matched external edges will occur in pairs with the same x_i (e.g., for gate G_1 in Figure 5; such a pair occurs for x_3 and not for x_6). Overlaps with the parallel edges above will therefore again occur in pairs and cancel.

With regard to the details of the construction for composing matchgates that Figure 3 illustrates, the reader can verify that there is much redundancy. All the nodes other than matchgate nodes or the circuit input and output nodes can be eliminated. □

We note that if we take into account the matchgate identities, then case (a) can be simulated by case (b) and is technically redundant in the above theorem. To see this note that a case (a) matrix is a diagonal matrix with some $(a, b, c, bc/a)$ on the diagonal. This can be simulated by two diagonal one-bit gates, the first with diagonal (a, b) acting on the first bit, and the second with diagonal $(1, c/a)$ acting on the second bit. Each of these one-bit gates can then be viewed as a two-bit gate that ignores the other bit and is feasible for a matchgate. Hence a case (a) two-qubit gate acting on bits i, j can be replaced by two one-bit gates acting on bits $i, i+1$ and $j, j+1$, respectively.

We call Ω the class of matchcircuits defined in the above theorem and limited to the classes of matchgates that can be constructed. In particular the gates that can be used in unrestricted ways include those of Proposition 2. Those that can be applied to neighboring bits include those of Proposition 3. Those that can be applied to the end bits x_1, x_2 include those of Proposition 4.

Now consider a member M of this class Ω. For any input variable x_i or output variable y_j that is to be fixed to have value 0 we delete this input/output node and the edges incident to it. This forces the circuit to have value 0 for this variable since the

corresponding $X \cup Y$ node will not be saturated. If we wish to set the variable to 1 we retain the corresponding nodes and edges. (This $\{0,1\}$ interpretation of the variable values is the complement of that considered for the character, but this is an inessential technicality.) For $\underline{x} \in \{0,1\}^n$, $\underline{y} \in \{0,1\}^n$ we define $M_{\underline{x},\underline{y}}$ to be the matchcircuit M with these modifications. In the case that the matchgates are unitary, we can regard the circuit as a quantum matchcircuit.

COROLLARY 1. *For a quantum matchcircuit $M \in \Omega$ started in state $\underline{x}$ the probability that it terminates in state $\underline{y}$ is $|\mathrm{PfS}(M_{\underline{x},\underline{y}})|^2$.* □

A critical aspect of our theory, which we have not mentioned and which goes beyond the standard quantum computational model, is that it is capable of simulating in polynomial time *nondeterministic summation* over the *inputs* of the simulated circuit evaluations in addition to the nondeterministic branches. Thus if $\underline{x}^*, \underline{y}^*$ assign subsets of the inputs and outputs fixed values, we first excise the external nodes that are so fixed to have value 0 and retain those fixed as 1. We then apply to the resulting matchcircuit M the various transformations described in the following.

THEOREM 5. *For any matchcircuit $M \in \Omega$, if x^*, y^* are partial assignments to the inputs and outputs, then each of the three quantities*

$$\left(\sum_{\underline{x},\underline{y}} \mathrm{PfS}(M_{\underline{x},\underline{y}})\right)^2, \qquad \sum_{\underline{x},\underline{y}} |\ \mathrm{PfS}(M_{\underline{x},\underline{y}})\ |^2, \quad \textit{and} \qquad \sum_{\underline{x},\underline{y}} (\mathrm{PfS}(M_{\underline{x},\underline{y}}))^2,$$

where summation is over all $\underline{x}$ and $\underline{y}$ that agree with $\underline{x}^*$ and $\underline{y}^*$, respectively, can be evaluated in time polynomial in the size of M.

Proof. For the first quantity we define the nondeterministic input and output nodes in M as omittable nodes in the matchcircuit and appeal to the Pfaffian Sum theorem.

For the second quantity we construct for M its complex conjugate matchcircuit M^* which is identical to M except that corresponding weights are complex conjugates of each other, and the node indices are from a disjoint interval of, say, higher numbers, and in reverse order. For each nondeterministic input or output node we join the corresponding nodes of M and M^* by an edge of weight one. We claim that the Pfaffian of the result is the desired quantity. This follows from the observation that each subset of the joining edges that is matched represents a distinct truth assignment to the nondeterministic nodes, and its contribution to the Pfaffian will be the product of some $\mathrm{PfS}(M_{\underline{x},\underline{y}})$ and its complex conjugate. The only technical issue that needs to be verified is that the contribution of the newly added edges to the parity of the sign is $+1$ for all choices of $\underline{x}$ and $\underline{y}$. To see this we note that the edges between the outputs clearly interact with each other with even parity and with the copies of M and M^* totally symmetrically. Similarly the edges between the inputs, which now span the whole circuit, also interact with each other and with the edges between the outputs with even parity. Also, they interact with the edges that touch the input nodes of M and M^* symmetrically and with all other edges of M, M^* with even parity. Figure 6 illustrates this construction.

The third quantity is computed in the same way as the second except that instead of M^* we simply use a second copy of M. □

This result, clearly, has applications broader than the quantum computation model, since it is not restricted to unitary matrices. We note, however, that if we do restrict ourselves to the quantum model, or any other model with a similar probabilistic interpretation, then our computational mechanisms may be compared with

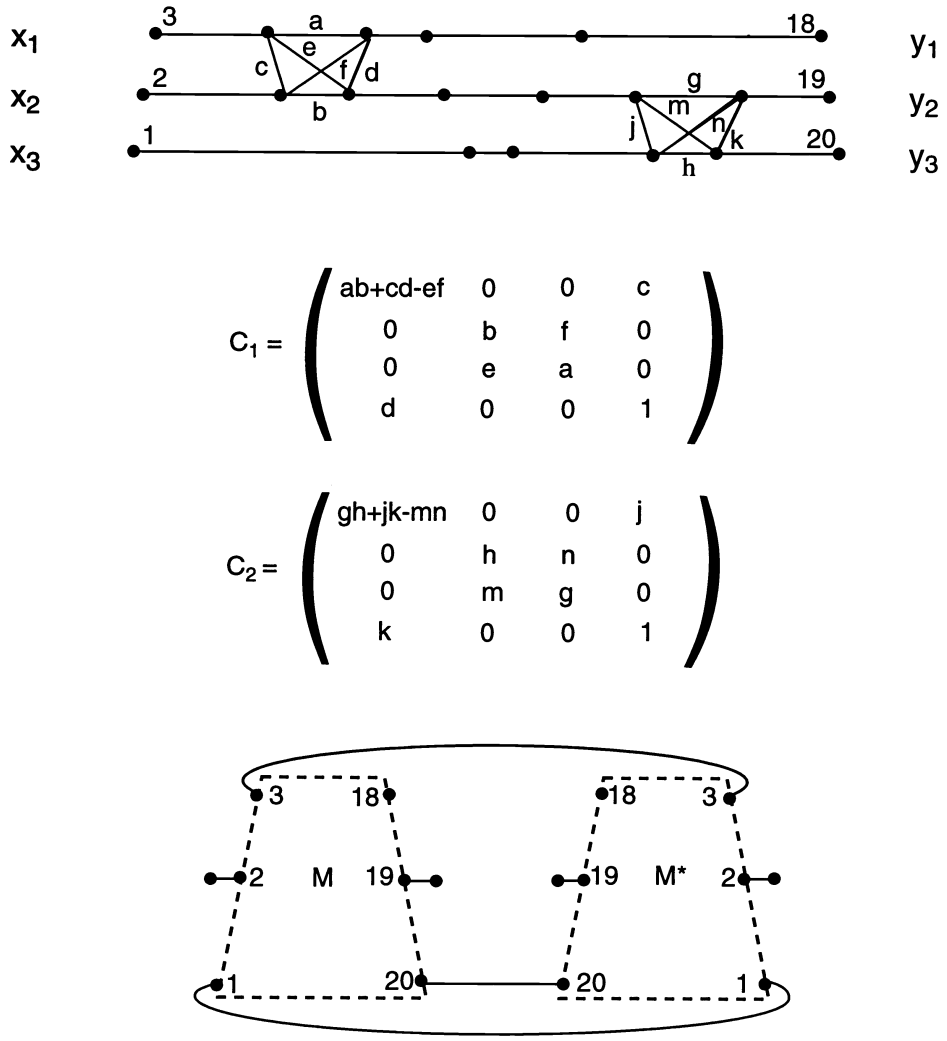


FIG. 6. *At the top we illustrate a* 20*-node matchcircuit* M *on* 3 *qubits that realizes the composition of a transformation* C_1 *on* x_1, x_2*, followed by* C_2 *on* x_2, x_3*. The* 0-1 *interpretation of the character matrix of* M *is the complement of that of the gates but could be made the same if each input and output of* M *were extended by a further edge. At the bottom the diagram illustrates how a copy of* M *is composed with a copy of its complex conjugate with reverse numbering in order to realize the second part of Theorem* 5*. In this example the Pfaffian(Sum) of the adjacency matrix will equal* $\Sigma|\mathrm{PfS}(M_{\underline{x},\underline{y}})|^2$ *with* $\underline{x}$ *summed over all values such that* $x_2 = 0$*, and* y *summed over all values with* $y_1 = 1$ *and* $y_2 = 0$*, and hence in the quantum interpretation will equal the probability that such evolutions occur.*

aspects of quantum physics. In particular, the second part of the above theorem can be seen as a deterministic process that computes the exact probability associated with a particular set of evolutions between quantum states. The result to follow builds on this by showing that in randomized polynomial time one can generate an output state according to the probability distribution that the quantum mechanical

interpretation of a quantum matchcircuit specifies. This is precisely equivalent to the idea of performing a measurement in quantum mechanics.

MEASUREMENT THEOREM. *For any quantum matchcircuit $M \in \Omega$ if $\underline{x}$ is a total assignment to the inputs, and $\underline{y}^*$ a partial assignment to the outputs, then there is a randomized procedure that runs in time polynomial in the size of M and generates an output $\underline{y}$ consistent with $\underline{y}^*$ with probability*

$$|\mathrm{PfS}(M_{\underline{x},\underline{y}})|^2 / \sum |\mathrm{PfS}(M_{\underline{x},\underline{z}})|^2,$$

where summation in the denominator is over all $\underline{z}$ consistent with $\underline{y}^*$.

Proof. We proceed in the manner described in [23] for generating a random combinatorial structure when we can count their number. The procedure has m stages if there are m nondeterministic variables. In each stage we consider one such variable y_i and fix its value as follows: We evaluate for $y_i = 0$, and separately for $y_i = 1$, the quantity

$$\sum |\mathrm{PfS}(M_{\underline{x},\underline{y}})|^2,$$

where the summation is over the $\underline{y}$ variables that have not been fixed yet. Each of the two evaluations is a call of the procedure in the second part of Theorem 5. If we obtain the values p_0 and p_1 for these two calls, then with probability $p_0/(p_0 + p_1)$ we fix $y_i = 0$ and with probability $p_1/(p_0 + p_1)$ we fix $y_i = 1$. It is clear that the procedure has the desired result. □

In conclusion we note that we can generalize the Measurement theorem to sum nondeterministically over unspecified inputs also. There is an obvious restriction, however, on how the various nondeterministic choices can be correlated. If they are all independent, then nothing special needs to be done. Otherwise one option is to relate pairs of them by joining such a pair by an edge or a chain of two edges, and this corresponds to the read-twice circuits discussed below. Controlling the overlaps of the added edges becomes problematic in general, however. The next section takes this route under circumstances when these overlaps do not matter.

6. Matchnets. We shall compose matchgates now in a different way. For quantum computation we needed to be careful in constructing matchcircuits so that the effects of the constituent matchgates simply add, without any uncontrollable sign effects. Thus our simulation of quantum computation was limited by (a) what individual matchgates could achieve and by (b) the constrained composition rules that were adopted to maintain control of the signs. In the two applications in this section we shall be limited by (a) alone. In the first application we shall be detecting the *existence* of solutions and in the second the *parity* of the number of solutions.

For our current purposes we shall consider matchgates with empty output node sets Y. For simulating i-argument Boolean functions we shall use matchgates with i input variables X. In this section the sign of the character of each gate will be immaterial, and hence consideration of the modifier in the definition of a matchgate is redundant.

In this section we define algorithms for *read-twice* Boolean formulae. These are formulae that are conjunctions of expressions such that each variable influences at most two of the expressions. To describe the individual expressions we consider the following classification of Boolean functions:

0-argument functions: $B01 = \{1, 0\}$
1-argument functions: $B11 = \{x\}$
2-argument functions: $B21 = \{xy, x + y, x \oplus y = 1, x \oplus y = 0\}$
3-argument functions: $B31 \cup B32 \cup B33$, where
$B31 = \{xyz, x(y = z), x'yz + xy'z + xyz', x(y + z),$
$x \oplus y \oplus z = 1, xy + yz + zx,$
$x + (y = z), xy + (y = z)\},$
$B32 = \{x + y + z, \neg(xyz + x'y'z')\}$, and
$B33 = \{xyz + x'y'z', xyz + x'y', xy + y'z,$
$x + yz, z = x + y, x'y'z' + xy + yz + zx\}.$

Altogether there are 23 Boolean functions listed here. It is easily verified that any Boolean function of three or fewer variables is identical to one of these 23 functions once x, y, and z have been replaced by appropriate instances of the three variables or of their negation. For example, the OR gate $z = x + y$ in B33 can be interpreted as $z' = x' + y'$ which is equivalent to the AND gate $z = xy$, which is therefore not listed separately. However, $xyz + x'y'z'$ and $\neg(xyz + x'y'z')$, for example, are not equivalent in this sense. We note that only the last 16 functions, those in $B31 \cup B32 \cup B33$, depend on all three of their arguments, and that the first seven, namely those in $B01 \cup B11 \cup B21$, can all be obtained by appropriately fixing some inputs of some member of $B31$. Hence our claims about algorithms for $B31$ apply equally to $B01 \cup B11 \cup B21 \cup B31$.

Let us therefore first consider the eight 3-argument functions in the set $B31$. In each case we shall describe a matchgate whose character, as described by a vector of length 8, specifies the Boolean function at the eight possible input values. A further special property of these gates is that the character entries are always 0 or 1 and (a) whenever it is 0 the gate has no matchings, and (b) whenever it is 1 then it has exactly one matching. In other words in the corresponding Pfaffian there is either zero or one nonvanishing monomial. Also in all nine cases $V = \{1, 2, 3, 4\}$, $X = \{1, 2, 3\}$, $Y = \emptyset$, and, where not otherwise stated, $T = \{4\}$. It remains only to specify the nonzero edge weight set in each case. We shall specify the actual function realized using x, y, z to represent inputs 1, 2, 3, respectively. We use the same numbering convention as before. Thus $Z = \{1, 3\} \subseteq X$ will give the character matrix element corresponding to $x = 1, y = 0, z = 1$. Also, we denote the negation of a variable x by x'.

xyz: All weights zero.
$(x = y)z$: $w(1, 2) = 1$.
$(x + y)z$: $w(1, 4) = 1$, $w(2, 4) = 1$.
$x + (y = z)$: $w(1, 4) = 1$, $w(2, 4) = 1$, $w(3, 4) = 1$, $w(2, 3) = 1$.
$xy + (y = z)$: $w(1, 4) = 1$, $w(3, 4) = 1$, $w(2, 3) = 1$.
$xy + yz + zx$: $w(1, 4) = 1$, $w(2, 4) = 1$, $w(3, 4) = 1$.
$x \oplus y \oplus z = 1$: $w(1, 2) = 1$, $w(2, 3) = 1$, $w(3, 1) = 1$.
$xyz' + xy'z + x'yz$: $w(1, 4) = 1$, $w(2, 4) = 1$, $w(3, 4) = 1$, $T = \emptyset$.

Let us next consider the two 3-argument functions in the set $B32$. The setup is exactly as for the $B31$ gates above with the one exception that whenever the Boolean function takes the value 1 there may now be more than one matching in the matchgate and this number is even (though the corresponding Pfaffian is in fact nonzero—a property which is not exploited.) The two functions here are disjunction and not-all-equals. We note that it is enough to implement functions to within negations of the inputs: One can unnegate an input by adding a single edge to the old input and

declaring the new endpoint the new input. This may change the modifier, but that is not relevant for this section.) In order to retain four node constructions we shall describe gates where x and z are negated in the disjunction and y in the not-all-equals:

$x' + y + z'$: $w(1,2) = 1$, $w(1,3) = 1$, $w(1,4) = 1$, $w(2,3) = 1$, $w(3,4) = 1$.
$\neg(xy'z + x'yz')$: $w(1,2) = 1$, $w(1,4) = 1$, $w(2,3) = 1$, $w(3,4) = 1$.

A matchnet is a set of matchgates with no output nodes where pairs of input nodes from different matchgates may be joined (a) by a single edge of weight 1 if the input nodes are to have the same value or (b) by a chain of two edges both of weight 1 if the input nodes are to have opposite values. We can then define the matchnet of a read-twice Boolean formula as follows: if the formula is a conjunction of a number of 3-argument Boolean expressions, then the matchnet is simply the union of matchgates, one for each such expression, in addition to single edges or chains of length two to constrain pairs of variables in the expressions to have equal or unequal values, as required.

More formally a *matchnet* is a matchcircuit having no input or output nodes, and composed of matchgates with no outputs, where condition (1) in the definition of matchcircuits is redundant, and conditions (2) and (3) are replaced by the above condition, that inputs of different matchgates may be linked by edge chains of length one or two.

The following theorem results using standard algorithms for finding matchings and for computing determinants.

THEOREM 6. *If F is a read-twice formula consisting of gates from $B01 \cup B11 \cup B21 \cup B31 \cup B32$, then the matchnet of F has a matching iff F is satisfiable. If F is a read-twice formula F consisting of gates from $B01 \cup B11 \cup B21 \cup B31$, then the matchnet of F has an odd number of matchings iff F has an odd number of satisfying assignments. The existence of a solution for the formula in the first case and the parity of the number of solutions in the second case can both be computed in deterministic time polynomial in the size of F.* □

A variety of special cases of these results were previously known and some related problems are known to be NP-hard. One strength of the result above is that a broad variety of gates can be treated together. Some trivial special cases include restricting the gates to two argument functions or to parity functions. A more interesting case that was previously known was read-twice formulae consisting only of $x + y + z$ gates (from $B32$), for which it was known that satisfiability is polynomial time solvable [28, p. 207]. For that case it was further known that counting the number of solutions is #P-complete [6], although the parity problem for it is apparently open. The construction in [22] for read-twice formulae (of the form of conjunctions of disjunctions of conjunctions) implies that the existence problem for read-twice formulae constructed from $z = x + y$ functions is NP-complete. Also one can deduce that the corresponding counting problem is #P-complete and, using [37], that the corresponding parity problem is NP-hard via randomized reduction.

THEOREM 7. *None of the six functions in $B33$ has a matchgate where either the parity of the number of matchings or the existence of matchings defines the required function. Neither of the functions in $B32$ has a matchgate where the parity of the number of matchings defines the required function.*

Proof. Assume to the contrary that such matchgates exist and, without loss of generality, that they have an odd number n of nodes. If n is even one can add

an omittable node having no incident edges. Assume that the input node set is $X = \{1, 2, 3\}$ and that $Y = \emptyset$.

Construct the matrix $P = B^+ + \Lambda^{(n+1)}$ exactly as in the second part of the Pfaffian Sum theorem. Let $i = 1, j = 2, k = 3, l = n + 1$ in Theorem 2 and use the abbreviation $P(a, b, c, d)$ to denote the Pfaffian of P with rows/columns $1, 2, 3, n + 1$ removed according to whether a, b, c, d equal 1. Then Theorem 2 implies

$$\begin{aligned} P(0,0,0,0)P(1,1,1,1) - P(1,1,0,0)P(0,0,1,1) \\ = P(1,0,0,1)P(0,1,1,0) - P(1,0,1,0)P(0,1,0,1). \end{aligned}$$

Since in each term $d = a \oplus b \oplus c$, the last argument is indeed redundant, and we can abbreviate $P(a, b, c, d)$ with $P(a, b, c)$ to get

$$\begin{aligned} P(0,0,0)P(1,1,1) - P(1,1,0)P(0,0,1) \\ = P(1,0,0)P(0,1,1) - P(1,0,1)P(0,1,0). \end{aligned} \tag{4}$$

The main observation now is that $P(a, b, c)$ is nothing other than the component of the character (up to a factor of ± 1) of the gate where the inputs set to 1 are removed from the gate. This is because renaming the nodes and then applying the Pfaffian Sum theorem would give exactly the same matrix as P with the corresponding rows/columns removed.

It follows, for example, that the all-equal function $xyz + x'y'z'$ cannot be realized since it would require the first of the four terms in (4) to be nonzero and the rest zero. This holds clearly if the Pfaffian equals the parity. Equally it precludes the possibility that matchings exist for just the desired subsets of inputs because in that case we could place random values on the edges of the matchgate and the Pfaffians would have forbidden zero and nonzero patterns for some such random values. Note that when considering existence rather than parity, instead of appealing to the Pfaffian Sum theorem to eliminate omittable nodes, we simply connect these nodes as a complete subgraph.

This same argument can be applied also to the remaining five functions in $B33$: $x'y'z' + xyz$, $z = x + y$, $x + yz$, $xy + y'z$, and $xyz + x'y'$. A similar argument holds for the $B32$ functions for the case of parity. □

We observe finally that numerous open problems remain regarding read-twice formulae composed of gates from the 23 gates that we enumerated. In particular the complexity of determining the existence, the parity, and the number of the solutions each remains unresolved for several cases.

7. Conclusion. The exact power of matchgate techniques for deriving polynomial time classical algorithms remains to be resolved. Are there indirect methods not yet identified for mapping richer classes computations into matchgates? Can our particular class of polynomially simulatable circuits be extended by allowing in addition arbitrary use of one-bit gates? Do circuits based on k-bit matchgates for $k > 2$ have greater power for encoding computations?

We have defined a class of quantum computations for which the outcome can be predicted by linear algebra computations in polynomial time by classical computers. While the standard quantum mechanical formulation is linear also, it is linear in exponentially more dimensions than is ours. This brings new focus to the question of whether scalable quantum devices are restricted to those that are polynomial time simulatable classically, as nonquantum devices are believed to be.

REFERENCES

[1] A. Barenco, *A universal two-bit gate for quantum computation*, Proc. Roy. Soc. London Ser. A, 449 (1995), pp. 679–683.

[2] P. A. Benioff, *Quantum mechanical Hamiltonian models of discrete processes that erase their own histories: Application to Turing machines*, Internat. J. Theoret. Phys., 21 (1982), pp. 177–201.

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

[4] A. Borodin, S. A. Cook, and N. J. Pippenger, *Parallel computation for well-endowed rings and space bounded probabilistic machines*, Inform. and Control, 58 (1983), pp. 113–136.

[5] R. A. Brualdi and H. J. Ryser, *Combinatorial Matrix Theory*, Cambridge University Press, Cambridge, UK, 1991.

[6] R. Bubley and M. Dyer, *Graph orientations with no sink and an approximation for a hard case of #SAT*, in Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, 1997, pp. 248–257.

[7] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Springer-Verlag, Berlin, 1996.

[8] P. Bürgisser, *Completeness and Reduction in Algebraic Complexity Theory*, Springer-Verlag, Berlin, 2000.

[9] A. Cayley, *Sur les determinants gauches*, Crelle's J., 38 (1847), pp. 93–96.

[10] S. A. Cook, *The complexity of theorem proving procedures*, in Proceedings of the 3rd ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[11] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[12] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. Roy. Soc. London Ser. A, 400 (1985), pp. 97–117.

[13] D. Deutsch, *Quantum computational networks*, Proc. Roy. Soc. London Ser. A, 425 (1989), pp. 73–90.

[14] D. Deutsch, A. Barenco, and A. Ekert, *Universality of quantum computation*, Proc. Roy. Soc. London Ser. A, 449 (1995), pp. 669–677.

[15] D. P. DiVincenzo, *Two-bit gates are universal for quantum computation*, Phys. Rev. A (3), 51 (1995), pp. 1015–1022.

[16] P. van Emde Boas, *Machine models and simulations*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 1–61.

[17] R. P. Feynman, *Simulating physics with computers*, Internat. J. Theoret. Phys., 21 (1982), pp. 467–488.

[18] L. Fortnow and J. Rogers, *Complexity limitations on quantum computation*, J. Comput. System Sci., 59 (1999), pp. 240–252.

[19] M. Freedman, *N/NP, and the quantum field computer*, Proc. Natl. Acad. Sci. USA, 95 (1998), pp. 98–101.

[20] D. Gottesman, *The Heisenberg Representation of Quantum Computers*, quant-ph/9807006, 1998.

[21] L. K. Grover, *Fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposim on Theory of Computing, ACM, New York, 1996, pp. 212–218.

[22] H. B. Hunt III and R. E. Stearns, *The complexity of very simple Boolean formulas with applications*, SIAM J. Comput., 19 (1990), pp. 44–70.

[23] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 46 (1986), pp. 169–188.

[24] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro, *Computability by probabilistic machines*, in Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, 1956, pp. 183–212.

[25] E. H. Lieb, *A theorem on Pfaffians*, J. Combinatorial Theory, 5 (1968), pp. 313–319.

[26] K. Murota, *Matrices and Matroids for Systems Analysis*, Springer-Verlag, Berlin, 2000.

[27] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.

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

[29] R.W. Robinett, *Quantum Mechanics*, Oxford University Press, New York, 1997.

[30] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

[31] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1968), pp. 354–356.

[32] L. Troyansky and N. Tishby, *Permanent uncertainty: On the quantum evaluation of the*

determinant and permanent of a matrix, Proc. Phys. Comp., 96 (1996).
[33] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. Lond. Math. Soc. Ser. 2, 42 (1936), pp. 230–265.
[34] L. G. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[35] L. G. Valiant, *Completeness classes in algebra*, in Proceedings of the 11th ACM Symposium on Theory of Computing, ACM, New York, 1979, pp. 249–261.
[36] L. G. Valiant, *Expressiveness of matchgates*, Theoret. Comput. Sci., to appear.
[37] L. G. Valiant and V. V. Vazirani, *NP is as easy as detecting single solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.
[38] A. C.-C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1993, pp. 352–360.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1255–1285

FINDING DOUBLE EULER TRAILS OF PLANAR GRAPHS IN LINEAR TIME*

ZHI-ZHONG CHEN†, XIN HE‡, AND CHUN-HSI HUANG§

Abstract. This paper answers an open question in the design of complimentary metal-oxide semiconductor VLSI circuits. The question asks whether a *polynomial-time* algorithm can decide if a given planar graph has a plane embedding $\mathcal{E}$ such that $\mathcal{E}$ has an Euler trail $P = e_1 e_2 \dots e_m$ and its dual graph has an Euler trail $P^* = e_1^* e_2^* \dots e_m^*$, where e_i^* is the dual edge of e_i for $i = 1, 2, \dots, m$. This paper answers this question in the affirmative by presenting a *linear-time* algorithm.

Key words. planar graph, dual graph, Euler trail

AMS subject classifications. 05C45, 05C85, 05C90, 68Q35, 68R10

PII. S0097539799354321

1. Introduction.

Throughout this paper, a graph may have multiple edges but never has loops. A graph G is *planar* if it can be drawn on the plane so that the edges intersect only at their end vertices. Such a drawing is called an *embedding* of G. A *plane graph* is a planar graph with a fixed embedding. A *two-terminal graph* [1] (TTG) $G = (V, E)$ is a plane graph with a pair (s, t) of specified vertices on the outer face such that adding the edge $\{s, t\}$ to G yields a biconnected plane graph [4, 6]. We call s the *source* of G and t the *sink* of G. We also call s and t the *poles* of G and the other vertices the *nonpoles* of G. As in previous studies, we visualize G in such a way that s is at the South Pole while t is at the North Pole.

The poles of G divide the boundary of the outer face into two paths, which are called the *west side* and the *east side* of G, respectively. In the dual graph $G^* = (V^*, E^*)$ of G, there are two dual vertices s^* and t^* corresponding to the outer face of G. The dual edges incident to s^* (respectively, t^*) in G^* correspond to the edges on the west (respectively, east) side of G. Figure 1.1(2) shows a TTG G and its dual G^*. G^* is a TTG with source s^* and sink t^* [12].

In complimentary metal-oxide semiconductor (CMOS) technology, the basic layout of a circuit C of transistors on a VLSI chip uses two rows of transistors: A row of pMOS transistors is laid out next to a row of nMOS transistors [13]. The circuit C can be represented by a pair of TTGs: The p-transistors are represented by G, where each edge of G represents a p-transistor. The n-transistors are represented by the dual graph G^* of G, where each edge of G^* represents an n-transistor. In Figure 1.1, (1) shows a circuit implementing the Boolean function $z = \bar{e} \wedge (\bar{a} \vee \bar{b}) \wedge (\bar{c} \vee \bar{d})$, and (2) shows the corresponding TTGs G and G^*.

*Received by the editors April 2, 1999; accepted for publication (in revised form) December 4, 2001; published electronically May 8, 2002. A preliminary version of this work was presented at the 40th Annual Symposium on Foundations of Computer Science, 1999, New York City, New York.
http://www.siam.org/journals/sicomp/31-4/35432.html

†Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan (chen@r.dendai.ac.jp).

‡Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (xinhe@cse.buffalo.edu). This author's research was supported in part by NSF grant CCR-9912418.

§Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269 (huang@cse.uconn.edu).

[1] Previously called *planar st-graphs* and treated as digraphs just for convenience.

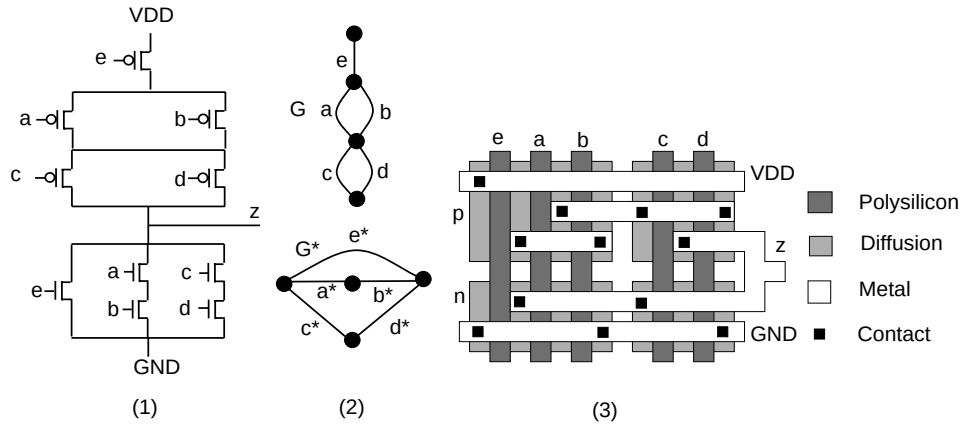


FIG. 1.1. (1) *A circuit C implementing $z = \bar{e} \wedge (\bar{a} \vee \bar{b}) \wedge (\bar{c} \vee \bar{d})$.* (2) *The corresponding TTGs G and G^*.* (3) *A layout of C.*

1.1. Double Euler trails and optimal layouts. A *trail* P in G is a sequence $e_1 e_2 \dots e_k$ of distinct and oriented edges of G such that, for each $i \in \{1, \dots, k-1\}$, the ending vertex of e_i and the starting vertex of e_{i+1} are the same. Possibly, P has repeated vertices. An *Euler trail* is a trail that visits each edge of G exactly once. For each edge e of G, let e^* be the dual edge of e in G^*. If the sequence $P^* = e_1^* e_2^* \dots e_k^*$ is also a trail in G^*, we call P a *double trail* [2] in G and call P^* the *dual trail* of P. If P is a double trail and traverses every edge of G exactly once, we call P a *double Euler trail* (DET) in G.

For each edge e in G, the p-transistor represented by e has the same input as the n-transistor represented by e^*; hence, in the layout, the two transistors are required to be laid out vertically aligned (to facilitate input line connection) [13]. Transistors which are physically adjacent and have common diffusion regions may be collapsed. Otherwise, a vertical gap must be inserted between the diffusion region of the two transistors. Two transistors corresponding to two consecutive edges on a double trail can be laid out physically adjacent without vertical gap [13]. Figure 1.1(3) shows a layout of C using this approach. Note that the sequence eab is a double trail of G. So, in the layout, no vertical gaps are needed between the transistors e, a, b. This is also true for transistors c, d because of the double trail cd. However, since the edges b and c are incident in G but not in G^*, there is a vertical gap between the transistors b and c in the layout. In Figure 1.2, (1) shows a logically equivalent circuit and (2) shows the corresponding TTGs G and G^*. Since $abecd$ is a DET of G, the transistors can be laid out in this order with no vertical gap, as seen in Figure 1.2(3).

1.2. Problem definitions. The circuits in Figures 1.1(1) and 1.2(1) are logically equivalent. Thus, the TTGs G in Figures 1.1(2) and 1.2(2) should be viewed as different embeddings of each other. That is, to obtain a different embedding of a TTG, we may not only horizontally flip blocks of the TTG as one does traditionally but also vertically flip blocks of the TTG. We postpone the precise definition of "different embeddings" until the end of section 2. For convenience, we regard G as a different embedding of itself.

[2] Previously called "dual trail." We reserve the term "dual trail" exclusively to refer to a trail in G^*.

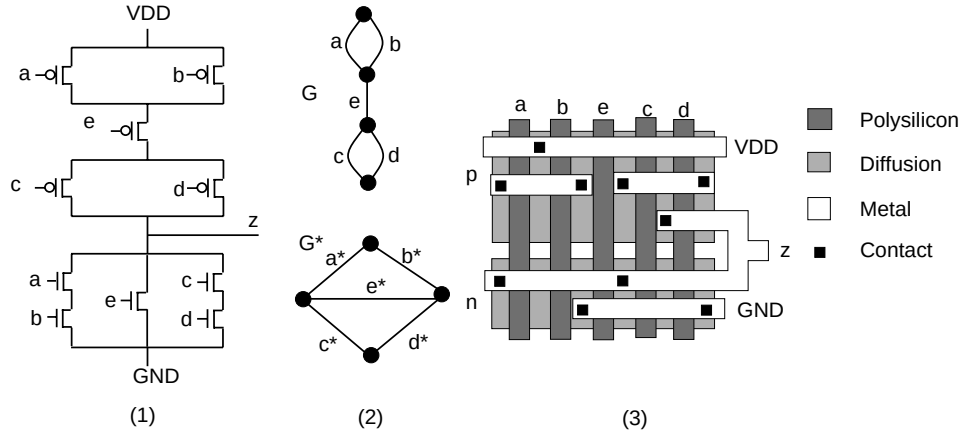


FIG. 1.2. (1) *A circuit C logically equivalent to the circuit in Figure* 1.1(1). (2) *The corresponding TTGs G and G*.* (3) *A layout of C using the DET abecd.*

As seen from Figures 1.1 and 1.2, even if G has no DET, a different embedding of G may have one. A *valid embedding* of G is a different embedding of G that has a DET. A *double-trail cover* (DTC) $\mathcal{Q}$ of G is a set of edge-disjoint double trails in G such that each edge of G appears in a trail in $\mathcal{Q}$. The *size* of $\mathcal{Q}$ is the number of double trails in it. $\mathcal{Q}$ is a *minimum* DTC of G if the size of $\mathcal{Q}$ is minimized over all DTCs of G. A different embedding of G with a minimum DTC leads to a layout with the minimum number of vertical gaps (hence the minimum chip area). We are interested in the following problems:

DET-F problem. Given a TTG G, decide whether G has a DET.

DET-V problem. Given a TTG G, decide whether G has a valid embedding.

DTC-F problem. Given a TTG G, find a minimum DTC of G.

DTC-V problem. Given a TTG G, find a different embedding $\mathcal{G}$ of G and a minimum DTC $\mathcal{Q}$ of $\mathcal{G}$ such that the size of $\mathcal{Q}$ is minimized over all different embeddings of G.

1.3. Previous work and our result. The layout method described above was first introduced by Uehara and VanCleemput [13]. Most previous studies in this area are concerned with *series-parallel* (SP) graphs, a much restricted subclass of TTGs. This is partially because the problems above for general TTGs are more difficult to solve. A heuristic algorithm for solving the DTC-V problem for SP graphs was given in [13]. Lengauer and Müller [7] developed linear-time algorithms for finding a minimum trail cover (not DTC) for SP graphs. Nair, Buss, and Reif [10] presented polynomial-time algorithms for solving the DTC-F problem and the DTC-V problem both for a restricted class of embedded SP graphs. Maziasz and Hayes [8, 9] gave a linear-time algorithm for solving the DTC-F problem for SP graphs and a worst-case exponential-time algorithm for the DTC-V problem for SP graphs. Linear-time algorithms for solving the DET-V problem for SP graphs were given in [5]. These algorithms are based on dynamic programming and are complicated. Whether the DTC-V problem for SP graphs can be solved in polynomial time is still open.

For general TTGs, a linear-time algorithm for solving the DET-F problem was given in [1]. The DTC-F problem is NP-hard [14, 15], which implies that the DTC-V problem for TTGs is also NP-hard. Whether the DET-V problem for general TTGs

can be solved in *polynomial time* is posed as an open question in [1].

Here, we answer this question in the affirmative by presenting a *linear-time* algorithm for the DET-V problem. Our algorithm is based on careful analysis of the structures of DTCs of TTGs. In more details, we first find out the types of those DTCs that are the intersection of a DET of a TTG G and a subgraph of G. We then define two operations for composing such DTCs of edge-disjoint subgraphs into a single such DTC of a larger subgraph. We further prove that the two operations have nice properties based on which a linear-time algorithm can be designed. The main data structure we use is the *SPQR-decomposition tree* of a TTG [3].

1.4. Organization of this paper. Section 2 describes the *SPQR decomposition* of a TTG. Section 3 shows several useful properties of different embeddings of a TTG. Section 4 discusses the special case of the DET-V problem where neither the input TTG nor its dual graph has an odd-degree vertex. Section 5 gives a polynomial-time algorithm for the DET-V problem. Section 6 refines this algorithm to a linear-time algorithm.

2. SPQR decomposition. Throughout this paper, for a TTG G, G^* denotes the dual graph of G, $s(G)$ denotes the source of G, $t(G)$ denotes the sink of G, and $|G|$ denotes the total number of vertices and edges in G. For an unordered list L, we write $L = \langle a_1, \ldots, a_k \rangle$, where a_1 through a_k are the elements of L. For an ordered list L, we write $L = (a_1, \ldots, a_k)$, where $a_1, \ldots, a_k$ are the elements of L and are ordered in this order.

In the rest of this section, fix a TTG G. A *cut vertex* of G is a vertex whose removal disconnects G. G is *biconnected* if it has no cut vertex. A *biconnected component* of G is a maximal subgraph of G with no cut vertex. A *split pair* of G is either a pair of adjacent vertices or a pair of vertices whose removal disconnects G', where G' is the graph obtained from G by adding the edge $\{s(G), t(G)\}$. A *split component* of a split pair $\langle u, v \rangle$ is either an edge $\{u, v\}$ or a maximal subgraph H of G such that H is a TTG with poles u and v and $\langle u, v \rangle$ is not a split pair of H. A split pair $\langle u, v \rangle$ of G is *maximal* if there is no other split pair $\langle w_1, w_2 \rangle$ in G such that $\langle u, v \rangle$ is contained in a split component of $\langle w_1, w_2 \rangle$.

The *decomposition tree* T of G describes a recursive decomposition of G with respect to its split pairs [2, 3]. T is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. Each node μ of T has an associated TTG, called the *skeleton* of μ and denoted by $\mathrm{Sk}(\mu)$. Also, it is associated with an edge in the skeleton of the parent χ of μ, called the *virtual edge* of μ in $\mathrm{Sk}(\chi)$. T is recursively defined as follows.

Trivial case. G consists of a single edge $\{s(G), t(G)\}$. Then, T consists of a single Q-node whose skeleton is G itself.

Series case. G is not biconnected. Let $c_1, \ldots, c_{k-1}$ ($k \geq 2$) be the cut vertices of G. Since G is a TTG, each $c_i \in \{c_1, \ldots, c_{k-1}\}$ is contained in exactly two biconnected components G_i and G_{i+1} of G such that $s(G)$ is in G_1 and $t(G)$ is in G_k. The root of T is an S-node μ. $\mathrm{Sk}(\mu)$ is a path $e_1, \ldots, e_k$, where the edge $e_i = \{c_{i-1}, c_i\}$, $c_0 = s(G)$, and $c_k = t(G)$.

Parallel case. $s(G)$ and $t(G)$ constitute a split pair of G with split components $G_1, \ldots, G_k$ ($k \geq 2$). Then, the root of T is a P-node μ. $\mathrm{Sk}(\mu)$ consists of k parallel edges between $s(G)$ and $t(G)$, denoted by $e_1, \ldots, e_k$.

Rigid case. None of the above cases applies. Let $\langle s_1, t_1 \rangle, \ldots, \langle s_k, t_k \rangle$ ($k \geq 1$) be the maximal split pairs of G. For $i = 1, \ldots, k$, let G_i be the union of all the split components of $\langle s_i, t_i \rangle$. The root of T is an R-node μ. $\mathrm{Sk}(\mu)$ is obtained from G by replacing each subgraph G_i with an edge $e_i = \{s_i, t_i\}$. Note that adding the edge

$\{s(G), t(G)\}$ to $\mathrm{Sk}(\mu)$ yields a triconnected graph. By this, $\mathrm{Sk}(\mu)$ has no different embedding other than itself [11].

In the last three cases, μ has children $\nu_1, \ldots, \nu_k$ (in this order) such that ν_i is the root of the decomposition tree of graph G_i for all $i \in \{1, \ldots, k\}$. The virtual edge of node ν_i is the edge e_i in $\mathrm{Sk}(\mu)$. G_i is called the *pertinent graph* of ν_i and is denoted by $\mathrm{Pt}(\nu_i)$. Note that G is the pertinent graph of the root of T. Figure 2.1 illustrates the decomposition tree of a TTG G. The skeletons of the nodes μ and ν are also shown.

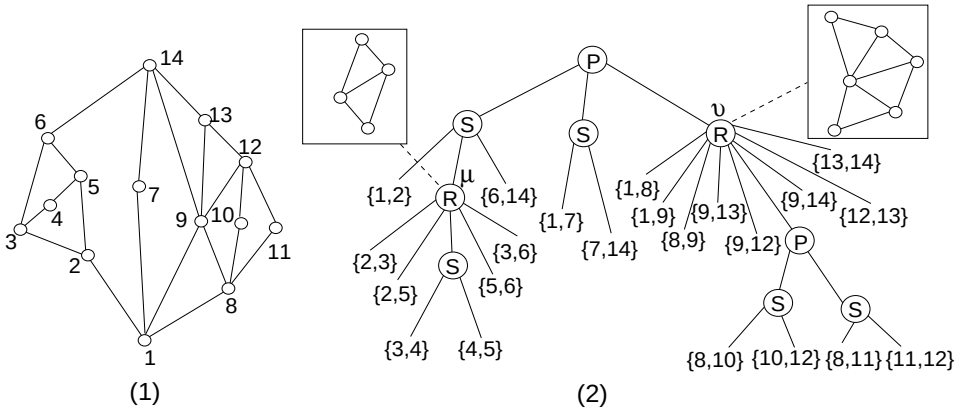


FIG. 2.1. (1) *A TTG G.* (2) *Its decomposition tree T.*

By the definition of T, no child of an S-node is an S-node and no child of a P-node is a P-node.

LEMMA 2.1 (see [3]). *T has $O(|G|)$ nodes. Also, the total number of edges of the skeletons stored at the nodes of T is $O(|G|)$.*

The *dual decomposition tree* of T is the decomposition tree of the dual graph of G.

LEMMA 2.2. *The dual decomposition tree of T can be obtained from T by exchanging the roles of the S-nodes and the P-nodes and further changing the skeleton of each node to the dual graph of the skeleton.*

Proof. The proof is obvious. □

A *different embedding* of G is obtained from G by a sequence of operations from the following set [7, 8, 9, 10, 13]:

- horizontally flipping the skeleton of an R-node around its poles;
- permuting the children of a P-node;
- vertically flipping the skeleton of an R-node, which is equivalent to horizontally flipping the dual skeleton around its poles;
- permuting the children of an S-node, which is equivalent to permuting the children of the dual P-node.

Note that if $\mathcal{G}$ is a different embedding of a TTG G, the circuits represented by $\mathcal{G}$ and G are logically equivalent. Also note that the commonly known definition of "different embeddings" allows only the first two operations [2, 3]. Hence, the meaning of the phrase "different embedding" used here is different from its commonly known meaning.

3. Properties of embeddings and double trails. Throughout this paper, a list is ordered unless stated explicitly otherwise. An unordered list is actually a multiset. So, for two unordered lists $L_1 = \langle a_1, \ldots, a_k \rangle$ and $L_2 = \langle b_1, \ldots, b_\ell \rangle$, we write

$L_1 \cup L_2$ for the unordered list $\langle a_1, \ldots, a_k, b_1, \ldots, b_\ell \rangle$. For an ordered (respectively, unordered) list L and a set S, let $L - S$ denote the ordered (respectively, unordered) list obtained from L by deleting all appearances of each $x \in S$.

The following fact is well known.

FACT 1. *For a connected graph G, the following two statements hold:*

- *G has a closed Euler trail if and only if it has no odd-degree vertex; in this case G is called* Eulerian.
- *G has an open Euler trail if and only if it contains exactly two odd-degree vertices, which are the first and the last vertex of the trail.*

In the rest of this section, fix a TTG G. Let T be the decomposition tree of G. A *block* of G is either the pertinent graph of a node in T or the union of the pertinent graphs of some consecutive children of a P- or S-node in T. A *proper block* of G is a block B such that $G \neq B$. Let $B = \mathrm{Pt}(\mu)$, where μ is a node of T. For every child ν of μ in T, we call $\mathrm{Pt}(\nu)$ a *child block* of B.

The following properties of double trails are clear. They play very important roles in our algorithm.

Noncrossing property. Let Q be a double trail of G. Let e' and e'' be two consecutive edges in Q. Then, e' and e'' appear consecutively around v in G, where v is the ending vertex of e' in Q.

Let B be a block of G. Let e_1 (e_2, respectively) be the edge in B incident to $s(B)$ that is on the west (east, respectively) side of B. (If there is only one edge in B incident to $s(B)$ then $e_1 = e_2$.) Let e_3 (e_4, respectively) be the edge in B incident to $t(B)$ that is on the west (east, respectively) side of B. (If there is only one edge in B incident to $t(B)$, then $e_3 = e_4$.) We call e_1, e_2, e_3, e_4 the *access edges* of B. Each access edge e_i is incident to a pole of B and its dual edge e_i^* is incident to a pole of B^*; only access edges of B satisfy this property.

Access-edge property. Let Q be any double trail of G. Let e', e'' be two consecutive edges in Q such that e' is in B and e'' is not in B. Then, e' must be an access edge of B.

LEMMA 3.1 (see [1]). *Let $e = \{u, v\}$ be an edge in G and $e^* = \{F_1, F_2\}$ its dual edge. Let Q be a maximal double trail in G starting with e. If the direction of e in Q and the direction of e^* in the dual trail Q^* are given, then Q is uniquely determined and can be traced out in $O(|Q|)$ time.*

Proof (sketch). Suppose that Q traverses e from u to v while Q^* traverses e^* from F_1 to F_2. Then, the successor of e in Q must be the edge incident to v and on the face F_2 of G. Repeating this argument, we can uniquely trace out all edges of Q. □

The *degree parity* of a vertex in G is 0 if its degree is even and is 1 otherwise.

FACT 2. *If G is Eulerian, then so is every different embedding of G.*

Proof. Let μ be a node of T, and $B = \mathrm{Pt}(\mu)$. Clearly, no vertex gets a different degree after B is horizontally flipped. Similarly, when μ is a P-node, no vertex gets a different degree after the children of μ are permuted.

Suppose that μ is an R-node. After B is vertically flipped, only the poles $s(B)$ and $t(B)$ may get a different degree in the resulting embedding. Let m_B be the number of edges in B. For each vertex v in B, let $d_B(v)$ be the degree of v in B. Thus, $2m_B = d_B(s(B)) + d_B(t(B)) + \sum_{v \neq s(B), t(B)} d_B(v)$. Since $d_B(v)$ is even for all $v \notin \{s(B), t(B)\}$, $d_B(s(B))$ and $d_B(t(B))$ must have the same parity. Thus, vertically flipping B creates no odd-degree vertex.

Next, suppose μ is an S-node. Let $B_1, \ldots, B_k$ be the child blocks of B. Since no B_i contains an odd-degree nonpole, the two poles $s(B_i)$ and $t(B_i)$ must have the same

degree parity within B_i (as proved in the last paragraph). Suppose that $d_{B_1}(s(B_1))$ is even. Then, $d_{B_1}(t(B_1))$ must be even. Since $t(B_1) = s(B_2)$ and $d_B(t(B_1)) = d_{B_1}(t(B_1)) + d_{B_2}(s(B_2))$ is even, $d_{B_2}(s(B_2))$ must be even. Continuing this argument, we can see that all $s(B_i)$ and $t(B_i)$ must have even degree within B_i. By this, permuting the children of μ creates no odd-degree vertex. Similarly, if $d_{B_1}(s(B_1))$ is odd, then all $s(B_i)$ and $t(B_i)$ must have odd degree within B_i. Thus, permuting the children of μ creates no odd-degree vertex. □

For a trail P in G and a proper block B of G, the *intersection* of P and B, denoted by $P \cap B$, is the list $(Q_1, Q_2, \ldots, Q_k)$, where Q_1 is the first (maximal) subtrail of P in B, Q_2 is the second (maximal) subtrail of P in $B, \ldots,$ and Q_k is the last (maximal) subtrail of P in B. An *ordered* DTC (ODTC) of G is a list $(P_1, \ldots, P_\ell)$ such that $\{P_1, \ldots, P_\ell\}$ is a DTC of G. For an ODTC $\mathcal{P} = (P_1, \ldots, P_\ell)$ of G and a proper block B of G, the *intersection* of $\mathcal{P}$ and B, denoted by $\mathcal{P} \cap B$, is the list $(Q_{1,1}, \ldots, Q_{1,k_1}, \ldots, Q_{\ell,1}, \ldots, Q_{\ell,k_\ell})$, where $(Q_{i,1}, \ldots, Q_{i,k_i})$ is the intersection of P_i and B for all $i \in \{1, \ldots, \ell\}$. Clearly, $\mathcal{P} \cap B$ is an ODTC of B.

Let $\varphi_G : V \to \{1, 2, 3\}$ be the function that maps $s(G)$ to 1, maps $t(G)$ to 2, and maps each nonpole of G to 3. The *starting pair* (respectively, *ending pair*) of a double trail P in G is the pair $(\varphi_G(u), \varphi_{G^*}(x))$, where u is the starting (respectively, ending) vertex of P and x is the starting (respectively, ending) vertex of the dual trail of P. The *extreme quadruple* of a double trail P in G is the quadruple (p_1, p_2, p_3, p_4), where (p_1, p_2) is the starting pair of P and (p_3, p_4) is the ending pair of P. The *σ-extreme list* of an ODTC $(P_1, \ldots, P_k)$ is the list $(q_1, \ldots, q_k)$, where q_i is the extreme quadruple of P_i for all $i \in \{1, \ldots, k\}$. The *π-extreme list* of an ODTC $(P_1, \ldots, P_k)$ is the list $(p_1, q_2, \ldots, q_k)$, where p_1 is the ending pair of P_1 and q_i is the extreme quadruple of P_i for all $i \in \{2, \ldots, k\}$. The *δ-extreme list* of an ODTC $(P_1, \ldots, P_k)$ is the list $(p_1, q_2, \ldots, q_{k-1}, p_k)$, where p_1 is the ending pair of P_1, p_k is the starting pair of P_k, and q_i is the extreme quadruple of P_i for all $i \in \{2, \ldots, k-1\}$.

LEMMA 3.2. *Let B be a block of G. Let $\mathcal{P} = (P_1, \ldots, P_\ell)$ be an ODTC of G. Let D be a TTG, and let $\mathcal{Q} = (Q_1, \ldots, Q_k)$ be an ODTC of D. Let H be the TTG obtained from G by replacing B with D. Then, the following three statements hold:*

1. *Suppose that* (1) *the σ-extreme list of $\mathcal{Q}$ equals that of $\mathcal{P} \cap B$ and* (2) *no trail in $\mathcal{P}$ starts or ends with an edge of B. Then, H has an ODTC $\mathcal{P}'$ whose σ-extreme list equals that of $\mathcal{P}$.*
2. *Suppose that* (1) *the π-extreme list of $\mathcal{Q}$ equals that of $\mathcal{P} \cap B$ and* (2) *P_1 starts with an edge of B while no trail in $\mathcal{P} - \{P_1\}$ starts or ends with an edge of B. Then, H has an ODTC $\mathcal{P}'$ whose π-extreme list equals that of $\mathcal{P}$.*
3. *Suppose that* (1) *the δ-extreme list of $\mathcal{Q}$ equals that of $\mathcal{P} \cap B$ and* (2) *P_1 starts with an edge of B and P_ℓ ends with an edge of B while no trail in $\mathcal{P} - \{P_1, P_\ell\}$ starts or ends with an edge of B. Then, H has an ODTC $\mathcal{P}'$ whose δ-extreme list equals that of $\mathcal{P}$.*

Proof. For each $P_i \in \mathcal{P}$, if $P_i \cap B$ is empty, then let $P'_i = P_i$; otherwise, let P'_i be the trail obtained from P_i by replacing each $S_j \in P_i \cap B$ with Q_j, where S_j is the jth trail in $\mathcal{P} \cap B$. Then, $P' = (P'_1, \ldots, P'_\ell)$ is the required ODTC of H. □

4. The case where G and G^* are Eulerian. Throughout this section, fix an Eulerian TTG G whose dual graph is also Eulerian. Let T be the decomposition tree of G. By Fact 2, each different embedding $\mathcal{G}$ of G and its dual $\mathcal{G}^*$ are still Eulerian. So, each DET P of a valid embedding of G must start and end at the same vertex, and so does the dual trail of P.

4.1. Types of DTCs of proper blocks. Suppose that G has a valid embedding $\mathcal{G}$ and P is a DET of $\mathcal{G}$. Fix a proper block B of $\mathcal{G}$. Since P starts and ends at the same vertex and so does the dual trail of P, we may assume that the starting edge of P is not in B. Then, each double trail in $P \cap B$ must start and end at poles of B. Let $P \cap B = (P_1, \ldots, P_k)$. For each $i \in \{1, \ldots, k\}$, let P_i^* be the dual trail of P_i.

LEMMA 4.1.

1. *If $k \geq 2$, then each $P_i \in \{P_1, \ldots, P_k\}$ uses two distinct access edges of B. Consequently, $k \leq 2$.*
2. *If P_i starts and ends at the same pole of B, then P_i^* must start and end at different poles of B^*.*
3. *If $k = 2$ and P_1 starts and ends at different poles of B, then P_1^* must start and end at the same pole of B^*.*

Proof. Let e_1 and e_2 be the two access edges that are incident to $s(B)$ and on the west side and the east side of B, respectively. Let e_3 and e_4 be the two access edges that are incident to $t(B)$ and on the west side and the east side of B, respectively. We prove the three statements separately as follows.

Statement 1. Suppose $k \geq 2$. Consider an arbitrary $P_i \in \{P_1, \ldots, P_k\}$. Let e (respectively, e') be the starting (respectively, ending) edge of P_i. Since the starting edge of P is not in B, $e \in \{e_1, \ldots, e_4\}$ by the access-edge property. Even if e' is the ending edge of P, since P is an Euler trail, e' must be incident to the starting edge of P which is not in B by our assumption. So, $e' \in \{e_1, \ldots, e_4\}$. Thus, we are done if $e \neq e'$. Towards a contradiction, assume $e = e'$. Then, $e = \{s(B), t(B)\}$ is the unique edge of P_i. In turn, $\{s(B^*), t(B^*)\}$ is the unique edge of P_i^*. It follows that B consists of a single edge, contradicting the assumption that $k \geq 2$. Therefore, $e \neq e'$ and P_i uses two distinct access edges of B. Since B has at most four access edges (namely e_1 through e_4), $k \leq 2$.

Statement 2. Without loss of generality, suppose that P_i starts and ends at $s(B)$. Then, $e_1 \neq e_2$. By the access-edge property, we can assume that P_i starts with e_1 and ends with e_2. Hence, P_i^* starts at the west side and ends at the east side of B. (See Figure 4.1(1).)

Statement 3. Suppose that $k = 2$ and P_1 starts and ends at different poles of B. Then, by statement 1, e_1 through e_4 are distinct edges. Without loss of generality, assume that P_1 starts with e_1. Towards a contradiction, suppose that P_1 ends with e_4. Then, P_2 starts with e_2 and ends with e_3. Since B is a plane graph, P_1 and P_2 must cross each other somewhere inside B. However, this is impossible by the noncrossing property. Hence, P_1 ends with e_3; in turn P_1^* starts and ends at the west side of B. (See Figure 4.1(2).) □

Based on Lemma 4.1, $P \cap B$ can be only of the following five possible types:

- *Type* σ_1. $k = 1$; P_1 starts at one pole and ends at the other pole, and so does P_1^*.
- *Type* σ_2. $k = 1$; P_1 starts at one pole and ends at the same pole while P_1^* starts at one pole and ends at the other pole.
- *Type* σ_3. $k = 1$; P_1 starts at one pole and ends at the other pole while P_1^* starts at one pole and ends at the same pole.
- *Type* σ_4. $k = 2$; P_1 starts at one pole u and ends at u while P_2 starts at the other pole v and ends at v; P_1^* starts at one pole and ends at the other pole, and so does P_2^*.
- *Type* σ_5. $k = 2$; P_1 starts at one pole and ends at the other pole, and so does P_2; P_1^* starts at one pole x and ends at x while P_2^* starts at the other pole y

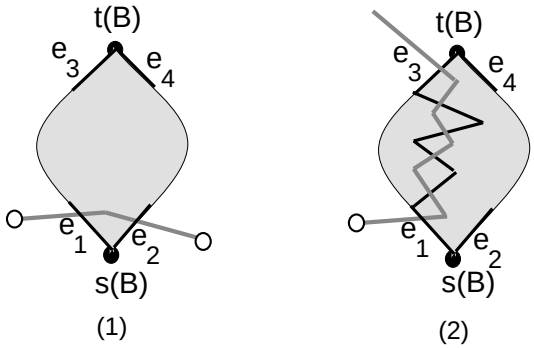


FIG. 4.1. *The proof of Lemma* 4.1.

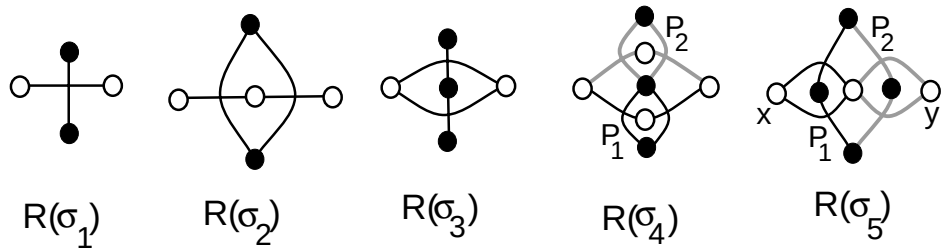


FIG. 4.2. *Representative graphs for* σ*-types.*

and ends at y.

Define $\Sigma = \{\sigma_1, \ldots, \sigma_5\}$. Each σ_i is called a *σ-type*. For a TTG H, let $\mathcal{K}_\sigma(H)$ denote the set of all $\sigma_i \in \Sigma$ such that a different embedding of H has a Type-σ_i ODTC. We call each member of $\mathcal{K}_\sigma(H)$ a *σ-type of H*. For each $\sigma_i \in \Sigma$, a *representative graph* $R(\sigma_i)$ is shown in Figure 4.2 in which the solid lines denote the edges in the trail P_1 and the dual trail P_1^*, while the light lines denote the edges in the trail P_2 and the dual trail P_2^*. Note that σ_i is the unique σ-type of $R(\sigma_i)$ for all $i \in \{1, \ldots, 5\}$.

4.2. Composing DTCs. Let H_1 and H_2 be two vertex-disjoint TTGs. The *series composition* of H_1 and H_2, denoted by $H_1 \star H_2$, is the TTG obtained from H_1 and H_2 by identifying $t(H_1)$ with $s(H_2)$, $s(H_1^*)$ with $s(H_2^*)$, and $t(H_1^*)$ with $t(H_2^*)$. The *parallel composition* of H_1 and H_2, denoted by $H_1 \diamond H_2$, is the TTG obtained from H_1 and H_2 by identifying $s(H_1)$ with $s(H_2)$, $t(H_1)$ with $t(H_2)$, and $t(H_1^*)$ with $s(H_2^*)$. For an integer $k \geq 3$ and k vertex-disjoint TTGs $H_1, \ldots, H_k$, let $H_1 \star H_2 \star \cdots \star H_k = (\cdots((H_1 \star H_2) \star H_3) \star \cdots H_{k-1}) \star H_k$ and $H_1 \diamond H_2 \diamond \cdots \diamond H_k = (\cdots((H_1 \diamond H_2) \diamond H_3) \diamond \cdots H_{k-1}) \diamond H_k$. We call $H_1 \star \cdots \star H_k$ (respectively, $H_1 \diamond \cdots \diamond H_k$) the *series* (respectively, *parallel*) *composition* of H_1 through H_k.

Define two binary operators $\star$ and $\diamond$ over Σ as in Tables 4.1 and 4.2; each empty entry in Table 4.1 (respectively, 4.2) indicates that the $\star$ (respectively, $\diamond$) operator is undefined for that entry.

LEMMA 4.2. *Both $\star$ and $\diamond$ are commutative and associative operators over Σ.*

Proof. One can use Tables 4.1 and 4.2 to verify. □

LEMMA 4.3. *Let H be a TTG. Then, the following two statements hold:*

1. *Suppose $H = H_1 \star H_2$, where $\mathcal{K}_\sigma(H_1) = \{\sigma_i\}$ and $\mathcal{K}_\sigma(H_2) = \{\sigma_j\}$. Then, $\mathcal{K}_\sigma(H) = \{\sigma_i \star \sigma_j\}$ if $\sigma_i \star \sigma_j$ is defined, while $\mathcal{K}_\sigma(H) = \emptyset$ otherwise.*

2. *Suppose* $H = H_1 \diamond H_2$, *where* $\mathcal{K}_\sigma(H_1) = \{\sigma_i\}$ *and* $\mathcal{K}_\sigma(H_2) = \{\sigma_j\}$. *Then,* $\mathcal{K}_\sigma(H) = \{\sigma_i \diamond \sigma_j\}$ *if* $\sigma_i \diamond \sigma_j$ *is defined, while* $\mathcal{K}_\sigma(H) = \emptyset$ *otherwise.*

Proof. By statement 1 in Lemma 3.2, it suffices to verify that for every $\sigma_i, \sigma_j \in \Sigma$, $\mathcal{K}_\sigma(R(\sigma_i) \star R(\sigma_j)) = \{\sigma_i \star \sigma_j\}$ and $\mathcal{K}_\sigma(R(\sigma_i) \diamond R(\sigma_j)) = \{\sigma_i \diamond \sigma_j\}$. For the verification, it is helpful to note that for each $\sigma_i \in \Sigma$, $R(\sigma_i)$ remains unchanged after a horizontal or vertical flipping. We show only one example for each of the $\star$ and the $\diamond$ operators. The other cases can be verified similarly.

TABLE 4.1
Operator $\star$ *over* Σ.

	σ_1	σ_2	σ_3	σ_4	σ_5
σ_1	σ_3		σ_1		
σ_2		σ_4			σ_2
σ_3	σ_1		σ_3		
σ_4					σ_4
σ_5		σ_2		σ_4	σ_5

TABLE 4.2
Operator $\diamond$ *over* Σ.

	σ_1	σ_2	σ_3	σ_4	σ_5
σ_1	σ_2	σ_1			
σ_2	σ_1	σ_2			
σ_3			σ_5	σ_3	
σ_4			σ_3	σ_4	σ_5
σ_5				σ_5	

Consider the graph $R(\sigma_1)$, with poles u and v, and the graph $R(\sigma_3)$ with poles v and w. Let x and y be the west side and the east side of their series composition $H' = R(\sigma_1) \star R(\sigma_3)$. (See Figure 4.3(1).) The only DTC of H' consists of a single double trail P_1 which starts at u, passing through v and ends at w, while the dual trail P_1^* starts at x, travels to y, to x, and to y again. This is a Type-σ_1 DTC. So $\mathcal{K}_\sigma(R(\sigma_1) \star R(\sigma_3)) = \{\sigma_1\} = \{\sigma_1 \star \sigma_3\}$ as to be shown.

Consider the graph $R(\sigma_4)$ with poles u, v, the west side x and the east side y; and the graph $R(\sigma_5)$ with poles u, v, the west side y and the east side z. (See Figure 4.3(2).) Let $H'' = R(\sigma_4) \diamond R(\sigma_5)$. The only DTC of H'' consists of two double trails P_1 and P_2 as shown in Figure 4.3(2). This is a Type-σ_5 DTC. So $\mathcal{K}_\sigma(R(\sigma_4) \diamond R(\sigma_5)) = \{\sigma_5\} = \{\sigma_4 \diamond \sigma_5\}$. □

Let μ be a node in T and $B = \mathrm{Pt}(\mu)$. For convenience, we also use $\mathcal{K}_\sigma(\mu)$ to denote $\mathcal{K}_\sigma(B)$.

LEMMA 4.4. *Let* μ *be a nonroot node of* T *and* $B = \mathrm{Pt}(\mu)$. *Then,* $|\mathcal{K}_\sigma(B)| \leq 1$. *Moreover,* $\mathcal{K}_\sigma(\mathrm{Pt}(\nu))$ *for all descendants* ν *(including* μ *itself) of* μ *in* T *can be computed in* $O(|B|)$ *time.*

Proof. The proof is by induction. In case μ is a leaf node of T, the lemma trivially holds because $\mathcal{K}_\sigma(B) = \{\sigma_1\}$. Suppose that μ is a nonleaf node of T. Let $\nu_1, \ldots, \nu_k$ be the children of μ in T. Let $B_i = \mathrm{Pt}(\nu_i)$ for all $i \in \{1, \ldots, k\}$. By the inductive hypothesis, $|\mathcal{K}_\sigma(B_i)| \leq 1$. If $\mathcal{K}_\sigma(B_i) = \emptyset$ for some $i \in \{1, \ldots, k\}$, then clearly $\mathcal{K}_\sigma(B) = \emptyset$. So, suppose that each $\mathcal{K}_\sigma(B_i)$ consists of a unique $\sigma_{j_i} \in \Sigma$. Consider the following three cases:

Case 1. μ is an R-node. Then, $\mathrm{Sk}(\mu)$ has no different embedding other than itself. Let H be the TTG obtained from $\mathrm{Sk}(\mu)$ by replacing the virtual edge of ν_i in $\mathrm{Sk}(\mu)$

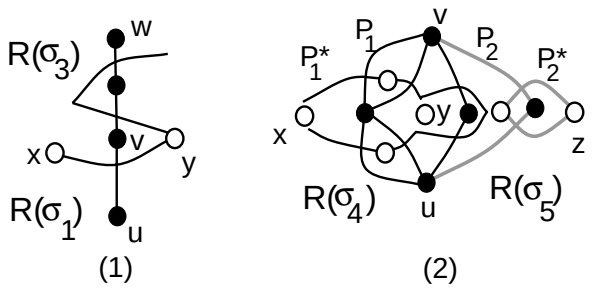


FIG. 4.3. *The proof of Lemma* 4.3.

with $R(\sigma_{j_i})$ for all $i \in \{1, \ldots, k\}$. Since $\mathcal{K}_\sigma(B_i) = \mathcal{K}_\sigma(R(\sigma_{j_i}))$ for all $i \in \{1, \ldots, k\}$, by repeatedly applying statement 1 in Lemma 3.2, we have $\mathcal{K}_\sigma(\mu) = \mathcal{K}_\sigma(H)$. $\mathcal{K}_\sigma(H)$ can be computed as follows.

Let P_1 be the maximal double trail in H which starts at $s(H)$ and whose dual trail P_1^* starts at $s(H^*)$. Let M be the set of all edges of H not traversed by P_1. In case $M = \emptyset$, we check if $\{P_1\}$ is a DTC of H of a σ-type; if it is, then $\mathcal{K}_\sigma(\mu)$ consists of this σ-type, while $\mathcal{K}_\sigma(\mu) = \emptyset$ otherwise. Suppose $M \neq \emptyset$. If M contains no access edge of H, then $\mathcal{K}_\sigma(\mu) = \emptyset$. Otherwise, let $e \in M$ be an access edge of H. We find the maximal double trail P_2 in H such that e and e^* are, respectively, the starting edge of P_2 and its dual trail P_2^*. If $\{P_1, P_2\}$ is a DTC of H of a σ-type, then $\mathcal{K}_\sigma(\mu)$ consists of this σ-type, while $\mathcal{K}_\sigma(\mu) = \emptyset$ otherwise.

Case 2. μ is an S-node. Then, each different embedding $\mathcal{B}$ of B can be written as $\mathcal{B}_{1'} \star \cdots \star \mathcal{B}_{k'}$, where $(1', 2', \ldots, k')$ is a permutation of $(1, 2, \ldots, k)$ and $\mathcal{B}_{i'}$ is a different embedding of $B_{i'}$ for all $i \in \{1, \ldots, k\}$. By Lemma 4.3, if $(\cdots((\sigma_{j_{1'}} \star \sigma_{j_{2'}}) \star \sigma_{j_{3'}}) \star \cdots) \star \sigma_{j_{k'}}$ is undefined, then $\mathcal{K}_\sigma(\mathcal{B}) = \emptyset$; otherwise, by Lemma 4.2, $\mathcal{K}_\sigma(\mathcal{B}) = \{(\cdots((\sigma_{j_1} \star \sigma_{j_2}) \star \sigma_{j_3}) \star \cdots) \star \sigma_{j_k}\}$. Thus, $|\mathcal{K}_\sigma(B)| \leq 1$.

Case 3. μ is a P-node. It is similar to Case 2. □

4.3. The algorithm. We process the nodes of T in post-order. When processing a nonroot node μ of T, we compute $\mathcal{K}_\sigma(\mu)$ as in the proof of Lemma 4.4. Let μ be the root of T. Let $\nu_1, \ldots, \nu_k$ be the children of μ in T. For each $i \in \{1, \ldots, k\}$, let $B_i = \mathrm{Pt}(\nu_i)$. If $\mathcal{K}_\sigma(B_i) = \emptyset$ for some $i \in \{1, \ldots, k\}$, then G has no valid embedding and we stop immediately. So, suppose that $\mathcal{K}_\sigma(B_1) = \{\sigma_{j_1}\}, \ldots, \mathcal{K}_\sigma(B_k) = \{\sigma_{j_k}\}$. We distinguish three cases as follows.

- *Case* 1. μ is an R-node. Then, $\mathrm{Sk}(\mu)$ has no different embedding other than itself. As in Case 1 in the proof of Lemma 4.4, we construct a TTG H from $\mathrm{Sk}(\mu)$. By the same argument as in Case 1 in the proof of Lemma 4.4, G has a DET if and only if H has a DET.

  To test if H has a DET, we pick an arbitrary edge e in H and fix a direction as the starting point. The dual edge e^* can be directed in two ways. Once the direction of e^* is fixed, we can uniquely trace the dual trail by Lemma 3.1.

  More specifically, let e be the edge incident to $s(B)$ whose dual edge e^* is incident to $s(H^*)$. Let x be the endpoint of e^* other than $s(H^*)$. Let P (respectively, Q) be the maximal double trail in H which starts at $s(H)$ and whose dual trail starts at $s(H^*)$ (respectively, x). By Lemma 3.1, P and Q can be found in $O(|\mathrm{Sk}(\mu)|)$ time. By statement 1 in Lemma 3.2, if either P or Q is a DET in H, then G has a DET. If neither P nor Q is a DET in H,

then H has no DET, and hence G has no DET either.

- *Case* 2. μ is an S-node. First we prove the following claim.
  Claim. For a TTG $H = H_1 \star H_2$ with $|\mathcal{K}_\sigma(H_1)| \leq 1$ and $|\mathcal{K}_\sigma(H_2)| \leq 1$, H has a valid embedding if and only if $\mathcal{K}_\sigma(H_1) = \mathcal{K}_\sigma(H_2) = \{\sigma_2\}$.
  To see the "if" part of the claim, it suffices to observe that a series composition of two copies of $R(\sigma_2)$ has a valid embedding. To see the "only if" part, suppose that H has a DET P. Then, P is obtained by concatenating the double trail(s) in the DTC of H_1 and the double trail(s) in the DTC of H_2. This can be done only in the following two possible ways.
  (a) The DTC of H_1 consists of a single double trail P_1 starting and ending at the same pole of H_1, and the DTC of H_2 consists of a single double trail P_2 starting and ending at the same pole of H_2. That is, $\mathcal{K}_\sigma(H_1) = \mathcal{K}_\sigma(H_2) = \{\sigma_2\}$. In this case, if H is obtained by identifying the common pole where both P_1 and P_2 start and end, we indeed get a DET of H.
  (b) The DTC of H_1 consists of a single double trail P_1 starting and ending at the same pole of H_1, and the DTC of H_2 consists of two double trails P_2 and P_3, each starting and ending at the different poles of H_2. (In this case, by concatenating P_2, P_1, P_3 in this order, we get a closed Euler trail P of H.) That is, $\mathcal{K}_\sigma(H_1) = \{\sigma_2\}$ and $\mathcal{K}_\sigma(H_2) = \{\sigma_5\}$. In this case, however, the dual trail P^* of P starts at one pole of the dual graph H^* and ends at the other pole of H^*. (This can be verified by using the representative graphs in Figure 4.2.) Hence, P^* is not a closed Euler trail of H^*. So, this case is impossible. This completes the proof of the claim.
  Observe that for each nonempty proper subset $\{1', \ldots, h'\}$ of $\{1, \ldots, k\}$, $B_{1'} \star \cdots \star B_{h'}$ has at most one σ-type. This follows from Lemmas 4.2 and 4.3. By the claim and this observation, G has a valid embedding if and only if $\{1, \ldots, k\}$ can be partitioned into two sets $\{1', \ldots, h'\}$, $\{1'', \ldots, g''\}$ such that σ_2 is the σ-type of both $B_{1'} \star \cdots \star B_{h'}$ and $B_{1''} \star \cdots \star B_{g''}$. By Table 4.1, such a partition is possible if and only if (1) σ_2 is the σ-type of exactly two of $B_1, \ldots, B_k$ and (2) σ_5 is the σ-type of all the rest.
- *Case* 3. μ is a P-node. It suffices to modify the argument in Case 2 by replacing each symbol "$\star$" with "$\diamond$," "σ_2" with "σ_3," and "σ_5" with "σ_4."

By Lemmas 2.1 and 4.4, the time needed to process the entire tree T is $O(|G|)$. Thus, we have the following.

THEOREM 4.5. *Given an Eulerian TTG G whose dual graph is also Eulerian, it takes $O(|G|)$ time to decide if G has a valid embedding.*

5. An $O(|G|^3)$-time algorithm. Throughout the rest of this paper, let G be a TTG, and let T be its decomposition tree. The *reverse* of a trail $e_1 e_2 \ldots e_k$ in G is the trail $e_k e_{k-1} \ldots e_1$. Section 5.1 investigates how a DET of G can intersect a block of G. Section 5.2 (respectively, 5.3) shows how a DET P can intersect the series (respectively, parallel) composition of two or more blocks of G, given the way that P intersects the blocks. Section 5.4 describes the algorithm for deciding the existence of a DET.

5.1. Types of DTCs of proper blocks. Suppose that G has a valid embedding $\mathcal{G}$ and P is a DET of $\mathcal{G}$. Let e_s and e_t be the starting and the ending edge of P, respectively. Fix a proper block B of $\mathcal{G}$. Let $P \cap B = (P_1, \ldots, P_k)$. For each $i \in \{1, \ldots, k\}$, let P_i^* be the dual trail of P_i.

Case 1. Neither e_s nor e_t is in B. Then, as discussed in section 4.1, $P \cap B$ can only be of a σ-type.

Case 2. Exactly one of e_s and e_t is in B. If e_s is in B, let $Q_i = P_i$ for all $i \in \{1, \ldots, k\}$; otherwise, let Q_i be the reverse of P_{k-i+1} for all $i \in \{1, \ldots, k\}$. Let Q_i^* be the dual trail of Q_i for each $i \in \{1, \ldots, k\}$.

Without loss of generality, suppose that Q_1 starts with e_s and ends with an access edge of B that is incident to a pole u of B. Similarly to the proof of statement 1 in Lemma 4.1, we can show that each $Q_i \in \{Q_2, \ldots, Q_k\}$ must use exactly two access edges of B. Since B has at most four access edges, k must be either 1 or 2. If $k = 2$, the trail Q_2 or its reverse either starts with an access edge incident to u and ends with an access edge incident to the other pole v of B, or it starts and ends with the two access edges incident to v. Based on these observations, the ODTC $(Q_1, \ldots, Q_k)$ can only be of the following possible types:

- *Type* π_1. $k = 1$; Q_1 ends at a pole; Q_1^* ends at a pole.
- *Type* π_2. $k = 2$; Q_1 ends at a pole, and Q_2 starts at a pole and ends at the other pole; Q_1^* ends at a pole, and Q_2^* starts at a pole and ends at the other pole.
- *Type* π_3. $k = 2$; Q_1 ends at a pole, and Q_2 starts at a pole and ends at the other pole; Q_1^* ends at a pole x, and Q_2^* starts at the other pole y and ends at y.
- *Type* π_4. $k = 2$; Q_1 ends at a pole u, and Q_2 starts at the other pole v and ends at v; Q_1^* ends at a pole, and Q_2^* starts at a pole and ends at the other pole.

Note that, for each type π_i, the trail Q_1 is required to end at a pole and starts with the edge e_s. However, the edge e_s might be either incident to a pole or not incident to a pole.

Define $\Pi = \{\pi_1, \ldots, \pi_4\}$. We call the elements of Π *π-types*. For a TTG H, let $\mathcal{K}_\pi(H)$ denote the set of all $\pi_i \in \Pi$ such that a different embedding of H has a Type-π_i DTC. We call each member of $\mathcal{K}_\pi(H)$ a *π-type of H*. For each $\pi_i \in \Pi$, a *representative graph* $R(\pi_i)$ is shown in Figure 5.1. Note that for all $i \in \{1, \ldots, 4\}$, π_i is the unique π-type of $R(\pi_i)$.

Case 3. Both e_s and e_t are in B. Then, $k > 1$ or else B could not have been a proper block of G. Without loss of generality, suppose that P_1 starts with e_s and P_k ends with e_t. Each of P_1 and P_k uses at least one access edge of B. Similarly to the proof of statement 1 in Lemma 4.1, we can show that each $P_i \in \{P_2, \ldots, P_{k-1}\}$ must use exactly two access edges of B. Thus, k must be either 2 or 3. By considering which access edges are used by which P_i, it can be seen that $P \cap B$ can be only one of the following possible types:

- *Type* δ_1. $k = 2$; P_1 ends at a pole, and P_2 starts at the same pole; P_1^* ends at a pole, and P_2^* starts at the other pole.
- *Type* δ_2. $k = 2$; P_1 ends at a pole, and P_2 starts at the other pole; P_1^* ends at a pole, and P_2^* starts at the same pole.
- *Type* δ_3. $k = 2$; P_1 ends at a pole, and P_2 starts at the other pole; P_1^* ends at a pole, and P_2^* starts at the other pole.
- *Type* δ_4. $k = 3$; P_1 ends at a pole u, P_2 starts at the other pole v and ends at v, and P_3 starts at u; P_1^* ends at a pole x, P_2^* starts at x and ends at the other pole y, and P_3^* starts at y.
- *Type* δ_5. $k = 3$; P_1 ends at a pole u, P_2 starts at u and ends at the other pole v, and P_3 starts at v; P_1^* ends at a pole x, P_2^* starts at the other pole y and ends at y, and P_3^* starts at x.
- *Type* δ_6. $k = 3$; P_1 ends at a pole u, P_2 starts at a pole and ends at the other

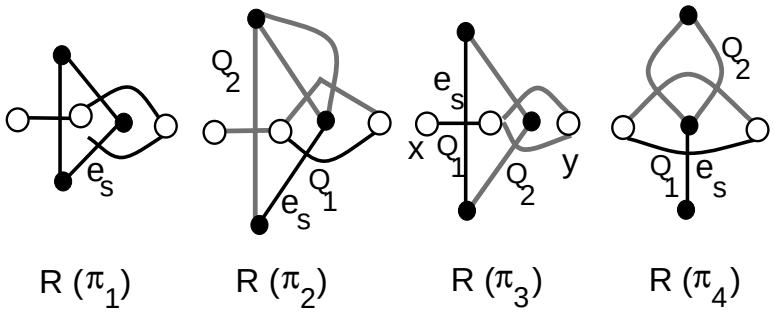


FIG. 5.1. *Representative graphs for π-types.*

pole, and P_3 starts at the pole $v \neq u$; P_1^* ends at a pole x, P_2^* starts at a pole and ends at the other pole, and P_3^* starts at the pole $y \neq x$.

Note that for each type δ_i, the trail P_1 is required to end at a pole and starts with the edge e_s. However, e_s might be either incident to a pole or not incident to a pole. The trail P_2 in the types $\delta_1, \delta_2, \delta_3$ (the trail P_3 in the types $\delta_4, \delta_5, \delta_6$, respectively) is required to start at a pole and ends with the edge e_t. However, e_t might be either incident to a pole or not incident to a pole.

Define $\Delta = \{\delta_1, \ldots, \delta_6\}$. We note that when $B = G$, $P \cap B$ must be of the following type:

- *Type* δ_0. $k = 1$.

We call the elements of $\Delta \cup \{\delta_0\}$ *δ-types*. For a TTG H, let $\mathcal{K}_\delta(H)$ denote the set of all $\delta_i \in \Delta \cup \{\delta_0\}$ such that a different embedding of H has a Type-δ_i DTC. We call each member of $\mathcal{K}_\delta(H)$ a *δ-type of H*. For each $\delta_i \in \Delta$, a *representative graph* $R(\delta_i)$ is shown in Figure 5.2. Note that for all $i \in \{1, \ldots, 6\}$, δ_i is the unique δ-type of $R(\delta_i)$.

Define $\Gamma = \Sigma \cup \Pi \cup \Delta$. Let H be a representative graph in Figures 5.1 and 5.2. If $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are two DTCs of H such that $\mathcal{Q}_2$ can be obtained from $\mathcal{Q}_1$ by reversing the direction of exactly one double trail on which neither e_s nor e_t appears, then we view $\mathcal{Q}_1$ and $\mathcal{Q}_2$ as the same DTC. A DTC $\mathcal{Q}$ of H is *valid* if $\mathcal{Q}$ is of a σ-, π-, or δ-type. From the figures, we can see that H has a unique valid DTC. Thus, if H is a graph in Figure 5.1, then we think that e_s and its dual edge have been oriented by the valid DTC of H. Similarly, if H is a graph in Figure 5.2, then we think that e_s and e_t and their dual edges have been oriented by the valid DTC of H.

5.2. Series composition of DTCs. This subsection shows how to compute the π- or δ-types of a series composition of two or more TTGs $H_1, \ldots, H_k$, when the σ-, π-, or δ-types of H_1 through H_k are given. This is needed by our algorithm to process the S-nodes of T. Let $\Gamma_1 = \Sigma_1 \cup \Pi_1 \cup \Delta_1$ and $\Gamma_2 = \Sigma_2 \cup \Pi_2 \cup \Delta_2$ be two subsets of Γ, where for $i = 1, 2$, $\Sigma_i \subseteq \Sigma$, $\Pi_i \subseteq \Pi$, and $\Delta_i \subseteq \Delta$. Define

$$\begin{aligned}\Gamma_1 \star \Gamma_2 = {} & (\cup_{\sigma_i \in \Sigma_1, \sigma_j \in \Sigma_2} \mathcal{K}_\sigma(R(\sigma_i) \star R(\sigma_j))) \cup (\cup_{\sigma_i \in \Sigma_1, \pi_j \in \Pi_2} \mathcal{K}_\pi(R(\sigma_i) \star R(\pi_j))) \\ & \cup (\cup_{\pi_i \in \Pi_1, \sigma_j \in \Sigma_2} \mathcal{K}_\pi(R(\pi_i) \star R(\sigma_j))) \cup (\cup_{\sigma_i \in \Sigma_1, \delta_j \in \Delta_2} \mathcal{K}_\delta(R(\sigma_i) \star R(\delta_j))) \\ & \cup (\cup_{\delta_i \in \Delta_1, \sigma_j \in \Sigma_2} \mathcal{K}_\delta(R(\delta_i) \star R(\sigma_j))) \cup (\cup_{\pi_i \in \Pi_1, \pi_j \in \Pi_2} \mathcal{K}_\delta(R(\pi_i) \star R(\pi_j))).\end{aligned}$$

If $\Gamma_1 = \emptyset$ or $\Gamma_2 = \emptyset$, then $\Gamma_1 \star \Gamma_2 = \emptyset$.

Recall that, by the definition, $\mathcal{K}_\sigma(H)$ is the set of $\sigma_i \in \Sigma$ such that a different embedding of H has a type-σ_i ODTC. In the case $H = H_1 \star H_2$, if we switch H_1 and

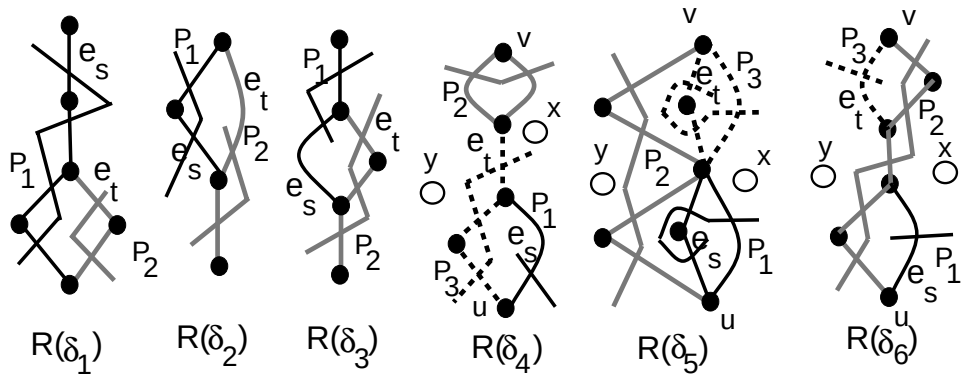


FIG. 5.2. *Representative graphs for δ-types.*

H_2, the graph $H_2 \star H_1$ is a different embedding of H, and hence the σ-types of $H_1 \star H_2$ and $H_2 \star H_1$ both belong to $\mathcal{K}_\sigma(H)$. This remark is also true for $\mathcal{K}_\pi(H)$ and $\mathcal{K}_\delta(H)$. Thus, the operator $\star$ defined on Γ_1 and Γ_2 is trivially commutative by its definition. For later reference, we state this in the following lemma.

LEMMA 5.1. *If* Γ_1 *and* Γ_2 *are two subsets of* Γ*, then* $\Gamma_1 \star \Gamma_2 = \Gamma_2 \star \Gamma_1$.

For each $\sigma_i \in \Sigma$ and each $\delta_j \in \Delta$, the (i,j)-entry of Table 5.1 contains all $\delta_k \in \{\sigma_i\} \star \{\delta_j\}$. For each $\pi_i \in \Pi$ and each $\sigma_j \in \Sigma$, the (i,j)-entry of Table 5.2 contains all $\pi_k \in \{\pi_i\} \star \{\sigma_j\}$. For each $\pi_i \in \Pi$ and each $\pi_j \in \Pi$, the (i,j)-entry of Table 5.3 contains all $\delta_k \in \{\pi_i\} \star \{\pi_j\}$. These tables can be verified by inspecting the representative graphs in Figures 4.2, 5.1, and 5.2. As examples, we verify one entry from each of Tables 5.1, 5.2, and 5.3. They are shown in Figure 5.3. In each case, two representative graphs are composed in series by identifying the common pole v. The type of the resulting graph is shown in the figure.

Suppose that $H = H_1 \star H_2 \star \cdots \star H_k$ and that $\mathcal{K}_\sigma(H_i)$, $\mathcal{K}_\pi(H_i)$, and $\mathcal{K}_\delta(H_i)$ have already been computed for all $i \in \{1, \ldots, k\}$. Our goal is to calculate $\mathcal{K}_\sigma(H)$, $\mathcal{K}_\pi(H)$, and $\mathcal{K}_\delta(H)$ for the series composition H. $\mathcal{K}_\sigma(H)$ can be computed as in section 4.2. By definition, $\mathcal{K}_\pi(H)$ contains all the π-types of all different embeddings of H. Essentially, this means that the computation of $\mathcal{K}_\pi(H)$ requires the evaluation of all expressions consisting of one operand from some $\mathcal{K}_\pi(H_i)$ and one operand from each of $\mathcal{K}_\sigma(H_1), \ldots, \mathcal{K}_\sigma(H_{i-1}), \mathcal{K}_\sigma(H_{i+1}), \ldots, \mathcal{K}_\sigma(H_k)$ connected by the operator $\star$. Although the operator $\star$ is commutative, it is not associative. This makes the calculation of $\mathcal{K}_\pi(H)$ very complex. The calculation of $\mathcal{K}_\delta(H)$ is even more complex. The rest of this subsection discusses how to perform these calculations.

A $\star$-*expression* E, its *value*, and its *operand multiset* $\mathrm{Op}(E)$ are defined inductively as follows. Each $\gamma \in \Gamma$ is a $\star$-expression, its value is $\{\gamma\}$, and its operand multiset is $\langle\gamma\rangle$. Each (possibly empty) $\Gamma_1 \subseteq \Gamma$ is a $\star$-expression, its value is Γ_1, and its operand multiset is $\langle\Gamma_1\rangle$. If E_1 and E_2 are $\star$-expressions, then $(E_1 \star E_2)$ is a $\star$-expression, its operand multiset is $\mathrm{Op}(E_1) \cup \mathrm{Op}(E_2)$, and its value is $(\Gamma_1 \star \Gamma_2)$, where Γ_1 and Γ_2 are the value of E_1 and E_2, respectively.

We identify a $\star$-expression with its value. For a list $(X_1, \ldots, X_k)$ of elements or subsets of Γ, let $X_1 \star X_2 \star \cdots \star X_k$ denote the $\star$-expression $(\cdots((X_1 \star X_2) \star X_3) \star \cdots X_k)$. For a subset Γ_1 of Γ and a list $L = (\sigma_{i_1}, \ldots, \sigma_{i_k})$ of σ-types, let $\Gamma_1 \star [L]_\star$ denote the $\star$-expression $\Gamma_1 \star \sigma_{i_1} \star \cdots \star \sigma_{i_k}$ if $k \geq 1$; and denote Γ_1 otherwise. For a $\sigma_i \in \Sigma$ and a nonnegative integer k, let σ_i^k denote the list of k σ_i's.

TABLE 5.1
$\{\sigma_i\} \star \{\delta_j\}$.

	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6
σ_1		δ_3	δ_2			
σ_2	δ_0, δ_4				δ_1	δ_1
σ_3		δ_2	δ_3			
σ_4					δ_4	δ_4
σ_5	δ_1			δ_4	δ_5	δ_6

TABLE 5.2
$\{\pi_i\} \star \{\sigma_j\}$.

	σ_1	σ_2	σ_3	σ_4	σ_5
π_1	π_1	π_4	π_1		
π_2	π_3	π_1	π_2	π_4	π_2
π_3	π_2	π_1	π_3	π_4	π_3
π_4	π_4		π_4		π_4

TABLE 5.3
$\{\pi_i\} \star \{\pi_j\}$.

	π_1	π_2	π_3	π_4
π_1	δ_0, δ_2, δ_3	δ_1	δ_1	
π_2	δ_1	δ_3, δ_5	δ_2, δ_6	δ_2, δ_3, δ_4
π_3	δ_1	δ_2, δ_6	δ_3, δ_5	δ_2, δ_3, δ_4
π_4		δ_2, δ_3, δ_4	δ_2, δ_3, δ_4	

The *parse tree* T_E of a $\star$-expression E is defined inductively as follows. If $|\mathrm{Op}(E)| = 1$, then T_E consists of a single node labeled with the unique element of $\mathrm{Op}(E)$. If $E = (E_1 \star E_2)$, then T_E is obtained from the parse tree T_{E_1} of E_1 and the parse tree T_{E_2} of E_2 by introducing a new node r and letting the roots of T_{E_1} and T_{E_2} be the children of r.

For convenience, each node v in a parse tree T_E is viewed as an ancestor of v itself in T_E.

LEMMA 5.2. *Suppose $\Delta_1 \subseteq \Delta$. Let $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ be a list of σ-types. Then, the following two statements hold:*

1. *For every $\star$-expression E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$, if σ_2 is not in L, then $E \subseteq \Delta_1 \star [L]_\star$; otherwise, $E \subseteq \Delta_1 \star [L - \{\sigma_5\}]_\star$.*
2. *Let U be the union of the values of all $\star$-expressions E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$. If σ_2 is not in L, then $U = \Delta_1 \star [L]_\star$; otherwise, $U = \Delta_1 \star [L - \{\sigma_5\}]_\star$.*

Proof. We prove the two statements separately as follows.

Statement 1. First, we claim that if σ_i and σ_j are (possibly the same) elements of Σ with $\{\sigma_i, \sigma_j\} \neq \{\sigma_2, \sigma_5\}$, then $\Delta_1 \star (\sigma_i \star \sigma_j) \subseteq \Delta_1 \star \sigma_i \star \sigma_j = \Delta_1 \star \sigma_j \star \sigma_i$. One may verify this claim using Tables 4.1 and 5.1, under the assumption that $|\Delta_1| = 1$. This claim then holds for every $\Delta_1 \subseteq \Delta$.

Second, we claim that $(\Delta_1 \star \sigma_5 \star \sigma_2) \cup (\Delta_1 \star \sigma_2 \star \sigma_5) \subseteq \Delta_1 \star (\sigma_5 \star \sigma_2) = \Delta_1 \star \sigma_2$. One may verify this claim using Tables 4.1 and 5.1, under the assumption that $|\Delta_1| = 1$. This claim then holds for every $\Delta_1 \subseteq \Delta$.

Third, we claim that for every nonempty list L of σ-types and every permutation L' of L, if σ_2 is not in L, then $\Delta_1 \star [L']_\star \subseteq \Delta_1 \star [L]_\star$; otherwise, $\Delta_1 \star [L']_\star \subseteq \Delta_1 \star [L - \{\sigma_5\}]_\star$. This is shown by induction on the number q of elements of L as follows. The claim trivially holds when $q = 1$. Let $q \geq 2$ and $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$

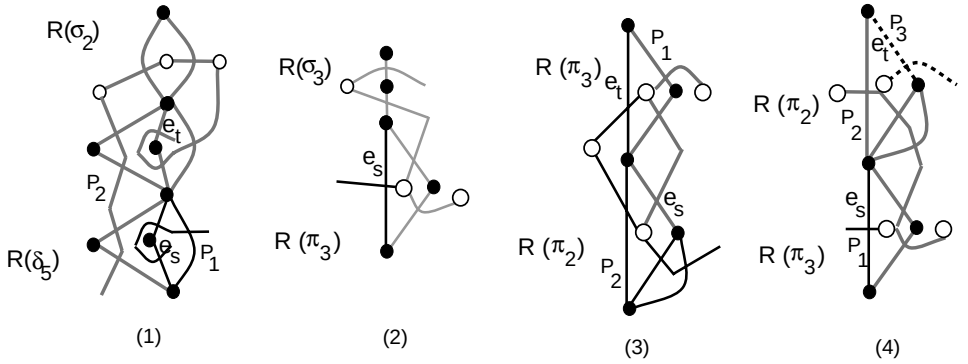


FIG. 5.3. (1) $\sigma_2 \star \delta_5 = \delta_1$; (2) $\pi_3 \star \sigma_3 = \pi_3$; (3) $\pi_2 \star \pi_3 = \delta_2$; (4) $\pi_2 \star \pi_3 = \delta_6$.

be a list of σ-types. Let $L' = (\sigma_{j_1}, \ldots, \sigma_{j_q})$ be a permutation of L. If σ_5 or σ_2 does not occur in L, then, by the first claim, $\Delta_1 \star [L']_\star \subseteq \Delta_1 \star [L]_\star$. So, suppose that both σ_5 and σ_2 occur in L. Consider a sublist $(\sigma_{j_k}, \sigma_{j_{k+1}}, \ldots, \sigma_{j_\ell})$ of L' such that $\{\sigma_{j_k}, \sigma_{j_\ell}\} = \{\sigma_5, \sigma_2\}$ and $\{\sigma_5, \sigma_2\} \cap \{\sigma_{j_{k+1}}, \ldots, \sigma_{j_{\ell-1}}\} = \emptyset$. By the first claim, $\Delta_1 \star [L']_\star = \Delta_1 \star \sigma_{j_1} \star \cdots \star \sigma_{j_k} \star \sigma_{j_\ell} \star \sigma_{j_{k+1}} \star \cdots \star \sigma_{j_{\ell-1}} \star \sigma_{j_{\ell+1}} \star \cdots \star \sigma_{j_q}$. This together with the second claim implies that $\Delta_1 \star [L']_\star \subseteq \Delta_1 \star \sigma_{j_1} \star \cdots \star \sigma_{j_{k-1}} \star \sigma_2 \star \sigma_{j_{k+1}} \star \cdots \star \sigma_{j_{\ell-1}} \star \sigma_{j_{\ell+1}} \star \cdots \star \sigma_{j_q}$. Now, the third claim follows from the inductive hypothesis.

Let L and E be as described in the lemma. Let T_E be the parse tree of E, and let v be the leaf node labeled with Δ_1. Let $v, u_1, \ldots, u_m$ be the path from v to the root of T_E. For each $i \in \{1, \ldots, m\}$, let E_i be the $\star$-expression whose parse tree is the subtree of T_E rooted at the child of u_i that is not an ancestor of v. Then, $E = \Delta_1 \star E_1 \star \cdots \star E_m$. Define $E'_1 = \sigma_{j_1} \star \cdots \star \sigma_{j_a}$, where $\langle \sigma_{j_1}, \ldots, \sigma_{j_a} \rangle$ equals $\mathrm{Op}(E_1)$ if σ_2 is not in $\mathrm{Op}(E_1)$, while it equals $\mathrm{Op}(E_1) - \{\sigma_5\}$ otherwise. Define $E'_2 = \sigma_{k_1} \star \cdots \star \sigma_{k_b}$, $\ldots$, $E'_m = \sigma_{\ell_1} \star \cdots \star \sigma_{\ell_c}$ similarly. Since operator $\star$ is commutative and associative over Σ (by Lemma 4.2) and $\sigma_5 \star \sigma_2 = \sigma_2$, we have $E_i = E'_i$ for all $i \in \{1, \ldots, m\}$. Thus, $E = \Delta_1 \star E'_1 \star \cdots \star E'_m$. By Table 4.1, if two elements σ_g and σ_h of Σ satisfy $\sigma_g \star \sigma_h = \sigma_2$, then $\sigma_2 \in \{\sigma_g, \sigma_h\}$; similarly, if two elements σ_g and σ_h of Σ satisfy $\sigma_g \star \sigma_h = \sigma_5$, then $\sigma_g = \sigma_h = \sigma_5$. By this fact, for every $i \in \{1, \ldots, m\}$, if σ_2 (respectively, σ_5) is not in $\mathrm{Op}(E'_i)$, then σ_2 (respectively, σ_5) does not result when we evaluate E'_i from left to right. Thus, by the first claim, $E \subseteq \Delta_1 \star \sigma_{j_1} \star \cdots \star \sigma_{j_a} \star \sigma_{k_1} \star \cdots \star \sigma_{k_b} \star \cdots \star \sigma_{\ell_1} \star \cdots \star \sigma_{\ell_c}$. Now, by the third claim, statement 1 holds.

Statement 2. First suppose that σ_2 is not in L. Then, by statement 1, $U \subseteq \Delta_1 \star [L]_\star$. On the other hand, $\Delta_1 \star [L]_\star \subseteq U$ because $\Delta_1 \star [L]_\star$ is a $\star$-expression whose operand multiset is $\langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$. So, $U = \Delta_1 \star [L]_\star$.

Next, suppose that σ_2 is in L. Then, by statement 1, $U \subseteq \Delta_1 \star [L - \{\sigma_5\}]_\star$. On the other hand, we can modify the $\star$-expression $\Delta_1 \star [L - \{\sigma_5\}]_\star$ by replacing one appearance of σ_2 with $(\sigma_2 \star \sigma_5 \star \cdots \star \sigma_5)$ in which σ_5 appears as many times as in L. The resulting $\star$-expression has the same value as $\Delta_1 \star [L - \{\sigma_5\}]_\star$, since $\sigma_2 \star \sigma_5 = \sigma_2$, and is a $\star$-expression whose operand multiset is $\langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$. So, $\Delta_1 \star [L - \{\sigma_5\}]_\star \subseteq U$. This completes the proof of statement 2. □

Lemma 5.2 still holds after replacing each appearance of "Δ" in it with "Π," as stated in the following.

LEMMA 5.3. *Suppose* $\Pi_1 \subseteq \Pi$. *Let* $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ *be a list of* σ*-types. Then, the following two statements hold:*

1. *For every* $\star$*-expression* E *with* $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1 \rangle$, *if* σ_2 *is not in* L,

then $E \subseteq \Pi_1 \star [L]_\star$*; otherwise,* $E \subseteq \Pi_1 \star [L - \{\sigma_5\}]_\star$.

2. *Let* U *be the union of the values of all* $\star$*-expressions* E *with* $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1 \rangle$*. If* σ_2 *is not in* L*, then* $U = \Pi_1 \star [L]_\star$*; otherwise,* $U = \Pi_1 \star [L - \{\sigma_5\}]_\star$.

Proof. It suffices to modify the proof of Lemma 5.2 by replacing each appearance of "Δ" with "Π" and "Table 5.1" with "Table 5.2." □

LEMMA 5.4. *Suppose that* $\Pi_1 \subseteq \Pi$ *and* $\Pi_2 \subseteq \Pi$*. Then, for every* $\sigma \in \{\sigma_1, \sigma_3, \sigma_4\}$, $\Pi_1 \star \Pi_2 \star \sigma = \Pi_2 \star \Pi_1 \star \sigma \subseteq \Pi_1 \star \sigma \star \Pi_2 = \Pi_1 \star (\sigma \star \Pi_2) = \Pi_1 \star (\Pi_2 \star \sigma)$.

Proof. One may verify the lemma using Tables 5.1, 5.2, and 5.3, under the assumption that $|\Pi_1| = |\Pi_2| = 1$. This then implies the lemma for every $\Pi_1 \subseteq \Pi$ and every $\Pi_2 \subseteq \Pi$. □

LEMMA 5.5. *Suppose that* $\Pi_1 \subseteq \Pi$ *and* $\Pi_2 \subseteq \Pi$*. Let* $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ *be a list of* σ*-types. Let* q_2 *and* q_5 *be the number of appearances of* σ_2 *and* σ_5 *in* L*, respectively. Let* $L' = L - \{\sigma_2, \sigma_5\}$*. Then, the following two statements hold:*

1. *For every* $\star$*-expression* E *with* $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$*, if* $q_2 \geq 7$*, then* $E = \emptyset$*; otherwise,* $E \subseteq \cup_{(a,b,c,d,h,k)} (\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$*, where* (a, b, c, d, h, k) *ranges over all sextuples of nonnegative integers such that* (1) $a + c + h = q_2$*;* (2) $b \leq 1$*,* $d \leq 1$*,* $k \leq 1$*, and* $b + d + k \leq q_5$*;* (3) *if* $a \geq 1$*, then* $b = 0$*;* (4) *if* $c \geq 1$*, then* $d = 0$*;* (5) *if* $h \geq 1$*, then* $k = 0$*; and* (6) *if* $q_2 = 0$*, then* $b + d + k \geq \min\{1, q_5\}$.
2. *Let* U *be the union of the values of all* $\star$*-expressions* E *with* $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$*. If* $q_2 \geq 7$*, then* $U = \emptyset$*; otherwise,* $U = \cup_{(a,b,c,d,h,k)} (\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$*, where* (a, b, c, d, h, k) *ranges over all sextuples of nonnegative integers that satisfy the conditions* (1) *through* (6) *in statement* 1.

Proof. We prove the two statements separately as follows.

Statement 1. Let T_E be the parse tree of E, v_1 be the leaf node of T_E labeled with Π_1, v_2 be the leaf node of T_E labeled with Π_2, and w be the lowest common ancestor of v_1 and v_2 in T_E. Let $v_1, u_1, \ldots, u_\ell, w$ be the path from v_1 to w in T_E. Similarly, let $v_2, u_{\ell+1}, \ldots, u_p, w$ be the path from v_2 to w in T_E. Let $w, u_{p+1}, \ldots, u_r$ be the path from w to the root of T_E. For each $i \in \{1, \ldots, r\}$, let E_i be the $\star$-expression whose parse tree is the subtree of T_E rooted at the child of u_i that is neither an ancestor of v_1 nor an ancestor of v_2. Then, $E = (\Pi_1 \star E_1 \star \cdots \star E_\ell) \star (\Pi_2 \star E_{\ell+1} \star \cdots \star E_p) \star E_{p+1} \star \cdots \star E_r$. Suppose that $\cup_{1 \leq x \leq \ell} \mathrm{Op}(E_x) = \langle \sigma_{j_1}, \ldots, \sigma_{j_f} \rangle$, $\cup_{\ell+1 \leq x \leq p} \mathrm{Op}(E_x) = \langle \sigma_{j_{f+1}}, \ldots, \sigma_{j_g} \rangle$, and $\cup_{p+1 \leq x \leq r} \mathrm{Op}(E_x) = \langle \sigma_{j_{g+1}}, \ldots, \sigma_{j_q} \rangle$. Let L_1 be the list $(\sigma_{j_1}, \ldots, \sigma_{j_f})$, L_2 be the list $(\sigma_{j_{f+1}}, \ldots, \sigma_{j_g})$, and L_3 be the list $(\sigma_{j_{g+1}}, \ldots, \sigma_{j_q})$. If σ_2 is not in L_1, then let $L_1' = L_1$; otherwise, let $L_1' = L_1 - \{\sigma_5\}$. Similarly, L_2' and L_3' are defined from L_2 and L_3, respectively. Then, by statement 1 in Lemma 5.2 and statement 1 in Lemma 5.3, $E \subseteq (\Pi_1 \star [L_1']_\star) \star (\Pi_2 \star [L_2']_\star) \star [L_3']_\star = (\Pi_1 \star [L_1']_\star) \star (\Pi_2 \star [L_2']_\star) \star [L_3' - \{\sigma_2, \sigma_5\}]_\star \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$. In turn, by Lemma 5.4, $E \subseteq (\Pi_1 \star [L_1']_\star \star [L_3' - \{\sigma_2, \sigma_5\}]_\star) \star (\Pi_2 \star [L_2']_\star) \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$. So, by statement 1 in Lemma 5.3, $E \subseteq (\Pi_1 \star [L_1']_\star \star [L_3' - \{\sigma_2, \sigma_5\}]_\star) \star (\Pi_2 \star [L_2' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star \star [L_2' - \{\sigma_2, \sigma_5\}]_\star) \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$. In turn, by Lemma 5.4, $E \subseteq (\Pi_1 \star [L_1']_\star \star [L_3' - \{\sigma_2, \sigma_5\}]_\star \star [L_2' - \{\sigma_2, \sigma_5\}]_\star) \star (\Pi_2 \star [L_2' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star) \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$. Moreover, by statement 1 in Lemma 5.3, $E \subseteq (\Pi_1 \star [L_1' - \{\sigma_2, \sigma_5\}]_\star \star [L_3' - \{\sigma_2, \sigma_5\}]_\star \star [L_2' - \{\sigma_2, \sigma_5\}]_\star \star [L_1' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star) \star (\Pi_2 \star [L_2' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star) \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$. Note that $\mathrm{Op}([L']_\star) = \mathrm{Op}([L_1' - \{\sigma_2, \sigma_5\}]_\star) \cup \mathrm{Op}([L_3' - \{\sigma_2, \sigma_5\}]_\star) \cup \mathrm{Op}([L_2' - \{\sigma_2, \sigma_5\}]_\star)$. Thus, by statement 1 in Lemma 5.3, $E \subseteq (\Pi_1 \star [L']_\star \star [L_1' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star) \star (\Pi_2 \star [L_2' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star) \star [L_3' - \{\sigma_1, \sigma_3, \sigma_4\}]_\star$.

Let a (respectively, c and h) be the number of appearances of σ_2 in L_1' (respectively, in L_2' and L_3'). Let b (respectively, d and k) be the smaller of 1 and the number

of appearances of σ_5 in L'_1 (respectively, L'_2 and L'_3). By the definition of L'_1, L'_2, L'_3, the sextuple (a, b, c, d, h, k) satisfies the conditions (1) through (6) in the lemma.

Note that for every $\pi \in \Pi$ and every $\delta \in \Delta$, $\pi \star \sigma_2 \star \sigma_2 \star \sigma_2 = \emptyset$, $\delta \star \sigma_2 \star \sigma_2 \star \sigma_2 = \emptyset$. If $q_2 \geq 7$, then at least one of a, c, h is ≥ 3 and hence $E = \emptyset$. Otherwise, since $\pi \star \sigma_5 \star \sigma_5 = \pi \star \sigma_5$ and $\delta \star \sigma_5 \star \sigma_5 = \delta \star \sigma_5$ for every $\pi \in \Pi$ and every $\delta \in \Delta$, $E \subseteq (\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$. This completes the proof of statement 1.

Statement 2. If $q_2 \geq 7$, then $U = \emptyset$ by statement 1. Next, suppose that $q_2 \leq 6$. Then, by statement 1, $U \subseteq \cup_{(a,b,c,d,h,k)}(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$, where (a, b, c, d, h, k) ranges over all sextuples of nonnegative integers that satisfy the conditions (1) through (6) in statement 1. On the other hand, if $q_2 = q_5 = 0$, then for each sextuple (a, b, c, d, h, k) satisfying the conditions (1) through (6) in statement 1, $(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$ is a $\star$-expression whose operand multiset is $\langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$; and so $(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star \subseteq U$. If $q_2 \geq 1$ or $q_5 \geq 1$, then for each sextuple (a, b, c, d, h, k) satisfying the conditions (1) through (6) in statement 1, we can modify the $\star$-expression $(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$ either (i) by replacing one appearance of σ_2 with $(\sigma_2 \star \sigma_5 \star \cdots \star \sigma_5)$ in which σ_5 appears $q_5 - (b + d + k)$ times or (ii) by replacing one appearance of σ_5 with $(\sigma_5 \star \cdots \star \sigma_5)$ in which σ_5 appears $q_5 - (b + d + k) + 1$ times. The resulting $\star$-expression has the same value as $(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$, since $\sigma_2 \star \sigma_5 = \sigma_2$ and $\sigma_5 \star \sigma_5 = \sigma_5$, and is a $\star$-expression whose operand multiset is $\langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$. So, $(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star \subseteq U$. Consequently, $\cup_{(a,b,c,d,h,k)}(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star \subseteq U$, where (a, b, c, d, h, k) ranges over all sextuples of nonnegative integers that satisfy the conditions (1) through (6) in statement 1. This completes the proof of statement 2. □

5.3. Parallel composition of DTCs. Let $\Gamma_1 = \Sigma_1 \cup \Pi_1 \cup \Delta_1$ and $\Gamma_2 = \Sigma_2 \cup \Pi_2 \cup \Delta_2$ be two subsets of Γ, where for $i = 1, 2$, $\Sigma_i \subseteq \Sigma$, $\Pi_i \subseteq \Pi$, and $\Delta_i \subseteq \Delta$. Define

$$\begin{aligned}\Gamma_1 \diamond \Gamma_2 = {} & (\cup_{\sigma_i \in \Sigma_1, \sigma_j \in \Sigma_2} \mathcal{K}_\sigma(R(\sigma_i) \diamond R(\sigma_j))) \cup (\cup_{\sigma_i \in \Sigma_1, \pi_j \in \Pi_2} \mathcal{K}_\pi(R(\sigma_i) \diamond R(\pi_j))) \\ & \cup (\cup_{\pi_i \in \Pi_1, \sigma_j \in \Sigma_2} \mathcal{K}_\pi(R(\pi_i) \diamond R(\sigma_j))) \cup (\cup_{\sigma_i \in \Sigma_1, \delta_j \in \Delta_2} \mathcal{K}_\delta(R(\sigma_i) \diamond R(\delta_j))) \\ & \cup (\cup_{\delta_i \in \Delta_1, \sigma_j \in \Sigma_2} \mathcal{K}_\delta(R(\delta_i) \diamond R(\sigma_j))) \cup (\cup_{\pi_i \in \Pi_1, \pi_j \in \Pi_2} \mathcal{K}_\delta(R(\pi_i) \diamond R(\pi_j))).\end{aligned}$$

If $\Gamma_1 = \emptyset$ or $\Gamma_2 = \emptyset$, then $\Gamma_1 \diamond \Gamma_2 = \emptyset$.

As in the paragraph right before Lemma 5.1, we can argue that the operator $\diamond$ is commutative by its definition. For later reference, we state this in the following lemma.

LEMMA 5.6. If Γ_1 and Γ_2 are two subsets of Γ, then $\Gamma_1 \diamond \Gamma_2 = \Gamma_2 \diamond \Gamma_1$.

For each $\sigma_i \in \Sigma$ and each $\delta_j \in \Delta$, the (i, j)-entry of Table 5.4 contains all $\delta_k \in \{\sigma_i\} \diamond \{\delta_j\}$. For each $\pi_i \in \Pi$ and each $\sigma_j \in \Sigma$, the (i, j)-entry of Table 5.5 contains all $\pi_k \in \{\pi_i\} \diamond \{\sigma_j\}$. For each $\pi_i \in \Pi$ and each $\pi_j \in \Pi$, the (i, j)-entry of Table 5.6 contains all $\delta_k \in \{\pi_i\} \diamond \{\pi_j\}$. These tables can be verified by inspecting the graphs in Figures 4.2, 5.1, and 5.2. As examples, we verify one entry from each of the Tables 5.4, 5.5 and 5.6. They are shown in Figure 5.4. In each case, two representative graphs are composed in parallel by identifying their common poles. The type of the resulting graph is shown in the figure.

By modifying the four paragraphs right before Lemma 5.2 by replacing each letter "$\star$" with "$\diamond$," we can define a $\diamond$-*expression* E, its *value*, etc. Moreover, we can prove

TABLE 5.4
$\{\sigma_i\} \diamond \{\delta_j\}$.

	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6
σ_1	δ_3		δ_1			
σ_2	δ_1		δ_3			
σ_3		δ_0, δ_5		δ_2		δ_2
σ_4		δ_2		δ_4	δ_5	δ_6
σ_5				δ_5		δ_5

TABLE 5.5
$\{\pi_i\} \diamond \{\sigma_j\}$.

	σ_1	σ_2	σ_3	σ_4	σ_5
π_1	π_1	π_1	π_3		
π_2	π_4	π_2	π_1	π_2	π_3
π_3	π_3	π_3		π_3	
π_4	π_2	π_4	π_1	π_4	π_3

TABLE 5.6
$\{\pi_i\} \diamond \{\pi_j\}$.

	π_1	π_2	π_3	π_4
π_1	δ_0, δ_1, δ_3	δ_2		δ_2
π_2	δ_2	δ_3, δ_4	δ_1, δ_3, δ_5	δ_1, δ_6
π_3		δ_1, δ_3, δ_5		δ_1, δ_3, δ_5
π_4	δ_2	δ_1, δ_6	δ_1, δ_3, δ_5	δ_3, δ_4

the following four lemmas, just via modifying the proofs of Lemmas 5.2 through 5.5 by replacing each appearance of "σ_2," "σ_3," "σ_4," "σ_5," "$\star$," "q_2," "q_5," "Table 5.1," "Table 5.2," and "Table 5.3" in them with "σ_3," "σ_2," "σ_5," "σ_4," "$\diamond$," "q_3," "q_4," "Table 5.4," "Table 5.5," and "Table 5.6," respectively. We state these corresponding lemmas as follows.

LEMMA 5.7. *Suppose $\Delta_1 \subseteq \Delta$. Let $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ be a list of σ-types. Then, the following two statements hold:*

1. *For every $\diamond$-expression E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$, if σ_3 is not in L, then $E \subseteq \Delta_1 \diamond [L]_\diamond$; otherwise, $E \subseteq \Delta_1 \diamond [L - \{\sigma_4\}]_\diamond$.*
2. *Let U be the union of the values of all $\diamond$-expressions E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Delta_1 \rangle$. If σ_3 is not in L, then $U = \Delta_1 \diamond [L]_\diamond$; otherwise, $U = \Delta_1 \diamond [L - \{\sigma_4\}]_\diamond$.*

LEMMA 5.8. *Suppose $\Pi_1 \subseteq \Pi$. Let $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ be a list of σ-types. Then, the following two statements hold:*

1. *For every $\diamond$-expression E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1 \rangle$, if σ_3 is not in L, then $E \subseteq \Pi_1 \diamond [L]_\diamond$; otherwise, $E \subseteq \Pi_1 \diamond [L - \{\sigma_4\}]_\diamond$.*
2. *Let U be the union of the values of all $\diamond$-expressions E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1 \rangle$. If σ_3 is not in L, then $U = \Pi_1 \diamond [L]_\diamond$; otherwise, $U = \Pi_1 \diamond [L - \{\sigma_4\}]_\diamond$.*

LEMMA 5.9. *Suppose that $\Pi_1 \subseteq \Pi$ and $\Pi_2 \subseteq \Pi$. Then, for every $\sigma \in \{\sigma_1, \sigma_2, \sigma_5\}$, $\Pi_1 \diamond \Pi_2 \diamond \sigma = \Pi_2 \diamond \Pi_1 \diamond \sigma \subseteq \Pi_1 \diamond \sigma \diamond \Pi_2 = \Pi_1 \diamond (\sigma \diamond \Pi_2) = \Pi_1 \diamond (\Pi_2 \diamond \sigma)$.*

LEMMA 5.10. *Suppose that $\Pi_1 \subseteq \Pi$ and $\Pi_2 \subseteq \Pi$. Let $L = (\sigma_{i_1}, \ldots, \sigma_{i_q})$ be a list of σ-types. Let q_3 and q_4 be the number of appearances of σ_3 and σ_4 in L, respectively. Let $L' = L - \{\sigma_3, \sigma_4\}$. Then, the following two statements hold:*

1. *For every $\diamond$-expression E with $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$, if $q_3 \geq 7$, then $E = \emptyset$; otherwise, $E \subseteq \cup_{(a,b,c,d,h,k)} (\Pi_1 \diamond [L']_\diamond \diamond [\sigma_3^a]_\diamond \diamond [\sigma_4^b]_\diamond) \diamond (\Pi_2 \diamond [\sigma_3^c]_\diamond \diamond [\sigma_4^d]_\diamond) \diamond [\sigma_3^h]_\diamond \diamond [\sigma_4^k]_\diamond$, where (a, b, c, d, h, k) ranges over all sextuples of*

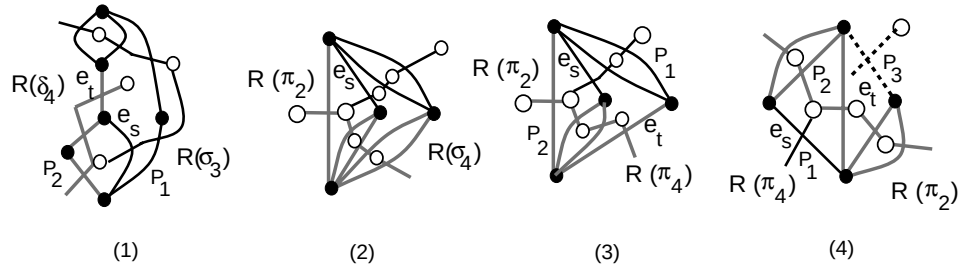


FIG. 5.4. (1) $\sigma_3 \diamond \delta_4 = \delta_2$; (2) $\pi_2 \diamond \sigma_4 = \pi_2$; (3) $\pi_2 \diamond \pi_4 = \delta_1$; (4) $\pi_2 \diamond \pi_4 = \delta_6$.

nonnegative integers such that (1) $a + c + h = q_3$; (2) $b \leq 1$, $d \leq 1$, $k \leq 1$, *and* $b + d + k \leq q_4$; (3) *if* $a \geq 1$, *then* $b = 0$; (4) *if* $c \geq 1$, *then* $d = 0$; (5) *if* $h \geq 1$, *then* $k = 0$; *and* (6) *if* $q_3 = 0$, *then* $b + d + k \geq \min\{1, q_4\}$.

2. *Let* U *be the union of the values of all* $\diamond$*-expressions* E *with* $\mathrm{Op}(E) = \langle \sigma_{i_1}, \ldots, \sigma_{i_q}, \Pi_1, \Pi_2 \rangle$. *If* $q_3 \geq 7$, *then* $U = \emptyset$; *otherwise,* $U = \cup_{(a,b,c,d,h,k)}(\Pi_1 \diamond [L']_\diamond \diamond [\sigma_3^a]_\diamond \diamond [\sigma_4^b]_\diamond) \diamond (\Pi_2 \diamond [\sigma_3^c]_\diamond \diamond [\sigma_4^d]_\diamond) \diamond [\sigma_3^h]_\diamond \diamond [\sigma_4^k]_\diamond$, *where* (a, b, c, d, h, k) *ranges over all sextuples of nonnegative integers that satisfy the conditions* (1) *through* (6) *in statement* 1.

5.4. An $O(|G|^3)$-time algorithm. Let e_s and e_t be a pair of distinct edges of G. We want to decide whether G has a valid embedding that has a DET starting with e_s and ending with e_t. To do this, we process the nodes of T in post-order. For each node μ of T such that neither e_s nor e_t is in $\mathrm{Pt}(\mu)$, we compute $\mathcal{K}_\sigma(\mathrm{Pt}(\mu))$ as in section 4.3; we may assume that $|\mathcal{K}_\sigma(\mathrm{Pt}(\mu))| = 1$ since otherwise no valid embedding of G has a DET starting with e_s and ending with e_t. For each Q-node μ such that $\mathrm{Pt}(\mu)$ consists of e_s only or e_t only, we set $\mathcal{K}_\pi(\mathrm{Pt}(\mu)) = \{\pi_1\}$.

Consider the processing of a non-Q-node μ such that e_s or e_t is in $\mathrm{Pt}(\mu)$. Let $\nu_1, \ldots, \nu_r$ be the children of μ in T. Let $B = \mathrm{Pt}(\mu)$. For each $i \in \{1, \ldots, r\}$, let $B_i = \mathrm{Pt}(\nu_i)$, and e_i be the virtual edge of ν_i in $\mathrm{Sk}(\mu)$.

- *Case* 1. B contains both e_s and e_t but no child of B does. Without loss of generality, we may assume that B_1 contains e_s and B_2 contains e_t. Then, none of $B_3, \ldots, B_r$ contains e_s or e_t. Let $\Pi_1 = \mathcal{K}_\pi(B_1)$ and $\Pi_2 = \mathcal{K}_\pi(B_2)$. For each $j \in \{3, \ldots, r\}$, let $\mathcal{K}_\sigma(B_j) = \{\sigma_{\ell_j}\}$. Let L be the list $(\sigma_{\ell_3}, \ldots, \sigma_{\ell_r})$.
  - *Case* 1.1. μ is an R-node. Then, $\mathrm{Sk}(\mu)$ has no different embedding other than itself. Based on statements 1 and 2 in Lemma 3.2, we compute $\mathcal{K}_\delta(B)$ as follows. We initialize $\mathcal{K}_\delta(B) = \emptyset$. After this, for each quadruple $D = (\pi_{i_1}, \pi_{i_2}, G_1, G_2)$ such that $\pi_{i_1} \in \mathcal{K}_\pi(B_1)$, $\pi_{i_2} \in \mathcal{K}_\pi(B_2)$, and G_1 (respectively, G_2) is obtained from $R(\pi_{i_1})$ (respectively, $R(\pi_{i_2})$) by horizontally and/or vertically flipping it, we first construct a TTG H_D from $\mathrm{Sk}(\mu)$ by replacing e_1 with G_1, e_2 with G_2, and e_j with $R(\sigma_{\ell_j})$ for all $j \in \{3, \ldots, r\}$. We then obtain the maximal double trail P_1 in H_D such that (1) e_s and its dual edge e_s^* are, respectively, the starting edge of P_1 and its dual trail P_1^* and (2) the direction of e_s in P_1 and that of e_s^* in P_1^* are the same as oriented by the unique valid DTC of G_1. Similarly, we obtain the maximal double trail P_2 in H_D such that (1) e_t and its dual edge e_t^* are, respectively, the starting edge of P_2 and its dual trail P_2^* and (2) the direction of e_t in P_2 and that of e_t^* in P_2^* are the same as oriented by the unique valid DTC of G_2. Let P_2' be the

reverse of P_2. Let M be the set of all edges of H_D traversed by neither P_1 nor P_2. Suppose $M = \emptyset$. If P_1 and P_2' are distinct, we check if P_1 and P_2' form a DTC of H_D of a δ-type. If P_1 and P_2' are the same, we check if P_1 forms a DTC of H_D of a δ-type. In both cases, if the condition is true, we add this δ-type to $\mathcal{K}_\delta(B)$.

Suppose that $M \neq \emptyset$ and P_1 and P_2 are edge-disjoint. We try to find an access edge $e \in M$ such that e is incident to a pole u of H_D and the dual edge e^* of e is incident to a pole x of H_D^*. If such e exists, then we obtain the maximal double trail P_3 in H_D such that (1) e and e^* are, respectively, the starting edge of P_3 and its dual trail P_3^* and (2) u and x are, respectively, the starting vertex of P_3 and P_3^*. Note that the other endpoint of e (respectively, e^*) is not a pole of H_D (respectively, H_D^*). If $\{P_1, P_2', P_3\}$ is a DTC of H_D of a δ-type, we add this δ-type to $\mathcal{K}_\delta(B)$. Similarly, if $\{P_1, P_2', P_3'\}$ is a DTC of H_D of a δ-type, we add this δ-type to $\mathcal{K}_\delta(B)$, where P_3' is the reverse of P_3.

- *Case* 1.2. μ is an S-node. Let q_2 and q_5 be the number of appearances of σ_2 and σ_5 in L, respectively. Let $L' = L - \{\sigma_2, \sigma_5\}$. By statement 2 in Lemma 5.5 and statements 1 and 2 in Lemma 3.2, if $q_2 \geq 7$, then $\mathcal{K}_\delta(B) = \emptyset$; otherwise, $\mathcal{K}_\delta(B) = \cup_{(a,b,c,d,h,k)}(\Pi_1 \star [L']_\star \star [\sigma_2^a]_\star \star [\sigma_5^b]_\star) \star (\Pi_2 \star [\sigma_2^c]_\star \star [\sigma_5^d]_\star) \star [\sigma_2^h]_\star \star [\sigma_5^k]_\star$, where (a, b, c, d, h, k) ranges over all sextuples of nonnegative integers that satisfy the conditions (1) through (6) in statement 1 in Lemma 5.5.
- *Case* 1.3. μ is a P-node. Let q_3 and q_4 be the number of appearances of σ_3 and σ_4 in L, respectively. Let $L' = L - \{\sigma_3, \sigma_4\}$. By statement 2 in Lemma 5.10 and statements 1 and 2 in Lemma 3.2, if $q_3 \geq 7$, then $\mathcal{K}_\delta(B) = \emptyset$; otherwise, $\mathcal{K}_\delta(B) = \cup_{(a,b,c,d,h,k)}(\Pi_1 \diamond [L']_\diamond \diamond [\sigma_3^a]_\diamond \diamond [\sigma_4^b]_\diamond) \diamond (\Pi_2 \diamond [\sigma_3^c]_\diamond \diamond [\sigma_4^d]_\diamond) \diamond [\sigma_3^h]_\diamond \diamond [\sigma_4^k]_\diamond$, where (a, b, c, d, h, k) ranges over all sextuples of nonnegative integers that satisfy the conditions (1) through (6) in statement 1 in Lemma 5.10.

• *Case* 2. B contains both e_s and e_t and so does one child block of B. Without loss of generality, we can assume that B_1 contains both e_s and e_t. Then, none of $B_2, \ldots, B_r$ contains e_s or e_t. Let $\Delta_1 = \mathcal{K}_\delta(B_1) - \{\delta_0\}$. For each $j \in \{2, \ldots, r\}$, let $\mathcal{K}_\sigma(B_j) = \{\sigma_{\ell_j}\}$. Let L be the list $(\sigma_{\ell_2}, \ldots, \sigma_{\ell_r})$.

- *Case* 2.1. μ is an R-node. Then, based on statements 1 and 3 in Lemma 3.2, we compute $\mathcal{K}_\delta(B)$ as follows. We initialize $\mathcal{K}_\delta(B) = \emptyset$. After this, for each pair $D = (\delta_i, G_1)$ such that $\delta_i \in \Delta_1$ and G_1 is obtained from $R(\delta_i)$ by horizontally and/or vertically flipping it, we first construct a TTG H_D from $\mathrm{Sk}(\mu)$ by replacing e_1 with G_1 and e_j with $R(\sigma_{\ell_j})$ for all $j \in \{2, \ldots, r\}$. Similarly to Case 1.1, we then check if H_D has a δ-type DTC; if it has, we add the δ-type of the DTC into $\mathcal{K}_\delta(B)$.
- *Case* 2.2. μ is an S-node. Then, by statement 2 in Lemma 5.2 and statements 1 and 3 in Lemma 3.2, if σ_2 is not in L, then $\mathcal{K}_\delta(B) = \Delta_1 \star [L]_\star$; otherwise, $\mathcal{K}_\delta(B) = \Delta_1 \star [L - \{\sigma_5\}]_\star$.
- *Case* 2.3. μ is a P-node. Then, by statement 2 in Lemma 5.7 and statements 1 and 3 in Lemma 3.2, if σ_3 is not in L, then $\mathcal{K}_\delta(B) = \Delta_1 \diamond [L]_\diamond$; otherwise, $\mathcal{K}_\delta(B) = \Delta_1 \diamond [L - \{\sigma_4\}]_\diamond$.

• *Case* 3. B contains one of e_s and e_t. We assume that B contains e_s; the other case is similar. Without loss of generality, we can assume that B_1 contains e_s. Then, none of $B_2, \ldots, B_r$ contains e_s or e_t. Let $\Pi_1 = \mathcal{K}_\pi(B_1)$. For each

$j \in \{2, \ldots, r\}$, let $\mathcal{K}_\sigma(B_j) = \{\sigma_{\ell_j}\}$. Let L be the list $(\sigma_{\ell_2}, \ldots, \sigma_{\ell_r})$.

- *Case* 3.1. μ is an R-node. Then, based on statements 1 and 2 in Lemma 3.2, we compute $\mathcal{K}_\pi(B)$ as follows. We initialize $\mathcal{K}_\pi(B) = \emptyset$. After this, for each pair $D = (\pi_i, G_1)$ such that $\pi_i \in \Pi_1$ and G_1 is obtained from $R(\pi_i)$ by horizontally and/or vertically flipping it, we first construct a TTG H_D from $\mathrm{Sk}(\mu)$ by replacing e_1 with G_1 and e_j with $R(\sigma_{\ell_j})$ for all $j \in \{2, \ldots, r\}$. Similarly to Case 1.1, we then check if H_D has a π-type DTC; if it has, we add the π-type of the DTC into $\mathcal{K}_\pi(B)$.
- *Case* 3.2. μ is an S-node. Then, by statement 2 in Lemma 5.3 and statements 1 and 2 in Lemma 3.2, if σ_2 is not in L, then $\mathcal{K}_\pi(B) = \Pi_1 \star [L]_\star$; otherwise, $\mathcal{K}_\pi(B) = \Pi_1 \star [L - \{\sigma_5\}]_\star$.
- *Case* 3.3. μ is a P-node. Then, by statement 2 in Lemma 5.8 and statements 1 and 2 in Lemma 3.2, if σ_3 is not in L, then $\mathcal{K}_\pi(B) = \Pi_1 \diamond [L]_\diamond$; otherwise, $\mathcal{K}_\pi(B) = \Pi_1 \diamond [L - \{\sigma_4\}]_\diamond$.

By the above discussions, it takes $O(|\mathrm{Sk}(\mu)|)$ time to process μ. Thus, by Lemma 2.1, the time needed to compute $\mathcal{K}_\delta(G)$ is $O(|G|)$. Note that G has a DET starting with e_s and ending with e_t if and only if $\delta_0 \in \mathcal{K}_\delta(G)$. Thus, by trying all pairs (e_s, e_t) of edges of G, we can decide in $O(|G|^3)$ time whether G has a DET.

THEOREM 5.11. *Given a TTG G, it takes $O(|G|^3)$ time to decide if G has a valid embedding.*

6. An $O(|G|)$-time algorithm. This section refines the algorithm in section 5.4. The basic idea is that we do not specify the starting and the ending edges in advance. Rather, we try to discover them when processing T in post-order. Throughout this section, we assume that G^* is not Eulerian; the case where G is not Eulerian is similar. Let T^* be the dual decomposition tree of T.

First, we need some definitions and lemmas. Recall that the degree parity of a vertex in G is 0 if its degree is even and is 1 otherwise. The *pole-parity multiset* M_G of G consists of the degree parities of the poles of G. The *pole-status* of G is the pair (M_G, M_{G^*}), where M_G and M_{G^*} are the pole-parity multiset of G and G^*, respectively.

G is *E-good* if it is Eulerian. G is *O-good* if its odd-degree vertices are just its poles. G is *good* if it is either E- or O-good. G is *bad* if it has at least one odd-degree nonpole.

A node μ of T is *E-good* (respectively, *O-good*, *good*, or *bad*) if all different embeddings of $\mathrm{Pt}(\mu)$ are E-good (respectively, O-good, good, or bad).

LEMMA 6.1. *Let μ be a node of T. Let $B = \mathrm{Pt}(\mu)$. Then, exactly one of the following* (1) *through* (3) *holds:* (1) *μ is E-good;* (2) *μ is O-good;* (3) *μ is bad. Moreover, for all descendants ν (including μ itself) of μ in T, it takes $O(|B|)$ total time to* (a) *decide whether ν is E-good, O-good, or bad, and* (b) *compute the pole-parity multiset of $\mathrm{Pt}(\nu)$ when ν is good.*

Proof. The proof is by induction. In case μ is a leaf node of T, B is a single edge; hence, μ is O-good.

Next, suppose that μ is a nonleaf node. Let $\nu_1, \ldots, \nu_q$ be the children of μ in T. For each $i \in \{1, \ldots, q\}$, let $B_i = \mathrm{Pt}(\nu_i)$ and e_i be the virtual edge of ν_i in $\mathrm{Sk}(\mu)$. If some child of μ is bad, then μ must be bad. So, assume that no child of μ is bad. By the inductive hypothesis, each child of μ is either E- or O-good. Let $\mathcal{B}$ be an (arbitrary) different embedding of B. For each $i \in \{1, \ldots, q\}$, let $\mathcal{B}_i$ be the child block of $\mathcal{B}$ that is a different embedding of B_i. For the purpose of deciding whether $\mathcal{B}$ is E-good, O-good, or bad, we may assume that $\mathcal{B}_i = B_i$ for all $i \in \{1, \ldots, q\}$ because

$\mathcal{B}_i$ is E-good (respectively, O-good) if and only if B_i is too.

Case 1. μ is a P-node. Then, $\mathcal{B}$ can be obtained from B by permuting the children of μ. Permuting the children of μ does not change the degree parity of any vertex in B. Thus, if an odd (respectively, even) number of children of μ is O-good, μ is O-good (respectively, E-good).

Case 2. μ is an S-node. Then, $\mathcal{B}$ can be obtained from B by permuting the children of μ. When every child of μ is E-good, permuting the children of μ does not change the degree parity of any vertex in B, and hence $\mathcal{B}$ is E-good and so is μ. Similarly, when every child of μ is O-good, μ is O-good. So, suppose that μ has both E-good children and O-good children. Then, some E-good $\mathcal{B}_i$ must be next to an O-good $\mathcal{B}_j$ in $\mathcal{B}$. The common pole of $\mathcal{B}_i$ and $\mathcal{B}_j$ in $\mathcal{B}$ has odd degree. Thus, $\mathcal{B}$ is bad and so is μ.

Case 3. μ is an R-node. Then, $\mathrm{Sk}(\mu)$ has no different embedding other than itself. $\mathcal{B}$ is obtained from $\mathrm{Sk}(\mu)$ by replacing e_i with $\mathcal{B}_i$ for all $i \in \{1, \ldots, q\}$. Note that the edges of $\mathrm{Sk}(\mu)$ are just $e_1, \ldots, e_q$. For all $i \in \{1, \ldots, q\}$, we mark e_i if ν_i is O-good. If some nonpole in $\mathrm{Sk}(\mu)$ is incident to an odd number of marked edges, then μ is bad. Otherwise, if a pole of $\mathrm{Sk}(\mu)$ is incident to an odd number of marked edges, then μ is O-good; otherwise, μ is E-good.

Next we describe the implementation of this procedure. Let T_μ be the subtree of T rooted at the node μ. We perform a post-order computation on T_μ. Each leaf node ν is an O-good node, and both poles of $\mathrm{Pt}(\nu)$ have odd parity. For each internal node ν in T_μ, whether ν is E-good, O-good, or bad, the pole-parity multiset of $\mathrm{Pt}(\nu)$ can be determined as in the three cases above, in at most $O(|\mathrm{Sk}(\nu))|$ time. Thus, the total time needed is $O(|B|)$ by Lemma 2.1. □

By Lemma 2.2, each node of T is also a node of the dual decomposition tree T^* of T. A node μ of T^* is *E-good* (respectively, *O-good*, *good*, or *bad*) if all different embeddings of the dual graph of $\mathrm{Pt}(\mu)$ are E-good (respectively, O-good, good, or bad). Lemma 6.1 still holds after changing T to T^*, $\mathrm{Pt}(\mu)$ to its dual graph, and $\mathrm{Pt}(\nu)$ to its dual graph.

Let μ be a node of T and $B = \mathrm{Pt}(\mu)$. μ is *doubly good* if for every different embedding $\mathcal{B}$ of B, neither $\mathcal{B}$ nor $\mathcal{B}^*$ has an odd-degree nonpole. μ is *doubly bad* if for every different embedding $\mathcal{B}$ of B, $\mathcal{B}$ or $\mathcal{B}^*$ has at least one odd-degree nonpole. By Lemma 6.1, μ is either doubly good or doubly bad. μ is *critical* if (1) all children of μ are doubly good and (2) μ is doubly bad or is the root of T.

LEMMA 6.2. *If G has a valid embedding, then either one or two nodes of T are critical.*

Proof. Note that each leaf node of T is doubly good. Thus, at least one node of T is critical. If the root of T is critical, then all nodes of T except the root must be doubly good and the root is the unique critical node. Otherwise, let $\mu_1, \ldots, \mu_k$ be the critical nodes in T and $B_i = \mathrm{Pt}(\mu_i)$. By the definition of critical nodes, none of $\mu_1, \ldots, \mu_k$ can be an ancestor of another. Thus, no two of $B_1, \ldots, B_k$ share an edge. On the other hand, for each $i \in \{1, \ldots, k\}$, either B_i or B_i^* has an odd-degree nonpole; any DET of G must start or end with an edge in B_i. Therefore, if $k \geq 3$, G cannot have a DET. □

LEMMA 6.3. *Let μ be a doubly bad node of T, and let $B = \mathrm{Pt}(\mu)$. Then, all different embeddings $\mathcal{B}$ of B such that $\mathcal{B}$ has a π-type DTC have the same pole-status p, which can be computed as follows:*

1. *If μ is an E-good node of T, then $p = (\langle 0, 0\rangle, \langle 0, 1\rangle)$.*
2. *If μ is an O-good node of T, then $p = (\langle 1, 1\rangle, \langle 0, 1\rangle)$.*

3. *If μ is an E-good node of T^*, then $p = (\langle 0,1\rangle, \langle 0,0\rangle)$.*
4. *If μ is an O-good node of T^*, then $p = (\langle 0,1\rangle, \langle 1,1\rangle)$.*
5. *If μ is a bad node of both T and T^*, then $p = (\langle 0,1\rangle, \langle 0,1\rangle)$.*

Proof. Suppose that a different embedding $\mathcal{B}$ of B has a π-type DTC. Then, both $\mathcal{B}$ and $\mathcal{B}^*$ have at most one odd-degree nonpole (see Figure 5.1). Indeed, at least one of $\mathcal{B}$ and $\mathcal{B}^*$ has exactly one odd-degree nonpole because μ is doubly bad. If μ is a bad node of T, then the pole-parity multiset of $\mathcal{B}$ is $\langle 0,1\rangle$. On the other hand, by Lemma 6.1, if μ is an E-good (respectively, O-good) node of T, then the pole-parity multiset of $\mathcal{B}$ is $\langle 0,0\rangle$ (respectively, $\langle 1,1\rangle$). Similarly, by checking whether μ is a good node of T^*, we can decide the pole-parity multiset of $\mathcal{B}^*$. □

Let $B = \mathrm{Pt}(\mu)$, where μ is a nonroot node of T. A π- or δ-type γ of B is *useful* if some valid embedding $\mathcal{G}$ of G has a DET P such that $P \cap \mathcal{B}$ is of Type γ, where $\mathcal{B}$ is the block of $\mathcal{G}$ that is a different embedding of B.

Even for a doubly good node μ, the block $B = \mathrm{Pt}(\mu)$ might contain the starting edge e_s and/or the ending edge e_t of a DET of a valid embedding of G (when e_s and/or e_t is incident to a pole of B). Thus, a σ-type of B might actually be a π- or δ-type. In the following, we investigate which σ-type can be mapped to which π- or δ-type. The mapping from σ-types to π-types is specified by the following function: $f : \{1,\ldots,5\} \to \{1,\ldots,4\}$, where $f(1) = f(2) = f(3) = 1$, $f(4) = 4$, and $f(5) = 3$.

LEMMA 6.4. *Let μ be a doubly good nonroot node of T. Let $B = \mathrm{Pt}(\mu)$. Suppose that π_j is a useful π-type of B. Then, $\mathcal{K}_\sigma(B) = \{\sigma_i\}$ for some $\sigma_i \in \Sigma$, and $\pi_j = \pi_{f(i)}$.*

Proof. Suppose that a DET P of a valid embedding $\mathcal{G}$ of G starts with an edge e_s of $\mathcal{B}$ but ends with an edge not in $\mathcal{B}$, where $\mathcal{B}$ is the block of $\mathcal{G}$ that is a different embedding of B. Let P^* be the dual trail of P. Since μ is doubly good, neither $\mathcal{B}$ nor $\mathcal{B}^*$ has an odd-degree nonpole. Moreover, since G^* is not Eulerian, P^* must start at an odd-degree vertex of $\mathcal{B}^*$ by Facts 1 and 2. Thus, P^* starts at a pole of $\mathcal{B}^*$. On the other hand, P starts at a pole of $\mathcal{B}$. This is because of the following: (1) If G is Eulerian, P starts and ends at the same vertex of $\mathcal{B}$. Since e_s is in $\mathcal{B}$ and the ending edge e_t is not in $\mathcal{B}$, their common end vertex must be a pole of $\mathcal{B}$. (2) If G is not Eulerian, P starts at an odd-degree vertex of $\mathcal{B}$ by Facts 1 and 2. By these facts, $P \cap \mathcal{B}$ is of a σ-type; this type is unique by Lemma 4.4. Let σ_i be this type.

To determine the π-type of $P \cap \mathcal{B}$, we first note that if $P \cap \mathcal{B}$ is of Type σ_1, σ_2, or σ_3, then $P \cap \mathcal{B}$ consists of a single trail and hence can be only of Type π_1. Thus, $\pi_1 = \pi_{f(i)}$ for $i = 1,2,3$. Now, suppose that $P \cap \mathcal{B}$ is of Type σ_4. Then, $P \cap \mathcal{B}$ consists of two double trails P_1 and P_2, where P_1 starts with e_s. Moreover, P_1 starts and ends at the same pole of $\mathcal{B}$, and so does P_2; the dual trail P_1^* of P_1 starts and ends at different poles of $\mathcal{B}^*$, and so does the dual trail P_2^* of P_2. Thus, $P \cap \mathcal{B}$ is of Type π_4. On the other hand, $P \cap \mathcal{B}$ is not of Type π_1 because $P \cap \mathcal{B}$ consists of two double trails; $P \cap \mathcal{B}$ is not of Type π_2 or π_3 either because P_2 starts and ends at the same pole of $\mathcal{B}$. Therefore, $\pi_4 = \pi_{f(4)}$. Similarly, we can verify that $\pi_3 = \pi_{f(5)}$. □

The mapping from σ-types to δ-types is specified by the following function: $g : \{1,2,3,5\} \to \{1,\ldots,4\}$, where $g(1) = 2$, $g(2) = 4$, $g(3) = 3$, and $g(5) = 1$.

LEMMA 6.5. *Suppose that G is Eulerian. Let μ be a doubly good nonroot node of T. Let $B = \mathrm{Pt}(\mu)$. Suppose that δ_j is a useful δ-type of B. Then, the following two statements hold:*

1. *If μ is not an S-node, then $\mathcal{K}_\sigma(B) = \{\sigma_5\}$ and $\delta_j = \delta_{g(5)}$.*

2. *If μ is an S-node, then $\mathcal{K}_\sigma(B) = \{\sigma_i\}$ for some $\sigma_i \in \{\sigma_1, \sigma_2, \sigma_3\}$ and $\delta_j = \delta_{g(i)}$.*

Proof. Suppose that a DET P of a valid embedding $\mathcal{G}$ of G starts with an edge e_s of $\mathcal{B}$ and ends with an edge e_t of $\mathcal{B}$, where $\mathcal{B}$ is the block of $\mathcal{G}$ that is a different embedding of B. Let P^* be the dual trail of P. Since G is Eulerian but G^* is not, P starts and ends at the same vertex of $\mathcal{B}$ (say, v) while P^* starts and ends at different vertices of $\mathcal{B}^*$, by Facts 1 and 2. Both e_s and e_t are incident to v.

Case 1. v is a pole of $\mathcal{B}$. Then, since μ is doubly good, P^* starts at one pole of $\mathcal{B}^*$ and ends at the other pole. Thus, $P \cap \mathcal{B}$ is of a σ-type. Moreover, since $\mathcal{B} \neq \mathcal{G}$, $P \cap \mathcal{B}$ contains more than one trail (otherwise, the single trail in $P \cap \mathcal{B}$ would be the entire DET P). Hence, $P \cap \mathcal{B}$ is of Type σ_4 or σ_5. Indeed, it is not of Type σ_4 because P starts and ends at the same vertex. Thus, $P \cap \mathcal{B}$ is of Type σ_5 and hence consists of two double trails P_1 and P_2, where P_1 starts with e_s and P_2 ends with e_t. To determine the δ-type of $P \cap \mathcal{B}$, we note that P_1 starts at v and ends at the other pole u of $\mathcal{B}$ while P_2 starts at u and ends at v. Moreover, since $P \cap \mathcal{B} = (P_1, P_2)$ is of Type σ_5, the dual trail P_1^* of P_1 starts and ends at the same pole x of $\mathcal{B}^*$ while the dual trail P_2^* of P_2 starts and ends at the other pole y of $\mathcal{B}^*$. Thus, $P \cap \mathcal{B}$ is of Type δ_1. On the other hand, $P \cap \mathcal{B}$ is not of Type δ_4, δ_5 or δ_6 because $P \cap \mathcal{B}$ consists of only two double trails. Moreover, $P \cap \mathcal{B}$ is not of Type δ_2 because the ending pole of P_1^* is different from the starting pole of P_2^*. Similarly, $P \cap \mathcal{B}$ is not of Type δ_3 because the ending pole of P_1 is the same as the starting pole of P_2. Therefore, $\delta_1 = \delta_{g(5)}$.

Case 2. v is not a pole of $\mathcal{B}$. Then, since P^* starts at one pole of $\mathcal{B}^*$ and ends at the other pole, one of e_s and e_t is on the west side of $\mathcal{B}$ and the other is on the east side. Thus, v is a cut vertex of $\mathcal{B}$ and μ is an S-node. Let $\mathcal{B}_1$ (respectively, $\mathcal{B}_2$) be the block of $\mathcal{B}$ whose source (respectively, sink) is that of $\mathcal{B}$ and whose sink (respectively, source) is v. Then, $\mathcal{B} = \mathcal{B}_1 \star \mathcal{B}_2$. Since $\mathcal{B}^*$ is good, neither $\mathcal{B}_1^*$ nor $\mathcal{B}_2^*$ has an odd-degree nonpole. We assume that e_s is on the west side of $\mathcal{B}_1$; the other cases are similar.

Case 2.1. $\mathcal{B}_1$ contains e_t. Then, e_t is on the east side of $\mathcal{B}_1$. Moreover, P^* starts at the source of $\mathcal{B}_1^*$ and ends at the sink of $\mathcal{B}_1^*$, by Facts 1 and 2. As in Case 1, we can prove that $P \cap \mathcal{B}_1$ is of Type σ_5 and hence consists of two double trails P_1 and P_3, where P_1 starts with e_s and P_3 ends with e_t. On the other hand, $P \cap \mathcal{B}_2$ must start at the sink of $\mathcal{B}_2$ and can leave $\mathcal{B}_2$ only at its sink; so, $P \cap \mathcal{B}_2$ consists of a single trail P_2 and is of Type σ_2. Consequently, $P \cap \mathcal{B}$ is of Type $\sigma_5 \star \sigma_2 = \sigma_2$ by Table 4.1. To determine the δ-type of $P \cap \mathcal{B}$, we note that P_1 starts at v and ends at the other pole u of $\mathcal{B}_1$ while P_3 starts at u and ends at v. Moreover, P_2 starts at the pole $w \neq v$ of $\mathcal{B}_2$ and ends at w. Since u, v, w are three distinct vertices of $\mathcal{B}$, no two of the trails P_1 through P_3 can be concatenated into a single trail in $P \cap \mathcal{B}$. So, $P \cap \mathcal{B} = (P_1, P_2, P_3)$ because P_1 starts with e_s and P_3 ends with e_t. In turn, $P \cap \mathcal{B}$ is of Type δ_4. On the other hand, $P \cap \mathcal{B}$ is not of Type δ_1, δ_2, or δ_3 because $P \cap \mathcal{B}$ consists of three trails; $P \cap \mathcal{B}$ is not of Type δ_5 or δ_6 because P_2 starts and ends at the same vertex of $\mathcal{B}$. Therefore, $\delta_4 = \delta_{g(2)}$.

Case 2.2. $\mathcal{B}_1$ does not contain e_t. Then, e_t is on the east side of $\mathcal{B}_2$. Moreover, by Facts 1 and 2, P^* starts at the source of $\mathcal{B}_1^*$ and ends at the sink of $\mathcal{B}_2^*$. By this and the fact that P starts and ends at the common pole v of $\mathcal{B}_1$ and $\mathcal{B}_2$, $P \cap \mathcal{B}_1$ must start at v and cannot reenter $\mathcal{B}_1$ after leaving it, while $P \cap \mathcal{B}_2$ must start at the sink of $\mathcal{B}_2$ and cannot leave $\mathcal{B}_2$ after entering it. Consequently, $P \cap \mathcal{B}_1$ consists of a single double trail P_1, and $P \cap \mathcal{B}_2$ consists of a single double trail P_2 too. So, both $P \cap \mathcal{B}_1$ and $P \cap \mathcal{B}_2$ are of Type π_1. In turn, $P \cap \mathcal{B} = (P_1, P_2)$ is of Type δ_2 or δ_3 by

Table 5.3; $P \cap \mathcal{B}$ cannot be of Type δ_0 or else $\mathcal{B}$ could not have been a proper block of G. Moreover, each of P_1 and P_2 starts and ends at different vertices; so each of $P \cap \mathcal{B}_1$ and $P \cap \mathcal{B}_2$ is of Type σ_1 or σ_3.

Suppose that one of $P \cap \mathcal{B}_1$ and $P \cap \mathcal{B}_2$ is of Type σ_1 and the other is of Type σ_3. We assume that $P \cap \mathcal{B}_1$ is of Type σ_1 and $P \cap \mathcal{B}_2$ is of Type σ_3; the other case is similar. Then, the unique σ-type of $\mathcal{B}$ is $\sigma_1 \star \sigma_3 = \sigma_1$ by Table 4.1. Note that the starting vertex of P^* (which is also the starting vertex of the dual trail P_1^* of P_1) and the ending vertex of P^* (which is also the ending vertex of the dual trail P_2^* of P_2) are different vertices in G^*. Since $P \cap \mathcal{B}_1 = (P_1)$ is of Type σ_1, the starting and the ending vertices of P_1^* must be different. On the other hand, since $P \cap \mathcal{B}_2 = (P_2)$ is of Type σ_3, the starting and the ending vertices of P_2^* must be the same. These requirements together imply that the ending vertex of P_1^* and the starting vertex of P_2^* must be the same pole of $\mathcal{B}^*$. Therefore, the ODTC $P \cap \mathcal{B} = (P_1, P_2)$ is not of Type δ_3 but is of Type δ_2. So, $\delta_2 = \delta_{g(1)}$.

Suppose that both of $P \cap \mathcal{B}_1$ and $P \cap \mathcal{B}_2$ are of Type σ_1 or both of them are of Type σ_3. Then, the unique σ-type of $\mathcal{B}$ is σ_3 by Table 4.1. Similarly to the above case where $P \cap \mathcal{B}_1$ and $P \cap \mathcal{B}_2$ are of different σ-types, we can prove that the ending vertex of the dual trail P_1^* of P_1 and the starting vertex of the dual trail P_2^* of P_2 must be different poles of $\mathcal{B}^*$. Hence, the ODTC $P \cap \mathcal{B} = (P_1, P_2)$ is not of Type δ_2 but is of Type δ_3. So, $\delta_3 = \delta_{g(3)}$. □

LEMMA 6.6. *Suppose that G is not Eulerian. Let μ be a doubly good nonroot node of T. Let $B = \mathrm{Pt}(\mu)$. Suppose that δ_j is a useful δ-type of B. Then, $\mathcal{K}_\sigma(B)$ equals $\{\sigma_4\}$ or $\{\sigma_5\}$ and $\delta_j = \delta_3$.*

Proof. Suppose that a DET P of a valid embedding $\mathcal{G}$ of G starts with an edge e_s of $\mathcal{B}$ and ends with an edge e_t of $\mathcal{B}$, where $\mathcal{B}$ is the block of $\mathcal{G}$ that is a different embedding of B. Let P^* be the dual trail of P. Since $\mathcal{B}$ is a proper block of $\mathcal{G}$, $P \cap \mathcal{B}$ consists of at least two trails P_1 and P_2, where P_1 starts with e_s and P_2 ends with e_t. Since μ is doubly good, neither $\mathcal{B}$ nor $\mathcal{B}^*$ has an odd-degree nonpole. By Facts 1 and 2, P starts at a pole of $\mathcal{B}$ and ends at the other pole of $\mathcal{B}$, and P^* starts at a pole of $\mathcal{B}^*$ and ends at the other pole of $\mathcal{B}^*$. So, both e_s and e_t are access edges of $\mathcal{B}$. In turn, each of P_1 and P_2 uses exactly two access edges of $\mathcal{B}$. Since $\mathcal{B}$ has at most four access edges, P_1 and P_2 are the only two trails in $P \cap \mathcal{B}$. Therefore, $P \cap \mathcal{B} = (P_1, P_2)$ is of Type σ_4 or σ_5. Let P_1^* and P_2^* be the dual trail of P_1 and P_2, respectively.

Case 1. $P \cap \mathcal{B}$ is of Type σ_4. Then, P_1 starts at a pole u of $\mathcal{B}$ and ends at u while P_1^* starts at a pole x of $\mathcal{B}^*$ and ends at the other pole y of $\mathcal{B}^*$. Because neither $\mathcal{G}$ nor $\mathcal{G}^*$ is Eulerian, P_2 starts at the other pole v of $\mathcal{B}$ and ends at v while P_2^* starts at x and ends at y. Thus, $P \cap \mathcal{B}$ is of Type δ_3.

Case 2. $P \cap \mathcal{B}$ is of Type σ_5. Then, P_1 starts at a pole u of $\mathcal{B}$ and ends at the other pole v of $\mathcal{B}$ while P_1^* starts at a pole x of $\mathcal{B}^*$ and ends at x. Because neither $\mathcal{G}$ nor $\mathcal{G}^*$ is Eulerian, P_2 starts at u and ends at v while P_2^* starts at the other pole y of $\mathcal{B}^*$ and ends at y. Thus, $P \cap \mathcal{B}$ is of Type δ_3. □

Let μ be a node of T, and let ν be a child of μ in T. A σ-, π-, or δ-type γ_1 of $\mathrm{Pt}(\mu)$ is *consistent* with a σ-, π-, or δ-type γ_2 of $\mathrm{Pt}(\nu)$ if a different embedding H of $\mathrm{Pt}(\mu)$ has a Type-γ_1 DTC $\mathcal{Q}$ such that $\mathcal{Q} \cap B$ is a Type-γ_2 DTC of B, where B is the block of H that is a different embedding of $\mathrm{Pt}(\nu)$.

To describe the improved algorithm concisely, we abuse the notation to use $\mathcal{K}_\pi(B)$ (respectively, $\mathcal{K}_\delta(B)$) to mean its arbitrary subset that contains all useful π-types (respectively, δ-types) of a block B of G.

The algorithm works as follows. First, for all nodes μ of T, we determine if μ is a

good node of T and compute the pole-parity multiset of $\mathrm{Pt}(\mu)$ if so; we also determine if μ is a good node of T^* and compute the pole-parity multiset of the dual graph of $\mathrm{Pt}(\mu)$ if so. By Lemma 6.1, this takes $O(|G|)$ time in total. Then, we find the critical nodes of T based on Lemma 6.1. If T has more than two critical nodes, then we stop immediately since G has no valid embedding by Lemma 6.2. So, suppose that T has one or two critical nodes. For all doubly good nonroot nodes μ of T, we compute $\mathcal{K}_\sigma(\mu)$ as in section 4.3. After this, we distinguish two cases as follows.

- *Case* 1. T has two critical nodes, say μ and μ'. Let $B = \mathrm{Pt}(\mu)$ and $B' = \mathrm{Pt}(\mu')$. Then, μ is not an ancestor of μ' and vice versa. Thus, both μ and μ' are doubly bad, and every DET of a valid embedding of G starts with an edge of one of B and B' and ends with an edge of the other. We need to compute $\mathcal{K}_\pi(B)$ and $\mathcal{K}_\pi(B')$. After knowing $\mathcal{K}_\pi(B)$ and $\mathcal{K}_\pi(B')$, we are done by calling the algorithm in section 5.4 to compute $\mathcal{K}_\delta(G)$. We describe the computation of $\mathcal{K}_\pi(B)$ below. $\mathcal{K}_\pi(B')$ is computed similarly. Let $\nu_1, \ldots, \nu_q$ be the children of μ in T. For each $i \in \{1, \ldots, q\}$, let e_i be the virtual edge of ν_i in $\mathrm{Sk}(\mu)$. If $|\mathcal{K}_\sigma(\nu_i)| = 0$ for some $i \in \{1, \ldots, q\}$, then $\mathcal{K}_\pi(B) = \emptyset$ by Lemma 6.4. So, suppose $\mathcal{K}_\sigma(\nu_i)$ consists of a unique $\sigma_{k_i} \in \Sigma$ for all $i \in \{1, \ldots, q\}$.
  - *Case* 1.1. μ is an R-node. Let $H = \mathrm{Sk}(\mu)$. By an *extreme vertex* of H, we mean a pole of H or a nonpole v of H such that for an odd number of edges e_i incident to v in H, ν_i is an O-good node of T. Similarly, by an *extreme face* of H, we mean the west or east side of H or an inner face F of H such that for an odd number of edges e_i on the boundary of F, ν_i is an O-good node of T^*. If H has more than three extreme vertices, then it has at least two odd-degree nonpoles, x and y. In every DET P of G, each of x and y must be either the starting vertex or the ending vertex. However, we allow only one of the starting and the ending vertices of P to be in H. Hence, G has no DET in this case. Similarly, if H has more than three extreme faces, then G has no valid embedding. Let W be the set of all children ν_i of μ such that e_i is incident to an extreme vertex of H and is also on an extreme face of H. Since μ is an R-node, for each (extreme) vertex v of H and each (extreme) face F of H, at most two edges incident to v are on F; so, a very rough estimate shows $|W| \leq 18$. Moreover, every DET of a valid embedding of G either starts or ends with an edge of $\mathrm{Pt}(\nu_i)$ for some $\nu_i \in W$. Thus, for each $\nu_i \in W$, we set $\mathcal{K}_\pi(\mathrm{Pt}(\nu_i)) = \{\pi_{f(k_i)}\}$ and then compute the set Π_i of all π-types of B consistent with Type $\pi_{f(k_i)}$ of $\mathrm{Pt}(\nu_i)$ as in Case 3.1 in section 5.4. By Lemma 6.4 and statements 1 and 2 in Lemma 3.2, $\cup_{\nu_i \in W} \Pi_i$ is $\mathcal{K}_\pi(B)$.
  - *Case* 1.2. μ is an S-node (respectively, P-node). For each $j \in \{1, \ldots, 5\}$, let $I_j = \{i \in \{1, \ldots, q\} \mid \sigma_{k_i} = \sigma_j\}$. For each $j \in \{1, \ldots, 5\}$ such that $I_j \neq \emptyset$, we select an arbitrary $i \in I_j$, set $\mathcal{K}_\pi(\mathrm{Pt}(\nu_i)) = \{\pi_{f(k_i)}\}$, and compute the set Π_j of all π-types of B consistent with Type $\pi_{f(k_i)}$ of $\mathrm{Pt}(\nu_i)$ as in Case 3.2 (respectively, 3.3) in section 5.4. By Lemma 6.4 and statements 1 and 2 in Lemma 3.2, $\cup_{1 \leq j \leq 5, I_j \neq \emptyset} \Pi_j$ is $\mathcal{K}_\pi(B)$.
- *Case* 2. T has exactly one critical node, say μ. Let $B = \mathrm{Pt}(\mu)$. If μ is the root of T, we need to compute $\mathcal{K}_\delta(B)$; otherwise, we need to compute $\mathcal{K}_\pi(B)$ and $\mathcal{K}_\delta(B)$. The former case is easier, and here we consider only the latter case. So, we assume that μ is not the root. Then, μ is doubly bad. Let $\nu_1, \ldots, \nu_q$ be the children of μ in T. For each $i \in \{1, \ldots, q\}$, let e_i be the virtual edge of ν_i in $\mathrm{Sk}(\mu)$. If $\mathcal{K}_\sigma(\nu_i) = \emptyset$ for some $i \in \{1, \ldots, q\}$, then $\mathcal{K}_\pi(B) = \mathcal{K}_\delta(B) = \emptyset$ by

Lemmas 6.4, 6.5, and 6.6. So, suppose $\mathcal{K}_\sigma(\nu_i) = \{\sigma_{k_i}\}$ for all $i \in \{1, \ldots, q\}$. For each $j \in \{1, \ldots, 5\}$, let $I_j = \{i \in \{1, \ldots, q\} \mid \sigma_{k_i} = \sigma_j\}$. For each $j \in \{1, 2, 3\}$, let $I'_j = \{i \in I_j \mid \nu_i \text{ is an S-node}\}$. Let $I'_5 = \{i \in I_5 \mid \nu_i \text{ is not an S-node}\}$.

- *Case* 2.1. μ is an R-node. $\mathcal{K}_\pi(B)$ can be computed as in Case 1.1. We next explain the computation of $\mathcal{K}_\delta(B)$. Let $H = \mathrm{Sk}(\mu)$. As in Case 1.1, we define *extreme vertices* and *extreme faces* of H. If H has more than four extreme vertices, then it has at least three odd-degree nonpoles and each of them must be the starting or the ending vertex of any DET of G. Hence, G has no DET in this case. Similarly, if H has more than four extreme faces, then G has no valid embedding. If G is not Eulerian, we define a set W of children of μ as in Case 1.1; otherwise, let W be the set of all children ν_i of μ such that (1) e_i is on an extreme face of H and (2) at least one endpoint of e_i is on two extreme faces of H. Since μ is an R-node, for each pair of (extreme) faces of H, at most two vertices can appear on both of them; so, a very rough estimate shows $|W| \leq 96$. Moreover, every DET of a valid embedding of G that starts and ends both with edges of B must start with an edge of $\mathrm{Pt}(\nu_i)$ and end with an edge of $\mathrm{Pt}(\nu_j)$ for some $\nu_i, \nu_j \in W$ (possibly $\nu_i = \nu_j$). Thus, for each $\nu_i \in W$ and each $\nu_j \in W$ with $\nu_i \neq \nu_j$, we set $\mathcal{K}_\pi(\mathrm{Pt}(\nu_i)) = \{\pi_{f(k_i)}\}$ and $\mathcal{K}_\pi(\mathrm{Pt}(\nu_j)) = \{\pi_{f(k_j)}\}$, and further compute the set $\Delta_{i,j}$ of all δ-types of B consistent with both Type $\pi_{f(k_i)}$ of $\mathrm{Pt}(\nu_i)$ and Type $\pi_{f(k_j)}$ of $\mathrm{Pt}(\nu_j)$, as in Case 1.1 in section 5.4. Moreover, for each $\nu_i \in W$, we compute a set Δ_i as follows. In case G is not Eulerian, if $\sigma_{k_i} \in \{\sigma_4, \sigma_5\}$, then we set $\mathcal{K}_\delta(\mathrm{Pt}(\nu_i)) = \{\delta_3\}$, and set Δ_i to be the set of all δ-types of B consistent with Type δ_3 of $\mathrm{Pt}(\nu_i)$, as in Case 2.1 in section 5.4; otherwise, we set $\Delta_i = \emptyset$. In case G is Eulerian, if either (1) ν_i is not an S-node and $\sigma_{k_i} = \sigma_5$ or (2) ν_i is an S-node and $\sigma_{k_i} \in \{\sigma_1, \sigma_2, \sigma_3\}$, then we set $\mathcal{K}_\delta(\mathrm{Pt}(\nu_i)) = \{\delta_{g(k_i)}\}$, and set Δ_i to be the set of all δ-types of B consistent with Type $\delta_{g(k_i)}$ of $\mathrm{Pt}(\nu_i)$, as in Case 2.1 in section 5.4; otherwise, we set $\Delta_i = \emptyset$. By Lemmas 3.2, 6.4, 6.5, and 6.6, the union of the computed sets $\Delta_{i,j}$ and Δ_i is $\mathcal{K}_\delta(B)$.
- *Case* 2.2. μ is an S-node (respectively, P-node). $\mathcal{K}_\pi(B)$ can be computed as in Case 1.2. We next explain the computation of $\mathcal{K}_\delta(B)$. For each $j \in \{1, \ldots, 5\}$ with $|I_j| \geq 2$, we arbitrarily select two distinct $i, i' \in I_j$, set $\mathcal{K}_\pi(\mathrm{Pt}(\nu_i)) = \{\pi_{f(k_i)}\}$ and $\mathcal{K}_\pi(\mathrm{Pt}(\nu_{i'})) = \{\pi_{f(k_{i'})}\}$, and compute the set Δ_j of all δ-types of B consistent with both Type $\pi_{f(k_i)}$ of $\mathrm{Pt}(\nu_i)$ and Type $\pi_{f(k_{i'})}$ of $\mathrm{Pt}(\nu_{i'})$, as in Case 1.2 (respectively, 1.3) in section 5.4. Also, for each pair of distinct $j, j' \in \{1, \ldots, 5\}$ with $I_j \neq \emptyset$ and $I_{j'} \neq \emptyset$, we arbitrarily select one $i \in I_j$ and one $i' \in I_{j'}$, set $\mathcal{K}_\pi(\mathrm{Pt}(\nu_i)) = \{\pi_{f(k_i)}\}$ and $\mathcal{K}_\pi(\mathrm{Pt}(\nu_{i'})) = \{\pi_{f(k_{i'})}\}$, and compute the set $\Delta_{j,j'}$ of all δ-types of B consistent with both Type $\pi_{f(k_i)}$ of $\mathrm{Pt}(\nu_i)$ and Type $\pi_{f(k_{i'})}$ of $\mathrm{Pt}(\nu_{i'})$, as in Case 1.2 (respectively, 1.3) in section 5.4. Furthermore, in case G is Eulerian, for each $j \in \{1, 2, 3, 5\}$ with $I'_j \neq \emptyset$, we arbitrarily select one $i \in I'_j$, set $\mathcal{K}_\delta(\mathrm{Pt}(\nu_i)) = \{\delta_{g(k_i)}\}$, and compute the set Δ'_j of all δ-types of B consistent with Type $\delta_{g(k_i)}$ of $\mathrm{Pt}(\nu_i)$, as in Case 2.2 (respectively, 2.3) in section 5.4. On the other hand, in case G is not Eulerian, for each $j \in \{4, 5\}$ with $I_j \neq \emptyset$, we arbitrarily select one $i \in I_j$, set $\mathcal{K}_\delta(\mathrm{Pt}(\nu_i)) = \{\delta_3\}$, and compute the set Δ'_j of all δ-types of

B consistent with Type δ_3 of $\mathrm{Pt}(\nu_i)$, as in Case 2.2 (respectively, 2.3) in section 5.4. By Lemmas 3.2, 6.4, and 6.5, the union of the computed sets Δ_j, $\Delta_{j,j'}$, and Δ'_j is $\mathcal{K}_\delta(B)$.

After processing the unique critical node μ, we know $\mathcal{K}_\pi(B)$ and $\mathcal{K}_\delta(B)$. Note that μ is doubly bad and $\mathcal{K}_\sigma(B) = \emptyset$. We then proceed to processing the parent χ of μ. Let $B_\chi = \mathrm{Pt}(\chi)$. Let $\mu_1, \ldots, \mu_p$ be the children of χ, where $\mu_1 = \mu$. For each $i \in \{1, \ldots, p\}$, let e'_i be the virtual edge of μ_i in $\mathrm{Sk}(\chi)$. Since B or B^* has an odd-degree nonpole, each DET of a valid embedding of G must start and/or end with an edge of B. So, for each $i \in \{2, \ldots, p\}$, if $\mathcal{K}_\sigma(\mathrm{Pt}(\mu_i)) = \emptyset$, then $\mathcal{K}_\pi(\mathrm{Pt}(\mu_i)) = \emptyset$ too, and hence G has no valid embedding. Thus, we may assume each $\mathcal{K}_\sigma(\mathrm{Pt}(\mu_i))$ consists of a unique $\sigma_{\ell_i} \in \Sigma$. $\mathcal{K}_\pi(B_\chi)$ can be computed from $\mathcal{K}_\pi(B), \mathcal{K}_\sigma(\mathrm{Pt}(\mu_2)), \ldots, \mathcal{K}_\sigma(\mathrm{Pt}(\mu_p))$, as in Case 3 in section 5.4. To compute $\mathcal{K}_\delta(B_\chi)$, we first compute the set Δ_0 of all δ-types of B_χ consistent with a δ-type in $\mathcal{K}_\delta(B)$, as in Case 2 in section 5.4. After this, for each $\pi_h \in \mathcal{K}_\pi(B)$, we compute a set Δ_h as follows.

- *Case* 2.3. χ is an R-node. We use Lemma 6.3 to compute the unique pole-status S of the different embeddings of B that have a π-type DTC. Let $H = \mathrm{Sk}(\chi)$. Let v_1 and v_2 be the endpoints of e'_1. Let F_1 and F_2 be the endpoints of the dual edge of e'_1. For each list $L = (a_1, a_2, b_1, b_2)$ such that $S = (\langle a_1, a_2\rangle, \langle b_1, b_2\rangle)$, we compute a set Δ_L as follows. For each vertex $v \notin \{v_1, v_2\}$ of H, we define v to be an *extreme vertex* of H as in Case 1.1. Similarly, for each vertex $F \notin \{F_1, F_2\}$ of H^*, we define F to be an *extreme face* of H as in Case 1.1. For each $j \in \{1, 2\}$, we say that v_j is an *extreme vertex* of H if (1) v_j is a pole of H, or (2) $a_j = 0$ and an odd number of edges $e'_i \neq e'_1$ incident to v_j satisfy that μ_i is an O-good node of T, or (3) $a_j = 1$ and an even number of edges $e'_i \neq e'_1$ incident to v_j satisfy that μ_i is an O-good node of T. Similarly, for each $j \in \{1, 2\}$, we say that F_j is an *extreme face* of H if (1) F_j is a pole of H^*, or (2) $b_j = 0$ and an odd number of edges $e'_i \neq e'_1$ on F_j satisfy that μ_i is an O-good node of T^*, or (3) $b_j = 1$ and an even number of edges $e'_i \neq e'_1$ on F_j satisfy that μ_i is an O-good node of T^*. As in Case 2.1, H has at most four extreme vertices and at most four extreme faces, unless G has no valid embedding. As in Case 2.1, we define a set W of children μ_i of χ. Let $W' = W - \{\mu\}$. For each $\mu_i \in W'$, we set $\mathcal{K}_\pi(\mathrm{Pt}(\mu_i)) = \{\pi_{f(\ell_i)}\}$, and then compute the set Δ'_i of all δ-types of B_χ consistent with both Type π_h of $\mathrm{Pt}(\mu)$ and Type $\pi_{f(\ell_i)}$ of $\mathrm{Pt}(\mu_i)$, as in Case 1.1 in section 5.4. Let $\Delta_L = \cup_{\mu_i \in W'} \Delta'_i$. Let Δ_h be the union of the computed sets Δ_L.
- *Case* 2.4. χ is an S-node (respectively, P-node). For each $j \in \{1, \ldots, 5\}$, let $I''_j = \{i \in \{2, \ldots, p\} \mid \sigma_{\ell_i} = \sigma_j\}$. For each $j \in \{1, \ldots, 5\}$ with $|I''_j| \geq 1$, we arbitrarily select one $i \in I''_j$, set $\mathcal{K}_\pi(\mathrm{Pt}(\mu_i)) = \{\pi_{f(\ell_i)}\}$, and compute the set Δ'_j of all δ-types of B_χ consistent with both Type π_h of $\mathrm{Pt}(\mu)$ and Type $\pi_{f(\ell_i)}$ of $\mathrm{Pt}(\mu_i)$, as in Case 1.2 (respectively, 1.3) in section 5.4. Let Δ_h be the union of the computed sets Δ'_j.

By Lemmas 3.2, 6.4, 6.5, 6.6, and 6.3, $\mathcal{K}_\delta(B_\chi) = \Delta_0 \cup (\cup_{\pi_h \in \mathcal{K}_\pi(B)} \Delta_h)$. This completes the processing of χ; each ancestor of χ is processed in the same way. Once we finish processing the root of T, we have $\mathcal{K}_\delta(G)$; G has a valid embedding if and only if $\delta_0 \in \mathcal{K}_\delta(G)$.

Obviously, processing each node μ of T takes $O(|\mathrm{Sk}(\mu)|)$ time. Thus, we have the following.

THEOREM 6.7. *Given a TTG G whose dual graph is not Eulerian, it takes $O(|G|)$ time to decide if G has a valid embedding.*

Acknowledgment. The authors would like to thank the referee for suggestions that considerably improved the readability of this paper.

REFERENCES

[1] B. S. CARLSON, C. Y. R. CHEN, AND D. S. MELIKSETIAN, *Dual Eulerian properties of plane multigraphs*, SIAM J. Discrete Math., 8 (1995), pp. 33–50.

[2] G. DI BATTISTA AND R. TAMASSIA, *On-line maintenance of triconnected components with spqr-trees*, Algorithmica, 15 (1996), pp. 302–318.

[3] G. DI BATTISTA AND R. TAMASSIA, *On-line planarity testing*, SIAM J. Comput., 25 (1996), pp. 956–997.

[4] S. EVEN AND R. TARJAN, *Computing an st-numbering*, Theoret. Comput. Sci., 2 (1976), pp. 339–344.

[5] T.-Y. HO, T.-Y. SUNG, L.-H. HSU, C.-H. TSAI, AND J.-Y. HWANG, *The recognition of double Euler trails in series-parallel networks*, J. Algorithm, 28 (1998), pp. 216–257.

[6] A. LEMPEL, S. EVEN, AND I. CEDERBAUM, *An algorithm for planarity testing of graphs*, in Theory of Graphs, International Symposium, Gordon Breach, New York, 1967, pp. 215–232.

[7] T. LENGAUER AND R. MÜLLER, *Linear algorithms for optimizing the layout of dynamic CMOS cells*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 279–285.

[8] R. L. MAZIASZ AND J. P. HAYES, *Layout optimization of static CMOS functional cells*, in Proceedings of the 24th ACM/IEEE Design Automation Conference, 1987, ACM, New York, pp. 544–551.

[9] R. L. MAZIASZ AND J. P. HAYES, *Layout optimization of static CMOS functional cells*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 708–719.

[10] R. NAIR, A. BUSS, AND J. REIF, *Linear time algorithms for optimal CMOS layout*, in VLSI: Algorithms and Architectures, P. Bertolazzi and F. Luccio, eds., Elsevier Science, North-Holland, Amsterdam, 1985, pp. 327–338.

[11] T. NISHIZEKI AND N. CHIBA, *Planar Graphs: Theory and Algorithms*, North-Holland, Amsterdam, 1988.

[12] P. ROSENSTIEHL AND R. E. TARJAN, *Rectilinear planar layouts and bipolar orientations of planar graphs*, Discrete Comput. Geom., 1 (1986), pp. 343–353.

[13] T. UEHARA AND W. M. VANCLEEMPUT, *Optimal layout of CMOS functional arrays*, IEEE Trans. Comput., 30 (1981), pp. 305–312.

[14] S. UENO, K. TSUJI, AND Y. KAJITANI, *On dual Eulerian paths and in plane graphs*, in Proceedings of the IEEE International Symposium on Circuits and Systems, Espoo, Finland, 1988, pp. 1835–1838.

[15] S. UENO, K. TSUJI, AND Y. KAJITANI, *A note on dual trail partition of a plane graph*, IEICE Trans., E 74 (1991), pp. 1915–1917.

SIAM J. COMPUT.
Vol. 31, No. 4, pp. 1286–1313

THE COMBINATORIAL STRUCTURE OF WAIT-FREE SOLVABLE TASKS*

HAGIT ATTIYA† AND SERGIO RAJSBAUM‡

Abstract. This paper presents a self-contained study of wait-free solvable tasks. A new necessary condition for wait-free solvability, based on a restricted set of executions, is proved. This set of executions induces a very simple-to-understand structure, which is used to prove tight bounds for k-set consensus and renaming. The framework is based on topology, but uses only elementary combinatorics, and, in contrast to previous works, does not rely on algebraic or geometric arguments.

Key words. distributed systems, shared memory systems, atomic read/write registers, combinatorial topology, wait-free solvable tasks, consensus, set consensus, renaming

AMS subject classifications. 68Q10, 68Q22, 68Q25, 68R05, 68R10

PII. S0097539797330689

1. Introduction.

This paper studies the tasks that can be solved by a wait-free protocol in a shared-memory asynchronous system, consisting of $n+1$ processes that communicate by reading and writing atomic shared registers. We assume that processes are completely asynchronous and run at completely arbitrary speeds. Processes start with *inputs* and, after performing some protocol, have to decide on some *outputs*. A *task* specifies the sets of outputs that are allowable for each assignment of inputs to processes. A protocol is *wait-free* if a process decides on an output after a finite number of its own events, regardless of the behavior of other processes. A task is *wait-free solvable* if there is a wait-free protocol solving it.

The study of wait-free solvable tasks is central to the theory of distributed computing. Early research studied specific tasks and showed them to be solvable (e.g., approximate agreement [13], $2n$-renaming [2], k-set consensus with at most $k-1$ failures [11]) or unsolvable (e.g., consensus [15], $n+1$-renaming [2]). In 1988, a necessary and sufficient condition for the solvability of a task in the presence of a single process failure was presented [6]. In 1993, a significant advancement was made in the understanding of wait-free solvability [7, 22, 25], yielding new impossibility results for k-set consensus ([7, 22, 25], and later [10, 19, 20]) and renaming [20, 22], as well as a necessary and sufficient condition for wait-free solvability [22, 23] (which is undecidable [16, 21]).

Of particular interest is the use of topological notions to investigate the problem, suggested in [22, 25] and implicit in [7]. Yet, much of this development remained inaccessible to many researchers, since it relied on algebraic and geometric tools of topology. Furthermore, different topology techniques were used in each of these works,

*Received by the editors December 1, 1997; accepted for publication (in revised form) November 26, 2001; published electronically May 8, 2002. A preliminary version of this paper appeared in *Proceedings of the 10th International Workshop on Distributed Algorithms*, Lecture Notes in Comput. Sci. 1151, O. Babaoglu and K. Marzullo, eds., Springer-Verlag, Berlin, 1996, pp. 321–343.
http://www.siam.org/journals/sicomp/31-4/33068.html

†Department of Computer Science, The Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il). This author was supported by United States–Israel Binational Science Foundation (BSF) grant 92-0233, Jerusalem, Israel, and the fund for the promotion of research in the Technion.

‡Instituto de Matemáticas, UNAM, Ciudad Universitaria, D.F. 04510, México (rajsbaum@math.unam.mx, Sergio.Rajsbaum@compaq.com). Part of this author's work was done while visiting the MIT Laboratory for Computer Science and was supported by CONACyT and UNAM-DGAPA projects. This author is currently on leave at the Cambridge Research Laboratory of Compaq.

for example, a direct, combinatorial application of Sperner's lemma in [7], a more involved form of Brouwer fixed point theorem in [25], and a combination of continuous and algebraic tools of simplicial approximation and Mayer–Vietoris sequences in [22].

In this paper, we study wait-free solvable tasks, starting from first principles. We introduce a new necessary condition for wait-free solvability and use it to prove the impossibility of solving k-set consensus and renaming. Our approach borrows critical ideas from previous works in this area, especially [7, 8, 20, 22, 23, 25], and integrates them into a unified framework. Our goal was to deepen the understanding of the wait-free solvability problem, in particular to learn which tools of topology are needed to investigate this problem.

1.1. Our contributions. We develop a formal model that captures wait-free solvability in terms of combinatorial mathematics, inspired by topology. To explain our model, here is a rough description of key notions from combinatorial topology. (Precise definitions appear in section 4.1.)

A *colored simplex* is a set in which every element, called a *vertex*, is colored with a different process id. A *colored complex* is a collection of colored simplexes which is closed under containment. A mapping from the vertices of one colored complex to the vertices of another is *simplicial* if it maps a simplex to a simplex; it is *color preserving* if each vertex is mapped to a vertex with the same process id. Finally, a complex whose largest simplex contains m vertices is a *pseudomanifold* if every simplex with $m-1$ vertices is contained in either one or two simplexes with m vertices.

Although these definitions do not require algebraic or geometric interpretations, it is sometimes useful to have a topological intuition in mind. In topology a complex is viewed as a discretization of a continuous space. For example, in three dimensions, a pseudomanifold is obtained by "chopping" a surface into triangular pieces, where each one corresponds to a simplex. If we chop it into smaller and smaller pieces, we obtain closer approximations to the surface, and it is said that finer *subdivisions* are obtained. A simplicial map is analogous to a continuous map, and in topology is often interpreted as instructions of how to bend and fold one complex into another, possibly stretching it but without tearing it apart.

The novel combinatorial concept we use is of a pseudomanifold being a *divided image* of a simplex, having the same boundary as the simplex. A divided image of a complex is obtained by taking a divided image of each one of its simplexes and pasting them together. (An example appears in Figure 2.) A subdivision of a complex preserves all the topological structure of the complex, while a divided image preserves only some of its topological structure. (The divided image shown in Figure 2 is actually a subdivision.) Our results show that the topological structure that is preserved suffices to prove the set consensus impossibility, and if in addition the divided image is orientable we can also prove the renaming impossibility. This new necessary condition for wait-free solvability is our main result, Theorem 5.14.

Given a task, and a wait-free protocol solving it, we show that a subset of the protocol's executions, called *immediate snapshot executions* [7, 25], induces a divided image of the input complex. The decisions made by the protocol induce a simplicial map from this divided image to the output complex which must agree with the task specification. A solution for the participating set task [8] is used to show that the above property is also sufficient: If there is a simplicial map from the divided image induced by immediate snapshots executions to the output complex, which agrees with the task, then the task is wait-free solvable.

The divided image induced by immediate snapshot executions is shown to be

orientable and *connected*. We show that an orientable chromatic divided image induces an *algebraic span* [20].

Using the standard Sperner's lemma, we show that k-set consensus is wait-free solvable only if $k > n$. Using results of [20], we prove a combinatorial theorem about the number of monochromatic simplexes in any binary coloring of an orientable and connected divided image. This theorem is used to prove that M-renaming is wait-free solvable only if $M \geq 2n$. (Both bounds are tight; see [11] and [2], respectively.)

The main contribution of this paper, in our opinion, is in providing an alternative, combinatorial framework in which further research related to distributed solvability can be pursued. We believe our work is the first to provide an elementary, yet rigorous, treatment of this topic. From ideas that were developed in great generality in algebraic topology, our framework distills the properties that are required for the study of wait-free solvability, in particular properties that are sufficient to prove specific impossibility results. It does not require any previous topology background from the reader.

1.2. Relation to other work. The combinatorial topology framework we develop for wait-free solvability is related to the 1-failure model of Biran, Moran, and Zaks [6]; while they use graphs and edges (sets of two vertices), we use complexes and simplexes (sets of one or more vertices). Herlihy and Shavit also use simplexes [22, 23] but often given a geometric interpretation and considered as subsets of Euclidean space; in our work, we always consider combinatorial structures.

Divided images play a role similar to *spans* (both the geometric version used in [19, 22, 23] and the algebraic version introduced in [20]). As discussed after Definition 4.1, divided images have weaker mathematical properties than geometric spans; in particular, a divided image of a simplex may have "holes." We prove that an orientable divided image corresponds naturally to an algebraic span. It was shown that geometric spans exist [22], but the proof requires a combination of algebraic (homology theory) and geometric (subdivided simplexes) arguments.

The notion of immediate snapshot executions was introduced by Borowsky and Gafni [7] and by Saks and Zaharoglou [25]. These works contain many of the ideas needed to show that immediate snapshot executions induce a divided image. However, they centered on properties of immediate snapshot executions needed to prove the impossibility result for set consensus. For this result, it is not necessary to show that they are connected and orientable or that they induce an algebraic span or our simpler combinatorial notion of a divided image. No general conditions for wait-free solvability were derived from them, and no general model, like the one developed here, has appeared before.

The main difference between our necessary condition for wait-free solvability and the one proved by Herlihy and Shavit [22] is that we construct a *specific* well-structured divided image (induced by immediate snapshot executions), while they show that an *arbitrary* span exists.

1.3. Organization. The rest of this paper is organized as follows. In section 2, we describe the distributed computing model. In section 3, we define immediate snapshot executions and study their properties. Section 4 starts with the topology concepts used in the rest of the paper (section 4.1), and then we show how to model a task and a distributed computing system (as described in section 2) with the combinatorial topology concepts (sections 4.2 and 4.3). The condition for wait-free solvability is developed in section 5. Additional mathematics is developed in section 6, and is

later used in section 7, to prove impossibility of wait-free solutions for set consensus and renaming. We conclude with a discussion in section 8.

2. Model of computation. Our model is standard and was used in many papers; we follow [1].

A *system* consists of $n+1$ processes $p_0, \ldots, p_n$. A *process* is a deterministic state machine; each process has a (possibly infinite) set of *local states*, including a subset called the *initial states* and a subset called the *output states*. Processes communicate by means of a finite number of *single-writer multireader atomic registers*. No assumption is made regarding the size of the registers, and therefore we may assume that process p_i has a single register, R_i, to which it can write its entire state. Process p_i has two atomic operations available to it:

- $write_i$: p_i writes a value v to R_i.
- $read_i(R_j)$: p_i reads the register R_j and returns its value v.

A system configuration consists of the states of the processes and registers. Formally, a *configuration* C is a vector $\langle s_0, \ldots, s_n, v_0, \ldots, v_n \rangle$ where s_i is the local state of process p_i and v_j is the value of the register R_j. Denote $\text{state}_i(C) = s_i$. Each register may attain values from some *domain* which includes a special "undefined" value, $\perp$. An *initial configuration* is a configuration in which every local state is an initial state and all registers are set to $\perp$.

We consider an interleaving model of concurrency, where executions are modeled as sequences of events. An event is performed by a single process p_i, which applies either a $write_i$ operation or a $read_i(R_j)$ operation, but not both, performs some local computation, and changes to its next local state. The next configuration is the result of these modifications.

The state machine of each process p_i models a *local protocol*, $\mathcal{P}_i$, that determines p_i's next event—$\mathcal{P}_i$ determines whether p_i is to write or read, and (in case of a read) which register R_j to read—as a function of p_i's local state. If p_i reads R_j, then $\mathcal{P}_i$ determines p_i's next state as a function of p_i's current state and the value v read from R_j. If p_i writes to R_i, then $\mathcal{P}_i$ determines p_i's next state as a function of p_i's current state. We assume that all local protocols are identical; i.e., processes have the same state machine, which do not depend on the process id. If the local protocol has to depend on the id, then the id has to be encoded in the input (see the discussion below in this section and in section 4.3).

A *protocol* is a collection $\mathcal{P}$ of local protocols $\mathcal{P}_0, \ldots, \mathcal{P}_n$.

An *event of* p_i is denoted below simply by p_i's index, i. An *execution* of a system is a finite or infinite alternating sequence of configurations and events $C_0, j_1, C_1, \ldots, C_{k-1}, j_k, \ldots$, where C_0 is an initial configuration and C_k is the result of applying the event j_k to C_{k-1}, for all $k \geq 1$. The *schedule* of this execution is $j_1, \ldots, j_k, \ldots$.

Given an execution $\alpha = C_0, j_1, C_1, \ldots$, and a process p_i, the *view of* p_i *in* α, denoted $\alpha|i$, is the sequence $\text{state}_i(C_0), \text{state}_i(C_1), \ldots$.

To model decision tasks, we identify two special components of each process's state: an *input* and an *output*. It is assumed that initial states differ only in the value of the input component; moreover, the input component never changes. If we want to have a local protocol which depends on the process id, then the id has to be provided explicitly as part of the input. The protocol cannot overwrite the output: The output is initially $\perp$; once a non-$\perp$ value is written to the output component of the state, it never changes; when this happens, we say that the process *decides*. The output states are the states with non-$\perp$ output value.

A view of process p_i in a finite execution α is *final* if there is a configuration in α

in which p_i decides; since the output cannot be overwritten, it follows that p_i's output is not $\perp$ in some suffix of α. A final view is *minimal* if none of its prefixes is final. A minimal final view is the prefix of p_i's view in α up to (and including) the first configuration in which p_i decides.

A *task* Δ has a domain $\mathcal{I}$ of input values and a domain $\mathcal{O}$ of output values; Δ specifies for each assignment of inputs to the processes on which outputs processes can decide.

A protocol *solves* Δ if any finite execution α can be extended to an execution α' in which all processes decide on values which are allowable for the inputs in α. Because the outputs cannot be overwritten, if a process decides on a value in α, it must have the same output in α'. This means that outputs already written by the processes can be completed to outputs for all processes that are allowable for the inputs in α.

A protocol is *wait-free* if in any execution of the protocol a process either has a finite number of events or it decides. This implies that if a process has an infinite number of events, it must decide after a finite number of events.

We do not require processes to halt—they solve the decision task and decide by writing to the output component; processes can continue to participate. We typically consider the behavior of a process until it decides, and, therefore, the above distinction does not matter.

For the rest of the paper, we consider *full-information* protocols, in which a process remembers everything and writes everything it knows. Formally, we add a *history* component to the input and output components described above. The history component of p_i consists of the sequence of operations executed so far by p_i and their results. Initially, the sequence is empty; after p_i executes a $write_i$ operation, *write* is appended to the sequence; after it executes a $read_i(R_j)$, returning the value v, $read[j, v]$ is appended. Clearly, a task is solvable if and only if it can be solved by a full-information protocol; hence, there is no loss of generality in considering only full-information protocols if efficiency issues are ignored.

Since we consider only deterministic protocols, the state of a process is uniquely determined by its history component. In fact, the behavior of all full-information protocols is very similar; protocols differ only in what value a process decides on and when the process makes the decision. Thus, the decision value is a function of the input component and the history component, and various protocols differ only in this function.

We also assume, without loss of generality, that the local protocol of each process p_i is in *standard form*: p_i proceeds in a sequence of *steps*; in each step p_i writes its entire state to R_i and then reads $R_0, \ldots, R_n$ (in some fixed order); after reading, p_i determines whether to decide and on which value.

Below, we rely on the notion of executions that cannot be distinguished by a process; this happens when the process has the same view in both executions. Formally, two executions, α_1 and α_2, are *indistinguishable to process* p_i, denoted $\alpha_1 \overset{p_i}{\sim} \alpha_2$, if $\alpha_1|p_i = \alpha_2|p_i$. This notion of indistinguishability can be extended to sets of processes in the natural manner: if P is a set of processes, then $\alpha_1 \overset{P}{\sim} \alpha_2$ if and only if $\alpha_1 \overset{p_i}{\sim} \alpha_2$ for every process $p_i \in P$. Typically, we shall be interested in pairs of executions which are indistinguishable to all processes, except one. If $\alpha_1 \overset{P}{\sim} \alpha_2$, for $P = \{p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$, and $\alpha_1|p_i \neq \alpha_2|p_i$, then we denote $\alpha_1 \overset{\neg p_i}{\sim} \alpha_2$.

3. Immediate snapshot executions. We start by introducing immediate snapshot executions. In section 3.1, we prove their basic properties, and in section 3.2 these

properties are extended to prove that immediate snapshot executions are connected in a certain sense.

Consider a full-information protocol in standard form; we now restrict our attention to a subset of the executions of this protocol. An *immediate snapshot* (in short, IS) execution consists of a sequence of *rounds*. The kth round is specified by a nonempty *concurrency class* of process ids, s_k; s_k specifies the *active* processes that take a step in round k. First every process in s_k performs a write operation (in increasing order of ids), and then every process in s_k reads all the registers, i.e., performs $n+1$ read operations (in some fixed order, say increasing order of ids). Intuitively, each round contains a concurrent write by every active process, followed by a concurrent atomic snapshot by every active process.

We assume that the protocol is wait-free. Since a process decides within a finite number of its own steps, we can restrict our attention to IS executions in which all views are minimal final; that is, a process takes steps only until it decides. Specifically, all IS executions are finite, and a process decides in its last active round.

Given a protocol and initial states for the processes (inputs), an IS execution, α, is determined by the sequence of concurrency classes; thus, we abuse notation and write $\alpha = s_1, s_2, \ldots, s_l$. Sometimes, we view an IS execution as a *concatenation* of two sequences of concurrency classes, and we write $\alpha = \alpha_1\alpha_2$ if $\alpha_1 = s_1, s_2, \ldots, s_r$ and $\alpha_2 = s_{r+1}, s_{r+2}, \ldots, s_l$.

The *number of steps* of p_i in α is the number of rounds in which it is active. The *total number of steps* in α, denoted $|\alpha|$, is the sum of the number of steps of all processors.

IS executions capture the computational power of the model. If a full-information protocol $\mathcal{P}$ solves a task Δ, it solves Δ in the subset of IS executions. Conversely, a protocol for the participating set task [8] can be used to simulate IS executions in a full-information manner, implying the next lemma.

LEMMA 3.1. *Let Δ be a task. There is a wait-free protocol which solves this task if and only if there is a wait-free protocol which solves this task only in IS executions.*

3.1. Basic properties of IS executions. Here, we consider the IS executions of a given full-information protocol in standard form, with some fixed inputs. The first property of the IS executions follows from the fact that the protocol is deterministic, and the views are minimal final.

LEMMA 3.2. *If α and α' are two distinct IS executions, then α is not a prefix of α'.*

Although they are very well-structured, IS executions still contain uncertainty, since a process does not know exactly which processes are active in the last round. That is, if p_i is active in round k and observes some other process p_j to be active (i.e., perform a write), p_i does not know whether p_j is active in round $k-1$ or in round k. Consider, for example, the executions in Figure 1. Only p_0 distinguishes between executions α_1 and α_2; p_1 and p_2 have the same views in both executions and cannot distinguish between them. (Note that p_0 distinguishes between α_1 and α_2, since in α_2, p_0 reads the initial value in p_2's register, while in α_1, p_0 reads the first write of p_2.) In Lemma 3.4, we prove that this is the only uncertainty processes have in IS executions of full-information protocols.

A process p_j is *unseen* in an execution α, if there is a round k, such that $p_j \notin s_r$ for every round $r < k$ and $s_r = \{p_j\}$ for every nonempty round $r \geq k$. Intuitively, no other process ever sees a step by p_j, since p_j's steps (if any) are taken after all processes no longer take steps.

Round number	1	2	3
p_0	w	r	
p_1		w	r
p_2	w	r	

Execution α_1.

Round number	1	2	3
p_0	w	r	
p_1			w r
p_2		w	r

Execution α_2.

FIG. 1. *Executions α_1 and α_2 are indistinguishable to p_1 and p_2.*

LEMMA 3.3. *If a process p_j is unseen in an IS execution α, then there is no IS execution $\alpha' \neq \alpha$ such that $\alpha \overset{\neg p_j}{\sim} \alpha'$.*

Proof. Assume process p_j is unseen in $\alpha = s_1, \ldots, s_r$, but there exists an IS execution $\alpha' = s'_1, \ldots, s'_l \neq \alpha$ such that $\alpha \overset{\neg p_j}{\sim} \alpha'$. Let k be the round such that $s_i = \{p_j\}$ for every $i \geq k$ and $p_j \notin s_i$ for every round $y < k$.

Let k_0 be the first round in which α and α' differ, that is, $s_{k_0} \neq s'_{k_0}$. If $k_0 > \min\{r, l\}$, then one execution is a prefix of the other, which is impossible by Lemma 3.2; thus, $k_0 \leq \min\{r, l\}$.

If $k_0 \geq k$, then $s_{k_0} = \{p_j\}$ in α, while in α', s'_{k_0} must include some other process, p_i; however, p_i also distinguishes between α and α', which is a contradiction. If $k_0 < k$, then $p_j \notin s_{k_0}$. Since no process in s_{k_0} distinguishes between α and α', then the same processes must be in s'_{k_0}, which contradicts the assumption that $s_{k_0} \neq s'_{k_0}$. □

If p_j is not unseen in α, then it is *seen*; p_j is *seen* in round k, if $p_j \in s_k$, and for some $r \geq k$, there is some process $p_i \neq p_j$ that is in s_r. The *last seen round* of p_j is the largest k such that p_j is seen in round s_k; since IS executions are finite, the last seen round is well-defined.

Let $\{p_j\}^*$ be a finite (possibly empty) sequence of singleton concurrency classes, each one being $\{p_j\}$. Assume that p_j is seen in an IS execution $\alpha = s_1, \ldots, s_l$. The *tail position* of p_j in α is the smallest value of r for which α can be written as $s_1, \ldots, s_r, \{p_j\}^*$; $\{p_j\}^*$ is the *tail*, which may be empty. In this case, $s_r \neq \{p_j\}$ and $s_{r'} = \{p_j\}$ for every r', $r < r' \leq l$.

Below, the notation $\{p_j\}^*$ is used to denote the sequence of singleton concurrency classes needed to have a minimal final view for p_j; the views of other processes do not change. That is, we schedule p_j again and again in a concurrency class by itself until it decides.

LEMMA 3.4. *Let $\alpha = \alpha_1 s_k, s_{k+1}, \ldots, s_l$ be an IS execution in which a process p_j is seen. If s_k is the last seen round of p_j in α with tail position t, then consider an IS execution α' of the following form:*

(1) *If $s_k = \{p_j\}$, then $\alpha' = \alpha_1, s_k \cup s_{k+1}, s_{k+2}, s_{k+3}, \ldots, s_t, \{p_j\}^*$.*

(2) *If $s_k \neq \{p_j\}$, then $\alpha' = \alpha_1, \{p_j\}, s_k - \{p_j\}, s_{k+1}, s_{k+2}, \ldots, s_t, \{p_j\}^*$.*

In both cases, $\alpha' \overset{\neg p_j}{\sim} \alpha$, and α' is the only IS execution such that $\alpha' \overset{\neg p_j}{\sim} \alpha$.

Proof. Clearly, $\alpha' \overset{\neg p_j}{\sim} \alpha$, in both cases: p_j distinguishes between both executions and may take a different number of steps in the execution, but no other process does, since p_j cannot communicate with other processes. (Its steps afterwards are not seen.) We now show that α' is the only IS execution with this property.

First, notice that an IS execution α'' which is different from α and α' cannot be a prefix or an extension of either one, by Lemma 3.2.

If $s_k \neq \{p_j\}$ (case (2)), then clearly there is another process $p_i \in s_k$. If $s_k = \{p_j\}$ (case (1)), then there is another process $p_i \in s_{k+1}$: s_{k+1} is not empty and does not include p_j. Moreover, if some s_r, $r > k$, includes p_j, then $s_{r'} = \{p_j\}$ for every

nonempty round, with $r' \geq r$. (Otherwise, k is not the last seen round of p_j.) In both cases, for any IS execution α'', which is neither α nor α', $\alpha \overset{p_j}{\not\sim} \alpha''$, which proves the uniqueness of α'. ∎

In the similar execution, α', we either add p_j to the next concurrency class or take p_j into an earlier singleton concurrency class (by itself). Executions α_1 and α_2 in Figure 1 provide a simple example, where p_0's first step is either in a singleton class (the first concurrency class in α_2) or it is merged with the next class, with p_2 (the first concurrency class in α_2).

The views of processes other than p_j are the same in both executions, and they take the same number of steps. Process p_j distinguishes between the executions and may take a different number of steps until it decides. However, p_j distinguishes between the executions only after its last seen round and may take a different number of steps only in the tail.

3.2. Connectivity of IS executions. Recall that we are considering the IS executions of a given full-information protocol in standard form, with some fixed inputs.

The main theorem of this section is that any pair of IS executions with a common prefix γ are "transitively indistinguishable." Specifically, two IS executions α and α' are γ-*similar*, denoted $\alpha \approx_\gamma \alpha'$, if there exist IS executions $\gamma\beta_0, \ldots, \gamma\beta_r$, and processes, $p_{i_1}, \ldots, p_{i_r}$, such that $\alpha = \gamma\beta_0 \overset{\neg p_{i_1}}{\sim} \gamma\beta_1 \ldots \overset{\neg p_{i_r}}{\sim} \gamma\beta_r = \alpha'$. Note that γ-similarity is an equivalence relation. Clearly it is reflexive and symmetric, and it is a transitive property; that is, if $\alpha_1 \approx_\gamma \alpha_2$ and $\alpha_2 \approx_\gamma \alpha_3$, then $\alpha_1 \approx_\gamma \alpha_3$. A restricted case is when γ is empty, in which case the chain of indistinguishability need not have a common prefix.

The main lemma used to prove this theorem extends Lemma 3.4 to any round, not only the last seen round. We write $\alpha = \alpha_1 \cdot s_k \cdot \alpha_2$ when s_k is the kth concurrency class of α, where α_1 contains $k-1$ concurrency classes and α_2 is an arbitrary sequence of concurrency classes.

LEMMA 3.5. *Consider an IS execution $\alpha = \alpha_1 \cdot s_k \cdot \alpha_2$, in which a process $p_i \in s_k$. There is an IS execution α' of the following form:*

(1) *If $s_k = \{p_i\}$ and s_{k+1} is not empty with $p_i \notin s_{k+1}$, then $\alpha' = \alpha_1 \cdot s'_k \cdot \alpha'_2$, where $s'_k = s_{k+1} \cup \{p_i\}$.*

(2) *If $p_i \in s_k$ and $s_k - \{p_i\} \neq \emptyset$, then $\alpha' = \alpha_1 \cdot \{p_i\} \cdot s''_{k+1} \cdot \alpha''_2$, where $s''_{k+1} = s_k - \{p_i\}$.*

In both cases, $\alpha \approx_{\alpha_1} \alpha'$.

Before proving the lemma, we state two corollaries of it. The first corollary says that we can *flatten* a concurrency class, replacing it with a sequence of singleton sets with its processes. The result follows by repeatedly applying part (2) of Lemma 3.5 to all processes in s_k.

LEMMA 3.6. *Consider an IS execution $\alpha = \alpha_1 \cdot s_k \cdot \alpha_2$, where $s_k = \{p_{i_1}, \ldots, p_{i_r}\}$. Then $\alpha \approx_{\alpha_1} \alpha'$, where α' is an IS execution of the form $\alpha_1 \cdot \{p_{i_1}\} \cdots \{p_{i_r}\} \cdot \alpha'_2$.*

Since a concurrency class does not specify the order in which processes appear, the result allows us to reorder singleton concurrency classes. That is, we can take a pair of singleton concurrency classes, $\{p_i\}, \{p_j\}$ $(i \neq j)$, join it by Lemma 3.5(1) to get $\{p_j, p_i\}$, and then use Lemma 3.5(2) to flatten it in reverse order to get $\{p_j\}, \{p_i\}$. This implies the next lemma

LEMMA 3.7. *Consider an IS execution $\alpha = \alpha_1 \cdot \{p_i\} \cdot \{p_j\} \cdot \alpha_2$. Then $\alpha \approx_{\alpha_1} \alpha'$, where α' is an IS execution of the form $\alpha_1 \cdot \{p_j\} \cdot \{p_i\} \cdot \alpha'_2$.*

Now, applying Lemmas 3.6 and 3.7 to all concurrency classes of α_2, we can obtain an IS execution with only singleton sets, in any desired order.

LEMMA 3.8. *Consider an IS execution $\alpha = \alpha_1\alpha_2$. For any permutation $i_0, \ldots, i_n$ of $0, \ldots, n$, there is a unique IS execution α' of the form $\alpha_1 \cdot \{p_{i_0}\}^* \cdots \{p_{i_n}\}^*$ such that $\alpha \approx_{\alpha_1} \alpha'$.*

The uniqueness of α' follows from Lemma 3.2.

Proof of Lemma 3.5. Let L be the maximal number of steps taken in any IS execution with the same input as α.[1] We prove the lemma by reverse induction on $|\alpha_1 \cdot s_k|$. The base case is when $|\alpha_1 \cdot s_k| = L$. In this case, α_2 is empty and s_k is the last concurrency class; Lemma 3.4 can be applied to derive the lemma. (In fact, part (1) is trivial.)

For the induction step, assume the lemma holds when $|\alpha_1 \cdot s_k| \geq \ell+1$. We prove the lemma for $|\alpha_1 \cdot s_k| = \ell$. Write α_2 as a sequence of concurrency classes, $s_{k+1}, \ldots, s_{k+m}$. By the induction hypothesis, Lemma 3.8 can be applied to $\alpha_1 \cdot s_k, \ldots, s_{k+j}$, for every j, $1 \leq j \leq m$, since it has $\ell + 1$ or more steps.

To prove part (1), apply Lemma 3.8 with prefix $\alpha_1 \cdot s_k, s_{k+1}$ and move all singleton concurrency classes with p_i to the end; we get an IS execution $\alpha' = \alpha_1 \cdot s_k, s_{k+1} \cdot \{p_{i_1}\}^* \cdots \{p_{i_n}\}^* \cdot \{p_i\}^*$ such that $\alpha \approx_{\alpha_1 \cdot s_k, s_{k+1}} \alpha'$. Since p_i is last seen in s_k, Lemma 3.4(1) can be applied to obtain the desired execution.

To prove part (2), apply Lemma 3.8 with prefix $\alpha_1 \cdot s_k$ and move all singleton concurrency classes with p_i to the end; we get an IS execution $\alpha' = \alpha_1 \cdot s_k \cdot \{p_{i_1}\}^* \cdots \{p_{i_n}\}^* \cdot \{p_i\}^*$ such that $\alpha \approx_{\alpha_1 \cdot s_k} \alpha'$. Since p_i is last seen in s_k, Lemma 3.4 (2) can be applied to obtain the desired execution. □

We can now state and prove the main result of this section.

THEOREM 3.9. *If α and α' are IS executions of the same protocol with a common prefix γ and the same inputs, then $\alpha \approx_\gamma \alpha'$.*

Proof. Denote $\alpha = \gamma\alpha_2$ and $\alpha' = \gamma\alpha_2'$. Apply Lemma 3.8 to α_2 to obtain an IS execution with only singleton sets in which the processes appear in increasing order of indices $\{p_0\}^* \cdot \{p_1\}^*, \ldots, \{p_n\}^*$. Similarly, apply Lemma 3.8 to α_2' to obtain an IS execution with singleton sets in which the processes appear in the same order of indices. That is,

$$\gamma\alpha_2 \approx_\gamma \gamma\{p_0\}^* \cdot \{p_1\}^*, \ldots, \{p_n\}^* \quad \text{and} \quad \gamma\alpha_2' \approx_\gamma \gamma\{p_0\}^* \cdot \{p_1\}^*, \ldots, \{p_n\}^*.$$

By the uniqueness condition of Lemma 3.8, these IS executions are equal, and hence $\alpha \approx_\gamma \alpha'$. □

The theorem is proved completely within the domain of distributed computing. We believe that the proof technique is interesting in itself; it resembles lower bounds on the number of rounds necessary for solving consensus [14] or k-set consensus [12] in the synchronous model. This theorem directly implies that there is no wait-free protocol solving consensus.

4. Modeling tasks and protocols using topological structures. In this section we model distributed tasks using combinatorial topology; this is an adaptation of [22, 23] to our framework.

[1] The existence of L for a wait-free protocol follows from Konig's lemma. Consider the tree of executions of the protocol: the root is the initial configuration, and the children of a configuration are the configurations obtained after executing a concurrency class; the leaves are configurations with only minimal final views. Every node in the tree has finite degree, and there is no infinite path. Therefore the tree is finite.

4.1. Combinatorial topology preliminaries. We start with the basic topological concepts used in this paper. Previous papers in this area, e.g., [12, 19, 22, 23, 25], used geometric or algebraic interpretations of topological structures; in contrast, our approach is purely combinatorial, abstracting ideas from [17, 24, 26].

4.1.1. Basic notions. The basis of our definitions is the notion of a complex. A *complex* K is a collection of finite nonempty sets closed under containment; that is, if σ is an element of K, then every nonempty subset of σ is an element of K. A nonempty subset of σ is a *face* of σ. A face of σ is *proper* if it is not equal to σ. An element of a complex is called a *simplex*. A complex K' is a *subcomplex* of a complex K if $K' \subseteq K$.

The *dimension* of a simplex σ, $dim(\sigma)$, is the number of its elements minus one. A simplex of dimension m (with $m+1$ elements) is called an m-simplex. The *dimension* of a complex K is the maximum dimension of its simplexes; we consider only complexes of finite dimension. A complex of dimension m is called an m-complex. A superscript notation is often used to denote the dimension of simplexes and complexes: σ^m is an m-simplex and K^m is an m-complex.

The *vertex set* of K is the union of its 0-simplexes; the 0-simplex $\{v\}$ is identified with v.

Consider two complexes K and L. A function f from the vertices of K to the vertices of L is *simplicial* if, for every simplex $\{v_0, \ldots, v_k\}$ of K, $\{f(v_0), \ldots, f(v_k)\}$ is a simplex of L; $\{f(v_0), \ldots, f(v_k)\}$ is treated as a set, since f need not be one-to-one and may have repetitions. A simplicial map f can be extended to all simplexes of K: f maps every simplex σ of K to a simplex $f(\sigma)$ of L (perhaps of smaller dimension). We extend f to a set S of simplexes of K, by defining $f(S)$ to be the set of simplexes $f(\sigma)$ in L, where σ ranges over all simplexes of S. Clearly, if S is a subcomplex of K, then $f(S)$ is a subcomplex of L.

4.1.2. Divided images. An m-complex K^m is *full to dimension* i, $i \leq m$, if every j-simplex, $j \leq i$, of K^m is contained in some i-simplex of K^m.

Assume K^m is full to dimension m. An $(m-1)$-simplex of K^m is *external* if it is contained in exactly one m-simplex; otherwise, it is *internal*. The *boundary complex* of K^m, *bound*(K^m), is the subcomplex with all the faces of the external simplexes of K^m; clearly, *bound*(K^m) is full to dimension $m-1$. Abusing notation, *bound*(σ^m) denotes the $(m-1)$-faces of a simplex σ^m.

A complex K^m is an *m-pseudomanifold* if it is full to dimension m and every $(m-1)$-simplex is contained in either one or two m-simplexes.[2] An *m-manifold* is an m-pseudomanifold in which every $(m-1)$-simplex is contained in two m-simplexes; i.e., it has no external simplexes. For example, if K^3 is the complex containing a single 3-simplex σ^3 and all its faces, then *bound*(K^3) is a 2-manifold which looks like a hollow tetrahedron.

Let τ^{m-1} be an internal $(m-1)$-simplex of an m-pseudomanifold, which is contained in two m-simplexes, τ_1^m and τ_2^m. The *link* vertices of τ^{m-1} are v_1, the vertex of τ_1^m not in τ^{m-1}, and v_2, the vertex of τ_2^m not in τ^{m-2}.

The following combinatorial definition is used later (section 5.1) to cast the structure of a protocol in the topological framework. Very roughly, L^m will represent the possible input configurations, K^m the possible executions of a protocol, and $\psi(\sigma) \subseteq K^m$ will be the executions starting with input configuration σ.

[2] In topology, pseudomanifolds are assumed to have additional properties, which we do not require for our applications.

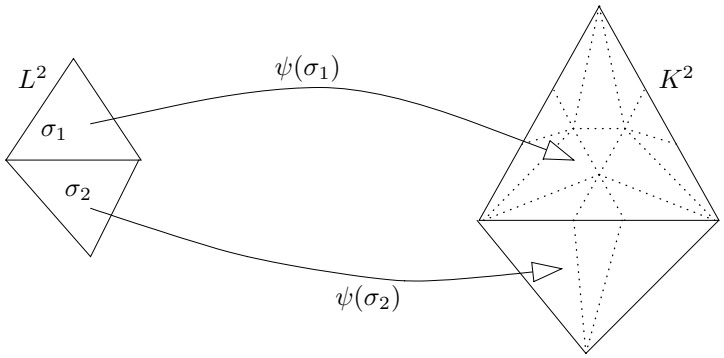


FIG. 2. *K^2 is a divided image of L^2 under ψ.*

DEFINITION 4.1. *Let L^m, K^m be complexes and ψ a function that assigns to each simplex of L^m a finite subcomplex of K^m. The complex K^m is a* divided image of L^m under ψ *if*

1. $\psi(\emptyset) = \emptyset$,
2. *for every $\tau \in K^m$ there is a simplex $\sigma \in L^m$ such that $\tau \in \psi(\sigma)$,*
3. *for every $\sigma^0 \in L^m$, $\psi(\sigma^0)$ is a single vertex,*
4. *for every $\sigma, \sigma' \in L^m$, $\psi(\sigma \cap \sigma') = \psi(\sigma) \cap \psi(\sigma')$, and*
5. *for every $\sigma \in L^m$, $\psi(\sigma)$ is a $dim(\sigma)$-pseudomanifold with $\psi(bound(\sigma)) = bound(\psi(\sigma))$.*

K^m is a divided image of L^m *if for some ψ, K^m is a divided image of L^m under ψ.*

Notice that in condition 5, $bound(\sigma)$ is a set of simplexes, while $\psi(bound(\sigma))$ is the complex which is the union of $\psi(\tau)$, over the simplexes $\tau \in bound(\sigma)$.

Intuitively, a divided image is obtained from L^m by replacing every simplex of L^m with a pseudomanifold, making sure that they "fit together" (in the sense of condition 4). In addition, condition 2 guarantees that ψ maps L^m "onto" K^m; condition 3 guarantees that ψ maps vertices of L^m to vertices of K^m; finally, condition 5 guarantees that ψ preserves the dimension and the boundary of simplexes in L^m.

Figure 2 shows a divided image of a complex containing two simplexes. In the figure, solid lines show the boundaries of σ_1 and of σ_2, and their image in K^2 under ψ.

Remark. The concept of a divided image is reminiscent of the notion of *acyclic carrier*[3] used by Munkres to study *subdivisions*, a fundamental concept of algebraic topology (cf. [24, 26]), as both associate subcomplexes of one complex to simplexes of another. However, divided images, even if they are connected, are different from subdivisions: For example, a 2-dimensional torus with a triangle removed from its surface is a divided image of a 2-simplex, since its boundary is a 1-dimensional triangle; yet, it is neither an acyclic carrier nor a subdivided simplex since it has "holes" (nontrivial homology groups).

Let $M(\sigma^m)$ be the complex consisting of a set σ^m with $m+1$ elements and all its proper subsets; $M(\sigma^m)$ is an m-pseudomanifold consisting of a single m-simplex and all its faces.

The next lemma states some simple properties of divided images.

LEMMA 4.2. *Let K^m be a divided image of L^m under ψ.*

(1) *For every σ, $\sigma' \in L^m$, if $\sigma' \subseteq \sigma$, then $\psi(\sigma') \subseteq \psi(\sigma)$.*

[3]Not to be confused with the notion of *carrier* defined later.

(2) *For every pair of j-simplexes $\sigma_1^j, \sigma_2^j \in L^m$, if $\sigma_1^j \neq \sigma_2^j$, and $\sigma_1^j \cap \sigma_2^j \neq \emptyset$, then $\psi(\sigma_1^j) \cap \psi(\sigma_2^j)$ is a pseudomanifold of dimension strictly smaller than j, i.e., dimension $|\sigma_1^j \cap \sigma_2^j| - 1$.*

(3) *For every i-simplex $\sigma^i \in L^m$, $\psi(\sigma^i)$ is a divided image of $M(\sigma^i)$ under $\psi|M(\sigma^i)$.*

(4) *A simplex $\tau^{m-1} \in K^m$ is external if and only if for some external simplex $\sigma^{m-1} \in L^m$, $\tau^{m-1} \in \psi(\sigma^{m-1})$.*

Proof. (1) By Definition 4.1(4) if $\sigma' \subseteq \sigma$, then $\psi(\sigma') = \psi(\sigma \cap \sigma') = \psi(\sigma) \cap \psi(\sigma') \subseteq \psi(\sigma)$.

(2) Let $\sigma^i = \sigma_1^j \cap \sigma_2^j$. Since $\sigma_1^j \neq \sigma_2^j$, and $\sigma_1^j \cap \sigma_2^j \neq \emptyset$, then $0 \leq i < j$. By Definition 4.1(4), $\psi(\sigma^i) = \psi(\sigma_1^j) \cap \psi(\sigma_2^j)$. By Definition 4.1(5), $\psi(\sigma^i)$ is an i-pseudomanifold.

(3) It follows from (1) that $\psi(\sigma^i) = K^i$ is a divided image of an i-simplex.

(4) A simplex $\tau^{m-1} \in K^m$ is external if and only if there is a unique τ^m containing τ^{m-1}. By Definition 4.1(2), for some σ^j, $\tau^m \in \psi(\sigma^j)$. By Definition 4.1(5) $j \geq m$. Since L^m is an m-complex, and $\sigma^j \in L^m$, $j = m$. So $\tau^{m-1} \in \mathit{bound}(\psi(\sigma^m))$. By Definition 4.1(5), this happens if and only if $\tau^{m-1} \in \psi(\mathit{bound}(\sigma^m))$, i.e., $\tau^{m-1} \in \psi(\sigma^{m-1})$, for some σ^{m-1}.

To prove that σ^{m-1} is external, assume, by way of contradiction, that there is another m-simplex, $\sigma_1^m \neq \sigma^m$, such that $\sigma^{m-1} \subseteq \sigma_1^m$. It follows from (1) that $\tau^{m-1} \in \psi(\sigma^m) \cap \psi(\sigma_1^m)$. Since $\tau^m \in \psi(\sigma^m)$, it cannot be that $\tau^m \in \psi(\sigma_1^m)$ because otherwise $\tau^m \in \psi(\sigma^m \cap \sigma_1^m)$, contradicting (2). However, since τ^{m-1} is contained in some m-simplex of $\psi(\sigma_1^m)$, it must be that such a simplex τ_1^m is different from τ^m, a contradiction to the fact that τ^{m-1} is external.

To prove the other direction, assume for some external simplex $\sigma^{m-1} \in L^m$, $\tau^{m-1} \in \psi(\sigma^{m-1})$. Thus, $\tau^{m-1} \in \psi(\mathit{bound}(\sigma_1^m))$ for some σ_1^m containing σ^{m-1}. By Definition 4.1(5) $\tau^{m-1} \in \mathit{bound}(\psi(\sigma_1^m))$, so τ^{m-1} is contained in a unique τ_1^m of $\psi(\sigma_1^m)$. If $\tau^{m-1} \in K^m$ is not external there is another simplex τ_2^m containing τ^{m-1}. As argued above, there is another simplex, σ_2^m, with $\tau_2^m \in \psi(\sigma_2^m)$. Since $\tau^{m-1} \in \tau_1^m \cap \tau_2^m$, $\tau^{m-1} \in \psi(\sigma_1^m) \cap \psi(\sigma_2^m)$, and by Definition 4.1(4) we have that $\tau^{m-1} \in \psi(\sigma_1^m \cap \sigma_2^m)$. Definition 4.1(5) implies that $\sigma_1^m \cap \sigma_2^m$ is an $(m-1)$-simplex, which is equal to σ^{m-1}, by Definition 4.1(4), a contradiction to the fact that σ^{m-1} is external. □

Of particular importance for us is the case where K^m is a divided image of $M(\sigma^m)$ under ψ. In this case, for any $\sigma \in M(\sigma^m)$, $\sigma \subseteq \sigma^m$, and by Lemma 4.2(1), $\psi(\sigma) \subseteq \psi(\sigma^m)$. Thus, $\cup_{\sigma \in M(\sigma^m)} \psi(\sigma) = \psi(\sigma^m)$. On the other hand, by condition (2), $\cup_{\sigma \in M(\sigma^m)} \psi(\sigma) = K^m$, which proves the next lemma.

LEMMA 4.3. $\psi(\sigma^m) = K^m$.

The *carrier* of a simplex $\tau \in K^m$, denoted $\mathit{carr}(\tau)$, is the simplex $\sigma \in L^m$ of smallest dimension such that $\tau \in \psi(\sigma)$, that is, the "smallest" simplex in L^m which is mapped to τ. By Definition 4.1(2), every simplex $\tau \in K^m$ is in $\psi(\sigma)$, for some $\sigma \in L^m$, and by Lemma 4.2(2) it is unique. For example, in Figure 2, the carrier of τ_1 is σ_1, while the carrier of τ_2 is σ_2.

4.1.3. Connectivity. For any j, $1 \leq j \leq m$, the *j-graph of K^m* consists of one vertex for every j-simplex of K^m, and there is an edge between two vertices if the intersection of the two corresponding simplexes is a $(j-1)$-simplex of K^m. K^m is *j-connected* if its j-graph is connected. A *j-path* in K^m is a sequence of j-simplexes such that every two consecutive simplexes share a $(j-1)$-face. Thus a j-path corresponds to a path in the j-graph.

Divided images need not be connected. For example, consider K^2 from Figure 2, take some disjoint 3-simplex σ^3, and let $K'^2 = K^2 \cup \mathit{bound}(\sigma^3)$. Then K'^2 is a divided image of L^2 under ψ', where $\psi'(\sigma_1) = \psi(\sigma_1)$, $\psi'(\sigma_2) = \psi(\sigma_2) \cup \mathit{bound}(\sigma^3)$. In Appendix A, we prove that any divided image contains a connected divided image; moreover, we later show that the specific divided images constructed in this paper are connected.

4.1.4. Colorings. A complex K is *colored* by associating a value from some set, called *colors*, with each of its vertices. A coloring is *proper* if different vertices in the same simplex have different colors. A simplicial map $f : K \to L$ is *color preserving* if, for every vertex v of K, $f(v)$ has the same color as v. If a coloring is proper and f is color preserving, then for any simplex $\{v_0, \ldots, v_k\}$ the vertices $f(v_0), \ldots, f(v_k)$ are different; i.e., $f(\sigma)$ has the same dimension as σ.

Let K^m be a divided image of L^m. A simplicial map $\chi : K^m \to L^m$ is a *Sperner coloring* if, for every $v \in K^m$, $\chi(v) \in \mathit{carr}(v)$. Intuitively, the "colors" are the vertices of L^m, and χ "folds" K^m into L^m with the requirement that each vertex of K^m goes to a vertex of its carrier. This notion generalizes the usual Sperner coloring, where L^m is an m-simplex with all its faces, and the vertices are the colors $\{0, 1, \ldots, m\}$. That is, the vertices at the "corners" of K^m are colored $0, 1, \ldots, m$, and a vertex in a boundary is colored with one of the colors of the "corners" of that boundary. The main combinatorial definition we use is the following.

Definition 4.4. *A complex K^m is a* chromatic divided image *of L^m if it is a divided image of L^m with a proper Sperner coloring χ.*

4.2. Modeling tasks. Here we cast the notion of task (introduced in section 2) in terms of combinatorial topology. Denote $\mathit{ids} = \{0, \ldots, n\}$. For some domain of values V, let $P(V)$ be the set of all pairs consisting of an id from ids and a value from V.

For a domain of inputs $\mathcal{I}$, an *input complex*, $\mathcal{I}^n$, is a complex that includes n-simplexes (i.e., subsets of $n+1$ elements) of $P(\mathcal{I})$ and all their faces such that the vertices in an n-simplex have different id fields. For a domain of outputs $\mathcal{O}$, an *output complex*, $\mathcal{O}^n$, is defined similarly over $\mathcal{O}$. That is, if (i, val) is a vertex of $\mathcal{I}^n$, then val denotes an input for process p_i, while if (i, val) is a vertex of $\mathcal{O}^n$, then val is an output for process p_i. $\mathcal{I}^n$ and $\mathcal{O}^n$ are properly colored by the id fields and are full to dimension n; each complex is also colored (not necessarily properly) by the corresponding domain of values.

Using the combinatorial topology notions, a *task* is a triple $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$; $\mathcal{I}^n$ is an input complex, $\mathcal{O}^n$ is an output complex, and Δ maps each n-simplex of $\mathcal{I}^n$ to a nonempty set of n-simplexes in $\mathcal{O}^n$. We sometimes mention only Δ when $\mathcal{I}^n$ and $\mathcal{O}^n$ are clear from the context. The simplexes in $\Delta(\sigma^n)$ are the *admissible* output simplexes for σ^n: If σ^n is an input simplex and $\tau^n \in \Delta(\sigma^n)$, then τ^n can be output when the system starts with input σ^n.

Δ can be extended to input simplexes of dimension smaller than n, since the outputs of some processes in an execution can be completed to outputs for all processes that are admissible for the inputs of the execution. Therefore, Δ maps an input simplex σ of dimension smaller than n to the faces of n-simplexes in $\Delta(\sigma^n)$ with the same dimension and ids for all input simplexes σ^n that contain σ. Extended in this manner, $\Delta(M(\sigma^n))$ is a subcomplex of $\mathcal{O}^n$.

Remark. There is another variant of wait-free solvability, where Δ is explicitly defined for input simplexes of dimension smaller than n. This can be captured in our

model by adding as part of the input a bit that tells the process whether to participate or not; nonparticipating processes output some default value.

4.3. Protocol complexes. For an execution α, the set $\{(0, \alpha|0), \ldots, (n, \alpha|n)\}$ is denoted by $views(\alpha)$. The *protocol complex* $\mathcal{P}^n$ of a protocol $\mathcal{P}$ is the complex whose n-simplexes are the sets of minimal final $views(\alpha)$, for every execution α of $\mathcal{P}$, and all their faces. $\mathcal{P}^n$ is colored with four colors—an id, an input, a view, and an output value; the ids coloring is proper. $\mathcal{P}^n$ depends on the possible interleavings of events (schedules) and on the transitions of processes and their local states.

Given an input n-simplex σ, the *protocol complex for* σ, $\mathcal{P}^n(\sigma)$, is the subcomplex of $\mathcal{P}^n$ containing all n-simplexes corresponding to executions of $\mathcal{P}$, where processes start with inputs σ, and all their faces. Intuitively, $\tau \in \mathcal{P}^n(\sigma)$ if and only if there is an execution α with inputs σ such that the views of processes in τ are the same as in α. It is possible that two executions, α and α', satisfy the above condition for a simplex of dimension n, but, in this case, α and α' are indistinguishable to all processes. If τ is a face of two n-simplexes, σ_1 and σ_2, corresponding to executions α_1 and α_2, and τ contains a vertex with id p_i, then $\alpha_1 \overset{p_i}{\sim} \alpha_2$.

Since $\mathcal{P}$ is deterministic and wait-free, every process writes an output after a finite number of steps, implying that $\mathcal{P}^n(\sigma)$ is finite.

$\mathcal{P}$ implies a *decision map* $\delta_{\mathcal{P}} : \mathcal{P}^n \to \mathcal{O}^n$ specifying the output for each final view of a process. If $\mathcal{P}$ solves Δ, then $\delta_{\mathcal{P}}$ is simplicial, since $\delta_{\mathcal{P}}(\tau)$ is an output simplex, for every $\tau \in \mathcal{P}^n$. Clearly, $\delta_{\mathcal{P}}$ preserves the ids coloring and, for every input n-simplex σ, $\delta_{\mathcal{P}}(\mathcal{P}^n(\sigma))$ is a complex.

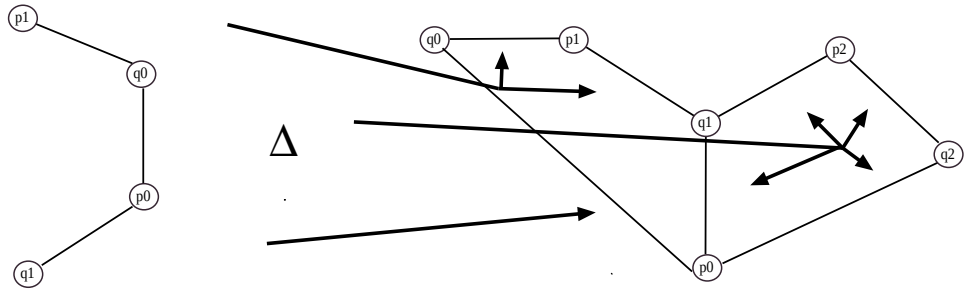
Figure 3 presents an example of a task for two processes, p and q. A vertex of the input (output) complex contains an id and an input (output) value. A task is defined by stating Δ: for input $\{(p,1),(q,0)\}$ the output can be $\{(p,1),(q,0)\}$ or $\{(p,1),(q,1)\}$; for input $\{(p,0),(q,1)\}$ the output can be $\{(p,0),(q,0)\}$; for input $\{(p,0),(q,0)\}$ the output can be $\{(p,0),(q,1)\}$, $\{(p,2),(q,1)\}$, $\{(p,2),(q,2)\}$, $\{(p,0),(q,2)\}$, or $\{(p,0),(q,1)\}$. The figure also includes a protocol complex (with all executions, not only IS executions) that solves the task. Here a vertex is labeled with an id, then an input, and then the output. For input $\{(p,1),(q,0)\}$, each process executes one read (of all the registers) and one write operation. For input $\{(p,0),(q,0)\}$, p executes one write, then one read, and then one write, while q executes one write and one read. For input $\{(p,0),(q,1)\}$, p executes one write, then one read, and then one write, while q does not execute any operation. There are various ways of assigning outputs to the processes so that the protocol solves the task.

The above definitions imply the following topological interpretation of the operational definition of a protocol solving a task (presented at the end of section 2).

PROPOSITION 4.5. *$\mathcal{P}$ solves $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ if and only if $\delta_{\mathcal{P}}(\mathcal{P}^n(\sigma)) \subseteq \Delta(M(\sigma))$ for every n-simplex $\sigma \in \mathcal{I}^n$. In this case, we say that $\delta_{\mathcal{P}}$* agrees *with Δ.*

Since the protocol depends only on the inputs (see section 2), if two input n-simplexes σ and σ' have the same inputs, i.e., differ only by a permutation of the ids, then $\mathcal{P}^n(\sigma)$ can be obtained from $\mathcal{P}^n(\sigma')$ by applying the same permutation to the ids. Therefore, the decision map must be *anonymous*; i.e., $\delta_{\mathcal{P}}(\mathcal{P}^n(\sigma))$ determines $\delta_{\mathcal{P}}(\mathcal{P}^n(\sigma'))$.

5. A condition for wait-free solvability. We prove that the subcomplex induced by IS executions is a chromatic divided image of the input complex; moreover, this subcomplex is shown to be connected and orientable. This implies our main result, a necessary condition for tasks which are solvable by a wait-free protocol (The-



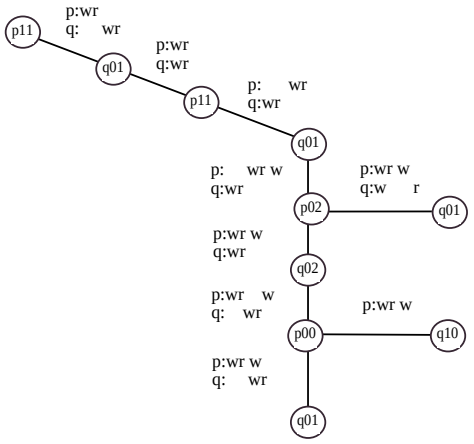


FIG. 3. *A protocol complex and a task for two processes, p and q.*

orem 5.14). In section 7, we present two impossibility results which are implied by this condition; the set consensus impossibility result follows from the divided image property only, while the renaming impossibility also relies on orientability.

5.1. The ISE complex and divided images. The *IS executions* (ISE) complex, $\mathcal{E}^n$, is the subcomplex of the protocol complex containing all IS executions; $\mathcal{E}^n(\sigma^n)$ is the subcomplex of all IS executions starting with an input simplex $\sigma^n \in \mathcal{I}^n$.

Figure 4 contains an example of an ISE complex for a single input simplex (the one on the left); each 2-simplex corresponds to an execution where each process is active in exactly one round. There are simplexes that correspond to executions α_1 and α_2 from Figure 1; indeed, the vertices that correspond to p_1 and p_2 are the same in these simplexes; i.e., p_1 and p_2 have the same views. The 1-simplexes on the boundary correspond to executions in which two processes take one step each and do not see the

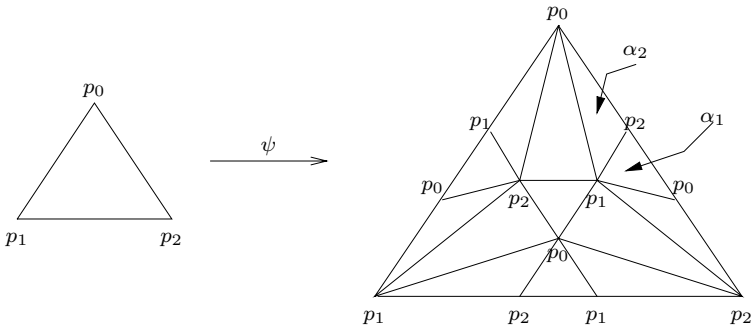


FIG. 4. *The ISE complex, when each process takes at most one step.*

third process; corner vertices correspond to "solo" executions where a process does not see other processes. The 2-simplex in the center corresponds to the IS execution consisting of one round, in which all three processes are active.

The protocol complex shown in Figure 3, including all executions, is not a divided image of the input complex, since the subcomplex for the input simplex $\{(p,0),(q,0)\}$ is not a pseudomanifold. We now show that the ISE complex of a wait-free protocol, using only read/write operations, is a chromatic divided image of the input complex; we start with some definitions.

For a simplex σ of $\mathcal{I}^n$, $\mathcal{O}^n$, or $\mathcal{E}^n$, $ids(\sigma)$ is the set of ids appearing as the first component of vertices in σ. For a simplex σ of $\mathcal{I}^n$, $inputs(\sigma)$ is σ. For a simplex σ of $\mathcal{E}^n$, $inputs(\sigma)$ is the set of pairs of ids with inputs appearing in vertices of σ (taking the input component from the state). For a simplex σ of $\mathcal{E}^n$, $views(\sigma)$ is the set of views appearing in vertices of σ and $observed(\sigma)$ is the set of ids of processes whose operations appear in $views(\sigma)$. If p_i is not in $observed(\sigma)$, then the views in σ are the same as in an execution in which p_i does not take a step; since a process always "observes itself," $ids(\sigma) \subseteq observed(\sigma)$.

We define a function ψ that assigns a subcomplex of $\mathcal{E}^n$ to each input simplex $\sigma \in \mathcal{I}^n$; intuitively, $\psi(\sigma)$ contains all the executions with input σ, in which only processes in $ids(\sigma)$ are observed taking steps.

DEFINITION 5.1. *For $\sigma \in \mathcal{I}^n$, let $\psi(\sigma)$ be the complex containing all simplexes $\tau \in \mathcal{E}^n$, such that $inputs(\tau) = inputs(\sigma)$ and $observed(\tau) = ids(\sigma)$, and all their faces.*

Clearly, $\psi(\sigma)$ is full to dimension $dim(\sigma)$. The next lemma is useful below.

LEMMA 5.2. *For any $\tau \in \mathcal{E}^n$ and $\sigma \in \mathcal{I}^n$, $\tau \in \psi(\sigma)$ if and only if $inputs(\tau) \subseteq inputs(\sigma)$ and $observed(\tau) \subseteq ids(\sigma)$.*

Proof. Assume τ is in $\psi(\sigma)$. If τ is of the same dimension as σ, then the claim is immediate from the definition. So assume τ is a face of τ' which is of the same dimension as σ. Thus, $inputs(\tau) \subseteq inputs(\tau')$ and $observed(\tau) \subseteq observed(\tau')$; the definition of ψ implies that $inputs(\tau) \subseteq inputs(\sigma)$ and $observed(\tau) \subseteq ids(\sigma)$.

To prove the converse direction, assume $inputs(\tau) \subseteq inputs(\sigma)$ and $observed(\tau) \subseteq ids(\sigma)$. Since the protocol is wait-free, the processes in $ids(\sigma) - ids(\tau)$ (if there are any) should be able to run on their own, after all processes in $ids(\tau)$ have obtained the views in τ, and decide. This implies that there is an execution in which all processes in $ids(\sigma) - ids(\tau)$ (if any) observe only processes in $ids(\sigma)$, and processes in $ids(\tau)$ have the same views as in τ. Let π be the simplex in $\mathcal{E}^n$ that corresponds to this execution. Note that $inputs(\pi) = inputs(\sigma)$ and $observed(\pi) = ids(\sigma)$. Therefore, $\pi \in \psi(\sigma)$. Since τ is a face of π, the claim follows. ∎

We proceed to prove that Definition 4.4 holds, and hence that the conditions of Definition 4.1 hold. Clearly, process ids are a proper Sperner coloring of $\mathcal{E}^n$. Condition 1 is trivial.

LEMMA 5.3. *Condition* 2 *of Definition* 4.1 *holds.*

Proof. Consider a simplex $\tau \in \mathcal{E}^n$. Let $\tau^n \in \mathcal{E}^n$ be such that $\tau \subseteq \tau^n$. Then there is a simplex $\sigma^n \in \mathcal{I}^n$ with $inputs(\tau^n) = inputs(\sigma^n)$. Since $ids(\tau^n) \subseteq observed(\tau^n)$, then $observed(\tau^n) = ids(\sigma^n)$ and $\tau^n \in \psi(\sigma^n)$. Since τ is a face of τ^n, $\tau \in \psi(\sigma^n)$. ∎

Since the protocol is deterministic, we have the next lemma.

LEMMA 5.4. *Condition* 3 *of Definition* 4.1 *holds.*

By Lemma 5.2, $\tau \in \psi(\sigma \cap \sigma')$ if and only if $inputs(\tau) \subseteq inputs(\sigma \cap \sigma') = inputs(\sigma) \cap inputs(\sigma')$ and $observed(\tau) \subseteq ids(\sigma \cap \sigma') = ids(\sigma) \cap ids(\sigma')$. This happens if and only if $\tau \in \psi(\sigma) \cap \psi(\sigma')$, implying the next lemma.

LEMMA 5.5. *Condition* 4 *of Definition* 4.1 *holds.*

By Lemma 3.3, if a process p_j is unseen in an execution, then there is no other execution that differs only in p_j's view. By Lemma 3.4, if p_i is seen in an execution, then there is another execution that differs only in p_j's view. This implies the next lemma.

LEMMA 5.6. *Let τ_1^i be an i-simplex of $\psi(\sigma^i)$ corresponding to an IS execution α, and $j \in ids(\tau_1^i)$. Process p_j is unseen in α by processes with $ids(\sigma^i)$, if and only if there is no other i-simplex $\tau_2^i \in \psi(\sigma^i)$, that differs only in the vertex colored with p_j.*

Since the boundary corresponds to executions in which some processes are unseen, the above lemma can be used to prove that ψ preserves the boundary.

LEMMA 5.7. *For every simplex $\sigma^i \in \mathcal{I}^n$, $\psi(bound(\sigma^i)) = bound(\psi(\sigma^i))$.*

Proof. To prove that $\psi(bound(\sigma^i)) \subseteq bound(\psi(\sigma^i))$, consider $\tau \in \psi(bound(\sigma^i))$. Then $\tau \in \psi(\sigma^{i-1})$ for some face σ^{i-1} of σ^i. Since $\psi(\sigma^{i-1})$ is full to dimension $i-1$, there is some $\tau^{i-1} \in \psi(\sigma^{i-1})$ such that $\tau \subseteq \tau^{i-1}$. By the definition of ψ, $\tau^{i-1} \in \psi(\sigma^i)$. Since $\psi(\sigma^i)$ is full to dimension i, there is some $\tau^i \in \psi(\sigma^i)$ such that $\tau^{i-1} \subset \tau^i$. Since $\tau^{i-1} \in \psi(\sigma^{i-1})$, it follows that $observed(\tau^{i-1}) = ids(\sigma^{i-1})$. Let p_j be the process id in $ids(\sigma^i) - ids(\sigma^{i-1})$. Note that $j \notin observed(\tau^{i-1})$; that is, no process sees a step by p_j in τ^{i-1}. Furthermore, in τ^i, p_j sees only steps by processes in $ids(\sigma^i)$. Since the protocol is deterministic, there is only one possible view for p_j. Namely, τ^{i-1} is contained in a single i-simplex τ^i, and hence $\tau^{i-1} \in bound(\psi(\sigma^i))$. Since τ is a face of τ^{i-1} and ψ is closed under inclusion, $\tau \in bound(\psi(\sigma^i))$.

To prove that $\psi(bound(\sigma^i)) \supseteq bound(\psi(\sigma^i))$, consider some $\tau \in bound(\psi(\sigma^i))$. Since $bound(\psi(\sigma^i))$ is full to dimension $i-1$, τ is a face of some $\tau^{i-1} \in bound(\psi(\sigma^i))$. Since τ^{i-1} is external in $\psi(\sigma^i)$, it is a face of a single $\tau^i \in \psi(\sigma^i)$. Assume that $ids(\tau^i) - ids(\tau^{i-1}) = \{p_j\}$; then, by Lemma 5.6, p_j is unseen in the execution corresponding to τ^i, and $p_j \notin observed(\tau^{i-1})$. Let σ^{i-1} be the face of σ^i that does not include the vertex whose id is p_j; clearly, $observed(\tau^{i-1}) = ids(\sigma^{i-1})$. By the definition of ψ, $\tau^{i-1} \in \psi(\sigma^{i-1})$; thus, $\tau^{i-1} \in \psi(bound(\sigma^i))$, implying that $\tau \in \psi(bound(\sigma^i))$. ∎

We have shown in Lemma 3.4 that the uncertainty about another process is restricted to its last seen round, if it is seen. Thus, once we fix the views of all processes but one, the remaining process may have only one of two views, which translates into the next lemma.

LEMMA 5.8. *Let τ_1^i be an i-simplex of $\psi(\sigma)$, corresponding to an IS execution α, such that $ids(\sigma) = ids(\tau_1^i)$ and $j \in ids(\tau_1^i)$. If p_j is seen in α, then there is exactly one other i-simplex $\tau_2^i \in \psi(\sigma)$ that differs only in p_j's view.*

Lemma 5.7 implies that if p_j is seen in an IS execution α, then α corresponds to an internal simplex. We now prove that an internal $(i-1)$-simplex is contained in

exactly two i-simplexes, showing that the ISE complex is a pseudomanifold.

LEMMA 5.9. *For every simplex $\sigma^i \in \mathcal{I}^n$, $\psi(\sigma^i)$ is an i-pseudomanifold.*

Proof. As noted before, $\psi(\sigma^i)$ is full to dimension i. We show that any simplex $\tau^{i-1} \in \psi(\sigma^i)$ is contained in at most two i-simplexes. Let $\tau^i \in \psi(\sigma^i)$ be such that τ^{i-1} is a face of τ^i. Since τ^{i-1} and τ^i are properly colored by the ids, $ids(\tau^i) - ids(\tau^{i-1}) = \{j\}$ for some process p_j. Furthermore, any i-simplex of $\psi(\sigma^i)$ containing τ^{i-1} has a vertex colored with p_j. Let α be the prefix of an execution with steps by processes in $ids(\sigma^i)$ corresponding to τ^i; this prefix exists since $observed(\tau^i) = ids(\sigma^i)$. There are two cases.

Case 1. p_j is unseen in α. Let σ^{i-1} be the face of σ^i that does not include the vertex whose id is p_j; then $observed(\tau^{i-1}) = ids(\sigma^{i-1})$. Since p_j does not see an id not in $ids(\sigma^i)$, its view is determined. Hence, τ^i is unique.

Case 2. p_j is seen in α. Let k be the last seen round of p_j in α. Lemma 5.8 implies that there are only two possible views for p_j which are compatible with the views in τ^{i-1}. □

Lemmas 5.7 and 5.9 imply condition 5 of Definition 4.1. Therefore, $\mathcal{E}^n$ is a chromatic divided image of $\mathcal{I}^n$ under ψ. If a protocol $\mathcal{P}$ solves a task Δ, then it solves Δ in IS executions. Clearly, $\delta_{\mathcal{P}}$ is a color-preserving (on process ids), anonymous simplicial map from $\mathcal{E}^n$ to $\mathcal{O}^n$ that agrees with Δ. This implies the next theorem.

THEOREM 5.10. *Let $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ be a task. If there is a wait-free protocol which solves this task, then there is a chromatic divided image of $\mathcal{I}^n$, K^n, and there is a color-preserving (on ids), anonymous simplicial map δ from K^n to $\mathcal{O}^n$ that agrees with Δ.*

5.2. Connectivity of the ISE complex. Since the ISE complex is a divided image of the input simplex, it contains a connected divided image of the input simplex (see Appendix A). Translating Theorem 3.9 to combinatorial topology implies the following, stronger property of the ISE complex—it is connected itself.

A divided image of a simplex σ^n under ψ is *well connected* if for every $\sigma^i \in M(\sigma^n)$, if $i \geq 1$, then $\psi(\sigma^i)$ is i-connected, and if $i \geq 2$, then $bound(\psi(\sigma^i))$ is $(i-1)$-connected.

THEOREM 5.11. *Let $\mathcal{E}^n$ be an ISE chromatic divided image of $M(\sigma^n)$ under ψ starting with input σ^n. Then $\mathcal{E}^n$ is well connected.*

The theorem can be extended to show that if $\mathcal{I}^n$ is i-connected, then so is $\psi(\mathcal{I}^n)$. The proof follows from the fact that if σ_1^i and σ_2^i are two simplexes of $\mathcal{I}^n$ with $\sigma^{i-1} = \sigma_1^i \cap \sigma_2^i$, then $\psi(\sigma_1^i) \cap \psi(\sigma_2^i)$ contains an $(i-1)$-simplex, τ^{i-1}, by Lemma 4.2(2). Thus, there is an i-path from any i-simplex in $\psi(\sigma_1^i)$ to any simplex in $\psi(\sigma_2^i)$ going through τ^{i-1}. Transitivity implies the claim for i-simplexes of $\mathcal{I}^n$ connected by an i-path.

5.3. Orientability of the ISE complex. Let K^m be an m-pseudomanifold. An *orientation* of a simplex is an equivalence class of orderings of its vertices, consisting of one particular ordering and all even permutations of it. If the vertices are colored with process ids, then the *positive* orientation is the one in which the vertices are ordered with the ids from small to large, and the *negative* orientation is the one where the two vertices with smallest ids are exchanged (each one with all its even permutations). Let $\sigma^{(i)}$ be the face of σ^m in which the vertex with id i is removed; e.g., $\sigma^{(1)}$ is the face with ids $\{0, 2, \ldots, m\}$. An orientation of an m-simplex σ induces an orientation on each of its faces, $\sigma^{(i)}$, according to the sign of $(-1)^i$. Consider an m-simplex σ and its $(m-1)$-face, $\sigma^{(i)}$. If σ is oriented positive, with its ids in order, $\langle 0, 1, 2, \ldots, m \rangle$, σ induces the orientation $(-1)^i$ to the face $\sigma^{(i)}$. On the other hand, if σ is oriented negative, $\langle 1, 0, 2, \ldots, m \rangle = -\langle 0, 1, 2, \ldots, m \rangle$, and σ induces the

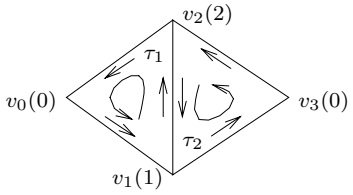


FIG. 5. *An oriented 2-pseudomanifold, with a coloring (in brackets).*

orientation $(-1)(-1)^i$ to the face $\sigma^{(i)}$. For example, in Figure 5, τ_1 is positively oriented $\langle v_0, v_1, v_2\rangle$, and the induced orientations are $\langle v_0, v_1\rangle$, $\langle v_1, v_2\rangle$, and $\langle v_2, v_0\rangle$; τ_2 is negatively oriented $\langle v_2, v_1, v_3\rangle$, and the induced orientations are $\langle v_1, v_3\rangle$, $\langle v_3, v_2\rangle$, and $\langle v_2, v_1\rangle$.

K^m is *orientable* if there is an orientation for each of its m-simplexes such that an $(m-1)$-simplex contained in two m-simplexes gets opposite induced orientations from each of the m-simplexes. K^m together with such an orientation is an *oriented* pseudomanifold. For example, the complex in Figure 5 is an oriented 2-pseudomanifold.

An m-pseudomanifold is *chromatic* if it is properly colored with $m+1$ colors. The next simple lemma (e.g., [5]) shows that, for chromatic pseudomanifolds, the previous (common) definition of orientability is equivalent to a simpler combinatorial definition.

LEMMA 5.12. *A chromatic pseudomanifold K^m is orientable if and only if its m-simplexes can be partitioned into two disjoint classes such that if two m-simplexes share an $(m-1)$-face, then they belong to different classes.*

Proof. Assume K^m is orientable and partition its simplexes into two classes, one containing the simplexes with positive orientations and the other containing the simplexes with negative orientations. Since K^m is orientable, if two m-simplexes τ_1 and τ_2 share an $(m-1)$-face, then the shared face gets opposite induced orientations. Since K^m is chromatic, this implies that τ_1 and τ_2 have opposite orientations, and they are in different classes.

To prove the converse direction, consider the partition of the simplexes of a chromatic pseudomanifold K^m into two disjoint classes; orient the simplexes of the first class in the positive direction and the simplexes of the second class in the negative direction. Consider two m-simplexes σ_1 and σ_2 that share an $(m-1)$-face, $\sigma^{(i)}$. By assumption, σ_1 and σ_2 are in different classes; without loss of generality, σ_1 is oriented positive and σ_2 is oriented negative, Since K^m is chromatic, σ_1 is oriented $\langle 0, 1, 2, \ldots, m\rangle$ and σ_2 is oriented $\langle 1, 0, 2, \ldots, m\rangle = -\langle 0, 1, 2, \ldots, m\rangle$. Hence, σ_1 induces the orientation $(-1)^i$ to the shared face $\sigma^{(i)}$, while σ_2 induces the orientation $(-1)(-1)^i$; that is, they induce opposite orientations. □

A chromatic divided image of $M(\sigma^m)$ under ψ is *orientable* if $\psi(\sigma)$ is orientable for every $\sigma \in M(\sigma^m)$.

THEOREM 5.13. *Let $\mathcal{E}^n$ be an ISE chromatic divided image of $M(\sigma^n)$ under ψ starting with input σ^n. Then $\mathcal{E}^n$ is orientable.*

Proof. Let σ^i be a face of σ^m; we explicitly partition the i-simplexes of $\psi(\sigma^i)$ to two disjoint classes. Consider an i-simplex $\tau \in \psi(\sigma^i)$ and the IS execution α corresponding to τ. For any process p_j that is seen in α, we can write $\alpha = s_1, \ldots, s_r, \{p_j\}^*$, where r is the tail position of p_j in α. The tail $\{p_j\}^*$ is nonempty for at most one seen process p_j. If the tail is not empty for some seen process p_j, then $\hat{r}(\sigma^i)$ is p_j's tail position. Otherwise, then the tail position r is the same for all seen processes

and is equal to the number of concurrency classes in α; in this case, $\hat{r}(\sigma^i) = r$. The simplex τ is in positive if $\hat{r}(\tau)$ is even; otherwise, it is in negative.

Consider two i-simplexes, τ_1^i and τ_2^i, sharing an $(i-1)$-face, τ^{i-1}; the link vertices of τ^{i-1} are colored with the same id, say p_j. By Lemma 5.9 and its proof, without loss of generality, p_j is in the kth concurrency class of τ_1^i and the kth concurrency class of τ_2^i is $\{p_j\}$, where k is the last seen round of p_j in τ_1^i. The views of all processes but p_j are the same in τ_1^i and in τ_2^i; p_j may have a tail with a different number of steps in τ_1^i and τ_2^i, but these are not counted in $\hat{r}(\tau_1^i)$ and $\hat{r}(\tau_2^i)$. Therefore, the number of concurrency classes in τ_1^i without its tail has the opposite parity of the number of concurrency classes in τ_2^i without its tail, and the simplexes are in different classes. □

Putting together Theorems 5.10, 5.11, and 5.13, we have our main result.

THEOREM 5.14 (main characterization). *Let $\langle \mathcal{I}^n, \mathcal{O}^n, \Delta \rangle$ be a task. If there is a wait-free protocol which solves this task, then there is a well connected, orientable, chromatic divided image of $\mathcal{I}^n$, and there is a color-preserving (on ids), anonymous simplicial map δ from it to $\mathcal{O}^n$ that agrees with Δ.*

Herlihy and Rajsbaum [20] introduced algebraic spans as a tool for proving impossibilities of renaming and set consensus using algebraic techniques. In Appendix B, we show that an orientable chromatic divided image induces an algebraic span.

6. Counting simplexes. In this section we present two theorems about the number of certain simplexes in a chromatic divided image. The first is the well-known Sperner's lemma, which will be applied in section 7.1 for the set consensus impossibility. The second is a theorem about the number of monochromatic simplexes in a binary coloring of a connected, orientable chromatic divided image; this theorem is used in section 7.2 to show a lower bound on renaming.

Recall the definition of a Sperner's coloring from section 4.1.4. The following is a restatement of Sperner's lemma in our notation; it holds for both oriented and nonorientable divided images. It will be used to prove the set consensus impossibility in section 7.1, and hence this result does not depend on orientability. It is a well-known result, and the proof is a simple parity counting argument (e.g., [4]).

THEOREM 6.1 (Sperner's lemma). *Consider a divided image K^m of $M(\sigma^m)$ under ψ and a Sperner coloring $\chi : K^m \to M(\sigma^m)$. There is an odd number of simplexes $\tau \in K^m$ with $\chi(\tau) = \sigma^m$; in particular, at least one simplex goes to σ^m.*

We proceed to the second result, which will be applied to prove the renaming impossibility.

Let K^m be an orientable, chromatic divided image of σ^m under ψ, with proper coloring *id*. Assume in addition that K^m has a binary coloring b, with colors $\{0, 1\}$. Fix an orientation of K^m and an induced orientation on its boundary.

Let $\#\text{mono}(K^m)$ be the number of monochromatic m-simplexes of K^m, counted by orientation; i.e., an m-simplex with all its vertices colored by b with the same color is counted as $+1$ if it is positively oriented; otherwise, it is counted as -1. For example, consider a complex K^m with two m-simplexes, as in Figure 5, with all vertices colored 1. In this case, both simplexes are monochromatic; the count is 0, since the simplexes have opposite orientations, and hence one is counted $+1$ and the other is counted -1.

The theorem applies to divided images with structural symmetry; i.e., any pair of divided images of faces (of the same dimension) are isomorphic under a bijection that maps the process ids consistently. Formally, a mapping $f : L^m \to K^m$ is *id consistent* if, for any two vertices $v_1, v_2 \in L^m$, $id(v_1) = id(v_2)$ implies $id(f(v_1)) = id(f(v_2))$. A

divided image of σ under ψ has *structural symmetry* if, for any two i-faces, σ_1^i and σ_2^i, of σ, there is an id consistent simplicial bijection (one-to-one mapping), ζ, between the vertices of $\psi(\sigma_1^i)$ and the vertices of $\psi(\sigma_2^i)$.

A binary coloring, b, of K^m with structural symmetry ζ is *symmetric* if isomorphic vertices have the same color; that is, $b(\zeta(v)) = b(v)$ for every vertex $v \in bound(K^m)$. Implicitly, if K^m has a symmetric binary coloring, then it has structural symmetry. Notice that ζ need not be unique, and a binary coloring is symmetric relative to a particular choice of ζ. It is not trivial to prove that a coloring is symmetric; this is done when the theorem is applied (section 7.2).

The main property used to prove the renaming impossibility has a flavor similar to Sperner's lemma, but it assumes that the divided image is orientable and symmetric. This result was proved implicitly in [20], based on the existence of algebraic spans. Here we state the result in our notation and show that divided images are sufficient to obtain the required algebraic spans. Thus, the proof uses a result of [20] that is stated in algebraic terms.[4] The basic algebraic formalisms needed appear in Appendix B.

THEOREM 6.2. *Assume K^m is a chromatic divided image of σ^m under ψ, with a symmetric binary coloring b. If K^m is connected and oriented, then $\#mono(K^m) \neq 0$.*

Proof. The proof is similar to the proof of Theorem 6.2 in [20]. The coloring of K^m defined by $c(v) = (b(v) + id(v)) \bmod (m+1)$, for every v, induces a symmetric map from K^m to $M(\sigma^m)$, and hence a corresponding symmetric chain map. Theorem B.2 implies that there is a chain map from $M(\sigma^m)$ to K^m. The composition of these chain maps is the required chain map from $M(\sigma^m)$ to $M(\sigma^m)$, and hence the result follows from Lemma 6.1 in [20]. □

7. Applications. In this section, we apply the condition for wait-free solvability presented earlier (Theorem 5.10) to derive lower bounds for k-set consensus and renaming. The renaming lower bound relies on Theorem 6.2 and, therefore, on the fact that the ISE complex is orientable.

7.1. *k*-set consensus. In the *k-set consensus* task [11], processes start with inputs from some domain and have to decide on at most k different outputs; each output has to be the input of some process.

Captured in our combinatorial topology language, the k-set consensus task is the triple $\langle \mathcal{D}^n, \mathcal{D}^n, \Delta \rangle$. $\mathcal{D}^n$ is $P(\mathcal{D})$, for some domain $\mathcal{D}$, and Δ maps each $\sigma^n \in \mathcal{D}^n$ to the subset of n-simplexes in $\mathcal{D}^n$ that contain at most k different values from the values in σ^n. Theorem 5.10 and Sperner's lemma are used to prove the next lower bound, which is tight: If $k > n$, then there is a simple wait-free protocol for k-set consensus [11].

THEOREM 7.1. *If $k \leq n$, then there does not exists a wait-free protocol that solves the k-set consensus task.*

Proof. Assume, by way of contradiction, that $\mathcal{P}$ is a wait-free protocol for the k-set consensus task, $k \leq n$. We show that $\mathcal{P}$ must have at least one execution in which $k+1$ different values are output, which implies that $\mathcal{P}$ does not solve the k-set consensus task correctly. This is done by applying Theorem 6.1 to the ISE complex, which is a chromatic divided image of the input complex, by Theorem 5.10. Let $\delta_{\mathcal{P}}$ be the decision map implied by $\mathcal{P}$. Note that $\delta_{\mathcal{P}}$ can also be seen as a simplicial map into $\mathcal{I}^n$.

[4] The combinatorial proof in [3] had a mistake.

Consider the input simplex σ of dimension $k+1$, where processes $p_0, \ldots, p_k$ have different inputs $v_0, \ldots, v_k$, respectively. By Theorem 5.10, $\mathcal{E}^n(\sigma)$ is a divided image of σ. Let v be the value $\delta_{\mathcal{P}}$ assigns to some vertex in $\mathcal{E}(\sigma)$, and let σ' be its carrier in $M(\sigma)$. The nontriviality condition implies that v is the input of some process in σ', and therefore we get the following.

CLAIM 7.2. *$\delta_{\mathcal{P}}$ is a Sperner coloring of $\mathcal{E}^n(\sigma)$.*

By Theorem 6.1, there is a simplex $\tau \in \mathcal{E}^n(\sigma)$ with $\delta_{\mathcal{P}}(\tau) = \sigma$. Thus, at least $k+1$ values are output in some IS execution with input σ, which is a contradiction. □

7.2. Renaming. In the *renaming* task [2], processes start with inputs (*original name*) from a large domain and are required to decide on distinct outputs (*new names*) from a domain which is as small as possible. Clearly, the task is trivial if processes can access their id; in this case, process p_i decides on i, which yields the smallest possible domain. To avoid trivial solutions, it is required that the processes and the protocol are *anonymous* [2]. That is, process p_i with original name x executes the same protocol as process p_j with original name x.

Captured in our combinatorial topology language, the M-renaming task is the triple $\langle \mathcal{D}^n, M^n, \Delta \rangle$. $\mathcal{D}^n$ contains all subsets of some domain $\mathcal{D}$ (of original names) with different values, M^n contains all subsets of $[0, M]$ (of new names) with different values, and Δ maps each $\sigma^n \in \mathcal{D}^n$ to all n-simplexes of M^n. We use Theorems 5.10 and 6.2 to prove that there is no wait-free anonymous protocol for the M-renaming task if $M \leq 2n-1$. The bound is tight, since there is an anonymous wait-free protocol for $2n$-renaming [2].

THEOREM 7.3. *If $M < 2n$, then there is no anonymous wait-free protocol that solves the M-renaming task.*

Proof. Assume, by way of contradiction, that $\mathcal{P}$ is a wait-free protocol for the M-renaming task, $M \leq 2n-1$. Assume that $\mathcal{D} = [0, 2n]$; i.e., assume that the original names are only between 0 and $2n$. Since we consider a bounded set of inputs, we can assume, without loss of generality, that every process takes the same number of steps. Moreover, by an observation of Eli Gafni (see Herlihy [18]), we may assume, without loss of generality, that $\mathcal{P}$ is comparison-based. That is, the protocol produces the same outputs on inputs which are order-equivalent.

By Theorem 5.10, the ISE complex, $\mathcal{S}$, is a chromatic divided image of the input complex $\mathcal{D}^n$. By Theorem 5.13, $\mathcal{S}$ is orientable, and by Theorem 5.11 $\mathcal{S}$ is connected. Let ζ be the *order preserving* mapping between any two subsets of process ids, of the same size, i.e., the mapping that maps the smallest id in one set to the smallest id in the second set, etc. It is obvious that $\mathcal{S}$ has structural symmetry under ζ, since we consider only executions in which all processes take the same number of steps.

Let $\delta_{\mathcal{P}}$ be the decision map implied by $\mathcal{P}$ and define δ' to be the binary coloring which colors each vertex in $\mathcal{S}$ with the parity of the value assigned to it by $\delta_{\mathcal{P}}$. Fix an input simplex σ^n. Since the protocol is comparison-based and anonymous, δ' is a symmetric coloring of $\mathcal{S}(\sigma^n)$ (with respect to ζ). Therefore, the assumptions of Theorem 6.2 are satisfied, and, therefore, at least one m-simplex of $\mathcal{S}(\sigma^n)$ is monochromatic under δ'.

On the other hand, note that the domain $[0, 2n-1]$ does not include $n+1$ different odd names; similarly, the domain $[0, 2n-1]$ does not include $n+1$ different even names. This implies that δ' cannot color any simplex of $\mathcal{S}$ with all zeroes or with all ones; i.e., no simplex of $\mathcal{S}$ is monochromatic, which is a contradiction. □

8. Discussion. This paper presents a study of wait-free solvability based on combinatorial topology. Informally, we have defined the notion of a chromatic divided

image and proved that a necessary condition for wait-free solvability is the existence of a simplicial chromatic mapping from a divided image of the inputs to the outputs that agrees with the task specification. In fact, we show the existence of a specific, well-structured chromatic divided image, the ISE complex; the ISE complex is proved to be orientable and connected, which allowed us to prove theorems about its combinatorial properties. These properties were then used to derive tight lower bounds for k-set consensus and renaming. Our results do not use homology groups, whose computation may be complicated.

Borowsky and Gafni [9] introduce another model of computation, based on an iterated version of ISs, and prove that it is equivalent to the wait-free model, when only decision tasks are concerned. In this model, there is a sequence of IS objects, each used at most once by each process: in its i-step, a process writes its entire state to the ith object. It is easy to check that the iterated ISE complex is a divided image of the input complex; the proof is completely analogous to the proof that the ISE complex is a divided image of the input complex. Therefore, the characterization and impossibility results of this paper hold for this model as well. This exemplifies how our combinatorial framework can be useful for other models of computation.

Several questions remain open. First, it is of interest to find other applications to the necessary condition presented here. Second, there are several directions to extend our framework, e.g., to allow fewer than n failures (as was done for one failure in [6]), to handle other primitive objects besides read/write registers (cf. [19, 10]), and to incorporate ongoing tasks.

Appendix A. Connectivity of divided images. It is useful to note the following lemma.

LEMMA A.1. *If K^m is full to dimension j and j-connected, then it is i-connected for every i, $0 < i < j$.*

Proof. Fix some i, $0 < i < j$, and consider two i-simplexes, σ_1^i and σ_2^i in K^m. Since K^m is full to dimension j, and $i < j$, it follows that there are two j-simplexes σ_1^j and σ_2^j in K^m such that $\sigma_1^i \subset \sigma_1^j$ and $\sigma_2^i \subset \sigma_2^j$. Since K^m is j-connected, it follows that there is a j-path in K^m from σ_1^j to σ_2^j. This path can be used to obtain an i-path in K^m from σ_1^i to σ_2^i. □

We now prove the following theorem.

THEOREM A.2. *Let K^m be a divided image of L^m under ψ. There exists a complex H^m, $H^m \subseteq K^m$, and $\tilde{\psi}$, a restriction of ψ to H^m, such that H^m is a divided image of L^m under $\tilde{\psi}$, and $\tilde{\psi}(\sigma^i)$ is i-connected. Moreover, bound$(\tilde{\psi}(\sigma^i))$ is $(i-1)$-connected for every $\sigma^i \in L^m$, $i > 1$.*

Proof. We construct H^m by defining $\tilde{\psi}(\sigma^i)$ for every $\sigma^i \in L^m$, by induction on i. We show that the properties of Definition 4.1 hold and that $\tilde{\psi}(\sigma^i)$ is i-connected.

For $i = 0$, for every $\sigma^0 \in L^m$, let $\tilde{\psi}(\sigma^0) = \psi(\sigma^0)$.

Assume $\tilde{\psi}(\sigma^j)$ has been defined for $j < i$, satisfying the required properties. Let $\tilde{\psi}(\sigma^i)$ be the complex containing every face of every $\tau^i \in \psi(\sigma^i)$ such that there exists an i-path from τ^i to some simplex with an $(i-1)$-face in $\tilde{\psi}(bound(\sigma^i))$.

First notice that $\tilde{\psi}(\sigma^i)$ is an i-pseudomanifold, since $\tilde{\psi}(\sigma^i) \subseteq \psi(\sigma^i)$, and it is full to dimension i. It is also simple to see it has properties 1, 2, and 3 of Definition 4.1. Next, we prove condition 4 of Definition 4.1.

CLAIM A.3. $\tilde{\psi}(\sigma \cap \sigma') = \tilde{\psi}(\sigma) \cap \tilde{\psi}(\sigma')$.

Proof. By definition of $\tilde{\psi}$ if $\sigma^{i-1} \subseteq \sigma^i$, then $\tilde{\psi}(\sigma^{i-1}) \subseteq \tilde{\psi}(\sigma^i)$. Hence, for every face σ^j of σ^i, $\tilde{\psi}(\sigma^j) \subseteq \tilde{\psi}(\sigma^i)$. This implies that $\tilde{\psi}(\sigma \cap \sigma') \subseteq \tilde{\psi}(\sigma) \cap \tilde{\psi}(\sigma')$. Assume $\tau^i \in \tilde{\psi}(\sigma) \cap \tilde{\psi}(\sigma')$. Thus $\tau^i \in \psi(\sigma) \cap \psi(\sigma')$ and $\tau^i \in \psi(\sigma \cap \sigma')$. This implies that there exists an i-path from τ^i to a simplex containing some $\tau^{i-1} \in \tilde{\psi}(\mathit{bound}(\sigma \cap \sigma'))$. ∎

Next, we prove condition 5 of Definition 4.1.

CLAIM A.4. $\tilde{\psi}(\mathit{bound}(\sigma^i)) = \mathit{bound}(\tilde{\psi}(\sigma^i))$.

Proof. Assume $\tau^{i-1} \in \tilde{\psi}(\mathit{bound}(\sigma^i))$. Then $\tau^{i-1} \in \tilde{\psi}(\sigma^{i-1})$ for some face σ^{i-1} of σ^i. Thus, $\tau^{i-1} \in \psi(\sigma^{i-1})$ and $\tau^{i-1} \in \psi(\mathit{bound}(\sigma^i)) = \mathit{bound}(\psi(\sigma^i))$. It follows that there exists a unique $\tau^i \in \psi(\sigma^i)$ containing τ^{i-1}. This implies that $\tau^{i-1} \in \mathit{bound}(\tilde{\psi}(\sigma^i))$.

Before proving the other direction of the claim, notice that $\mathit{bound}(\tilde{\psi}(\sigma^i)) \subseteq \mathit{bound}(\psi(\sigma^i))$. Otherwise, if $\tau^{i-1} \in \mathit{bound}(\tilde{\psi}(\sigma^i))$, then there exists a unique $\tau^i \in \tilde{\psi}(\sigma^i)$ containing τ^{i-1}. If there is another i-simplex $\tau \in \psi(\sigma^i)$ containing τ^{i-1}, τ would also be in $\tilde{\psi}(\sigma^i)$, by definition of $\tilde{\psi}$, since τ and τ^i are connected by a (one step) i-path.

Assume that $\tau^{i-1} \in \mathit{bound}(\tilde{\psi}(\sigma^i))$, and let τ^i be the unique i-simplex in $\tilde{\psi}(\sigma^i)$ containing τ^{i-1}. Then $\tau^{i-1} \in \mathit{bound}(\psi(\sigma^i))$, since we have just proved that $\mathit{bound}(\tilde{\psi}(\sigma^i)) \subseteq \mathit{bound}(\psi(\sigma^i))$. Hence $\tau^{i-1} \in \psi(\mathit{bound}(\sigma^i))$, and thus $\tau^{i-1} \in \psi(\sigma^{i-1})$, for some face σ^{i-1} of σ^i. Since $\tau^i \in \tilde{\psi}(\sigma^i)$, there exists an i-path from τ^i to an i-simplex containing some $\tau_1^{i-1} \in \tilde{\psi}(\mathit{bound}(\sigma^i))$. Assume $\tau_1^{i-1} \in \tilde{\psi}(\sigma_1^{i-1})$. Thus there exists an $(i-1)$-path from τ^{i-1} to τ_1^{i-1}, by using this i-path. By the induction hypothesis, there exists an $(i-1)$-path from τ_1^{i-1} to an $(i-1)$-simplex containing some $\tau^{i-2} \in \tilde{\psi}(\mathit{bound}(\sigma_1^{i-1}))$. Therefore there exists an $(i-1)$-path from τ^{i-1} to the $(i-1)$-simplex containing τ^{i-2}. The definition of $\tilde{\psi}$ implies that $\tau^{i-1} \in \tilde{\psi}(\sigma_1^{i-1})$, and thus $\tau^{i-1} \in \tilde{\psi}(\mathit{bound}(\sigma^i))$. ∎

To complete the proof of the lemma, we show, by induction on i, that $\tilde{\psi}(\sigma^i)$ is i-connected, and that, for $i > 1$, $\tilde{\psi}(\mathit{bound}(\sigma^i))$ is $(i-1)$-connected.

To show that $\tilde{\psi}(\mathit{bound}(\sigma^2))$ is 1-connected, observe that $\tilde{\psi}(\sigma^1)$ is 1-connected because, since it is finite, it is a graph which consists of a simple path. The lemma follows since Definition 4.1(4) holds.

For the induction step, we first show that $\tilde{\psi}(\sigma^i)$ is i-connected. By the induction hypothesis, the boundary of $\tilde{\psi}(\sigma^i)$ is $(i-1)$-connected. Thus, it suffices to consider two adjacent $(i-1)$-simplexes τ^{i-1}, τ'^{i-1} of $\mathit{bound}(\tilde{\psi}(\sigma^i))$, that is, such that $\tau^{i-1} \cap \tau'^{i-1}$ is an $(i-2)$-simplex, τ^{i-2}. In this case, we need to show that τ^i is i-connected to τ'^i, where these are the single i-simplexes of $\tilde{\psi}(\sigma^i)$ with $\tau^{i-1} \subset \tau^i$ and $\tau'^{i-1} \subset \tau'^i$.

To prove that τ^i is i-connected to τ'^i, we construct an i-path $(\tau^i =)\tau_0^i, \tau_1^i, \ldots, \tau_\ell^i (= \tau'^i)$, all of whose simplexes contain τ^{i-2}. Denote the pairwise intersections $\tau_1^{i-1}, \tau_2^{i-2}, \ldots, \tau_\ell^{i-1}$, and $\tau_0^{i-1} = \tau^{i-1}$.

Notice that there is a unique $(i-1)$-face of τ_0^i, τ_1^{i-1} that (a) contains τ^{i-2} and (b) is different from τ_0^{i-1}. Now, τ_1^{i-1} is not in $\mathit{bound}(\tilde{\psi}(\sigma^i))$ because $\mathit{bound}(\tilde{\psi}(\sigma^i))$ is a pseudomanifold, and hence τ^{i-2} is contained in only two $(i-1)$-simplexes, τ^{i-1} and τ'^{i-1}. It follows that τ_1^{i-1} is contained in two i-simplexes, $\tau^i (= \tau_0^i)$, and τ_1^i.

Repeating this argument, we obtain an i-path $(\tau^i =)\tau_0^i, \tau_1^i, \ldots, \tau_\ell^i$. This path either ends in τ'^i or it repeats a simplex and the construction is stopped at that point. We now show that it can end only in τ'^i, and hence $\tau'^i = \tau_\ell^i$. Consider $\tau_{\ell+1}^{i-1}$, the unique $(i-1)$-face of τ_ℓ^i, that satisfies conditions (a) and (b) above, i.e., that contains τ^{i-2}, and is different from τ_ℓ^{i-1}. We have seen that this face cannot be external (unless it is equal to τ'^{i-1}, $\tau_\ell^i = \tau'^i$, and then we are done) because $\mathit{bound}(\tilde{\psi}(\sigma^i))$ is a pseudomanifold, and hence τ^{i-2} is contained in only two $(i-1)$-simplexes, τ^{i-1}, τ'^{i-1}.

Thus the path must have a loop. That is, the face $\tau_{\ell+1}^{i-1}$ is contained in τ_ℓ^i, and in some other previous simplex in the i-path, and the next simplex after τ_ℓ^i in the i-path is a previous simplex. However, this is impossible, since each previous $(i-1)$-simplex satisfying conditions (a) and (b) in the i-path already is contained in two i-simplexes, or else is τ^{i-1} which is an external simplex.

The proof that $\tilde{\psi}(\mathit{bound}(\sigma^{i+1}))$ is i-connected follows from the fact that $\tilde{\psi}(\sigma^i)$ is i-connected: it suffices to prove that for any $\sigma_1, \sigma_2 \in \mathit{bound}(\sigma^{i+1})$, $\tilde{\psi}(\sigma_1) \cap \tilde{\psi}(\sigma 2)$ contains an $(i-1)$-simplex. This follows from Definition 4.1(4) because $\tilde{\psi}(\sigma_1) \cap \tilde{\psi}(\sigma 2) = \tilde{\psi}(\sigma_1 \cap \sigma_2)$, and from Definition 4.1(5) because $\tilde{\psi}(\sigma_1 \cap \sigma_2)$ is an $(i-1)$-pseudomanifold. □

Appendix B. Algebraic spans. We start by describing very briefly some algebraic preliminaries; for more details and examples see [24, section 1.13] and [20].

Let K be an n-dimensional simplicial complex and $\sigma = (v_0, \ldots, v_q)$ a q-simplex of K. An *orientation* for σ is an equivalence class of orderings on $v_0, \ldots, v_q$, consisting of one particular ordering and all even permutations of it. For example, an orientation of a 1-simplex (v_0, v_1) is just a direction, either from v_0 to v_1 or vice-versa. An orientation of a 2-simplex (v_0, v_1, v_2) can be either "clockwise," as in (v_0, v_1, v_2), or "counterclockwise," as in (v_0, v_2, v_1).

Recall that, by Lemma 5.12, a chromatic pseudomanifold K^m is orientable if and only if its m-simplexes can be partitioned into two disjoint classes such that if two m-simplexes share an $(m-1)$-face, then they belong to different classes.

A *q-chain* of K is a formal sum of oriented q-simplexes: $\sum_{i=0}^{\ell} \lambda_i \cdot \sigma_i^q$, where λ_i is an integer. When writing chains, we typically omit q-simplexes with zero coefficients, unless they are all zero, when we simply write 0. We write $1 \cdot \sigma^q$ as σ^q and $-1 \cdot \sigma^q$ as $-\sigma^q$. We identify $-\sigma^q$ with σ^q having the opposite orientation. The q-chains of K (with component-wise addition) form a free Abelian group $C_q(K)$, called the qth *chain group* of K. For dimension -1, we adjoin the infinite cyclic group Z in dimension -1, $C_{-1}(K) = Z$.

A *boundary* operator $\partial_q : C_q(K) \to C_{q-1}(K)$ is a homomorphism such that, for a q-chain α,

$$\partial_{q-1}\partial_q \alpha = 0,$$

and the *augmentation* $\partial_0 : C_0(K) \to C_{-1}(K)$ is an epimorphism (i.e., a surjective homomorphism).

For an oriented simplex $\sigma^q = (v_0, \ldots, v_q)$, let $\mathrm{face}_i(\sigma^q)$, the ith *face* of σ^q, be the $(q-1)$-simplex $(v_0, \ldots, \hat{v}_i, \ldots, v_q)$, where circumflex denotes omission. The usual boundary operator $\partial_q : C_q(K) \to C_{q-1}(K)$, $q > 0$, is defined on simplexes:

$$\partial \sigma^q = \sum_{i=0}^{q} (-1)^i \cdot \mathrm{face}_i(\sigma^q),$$

and is extended additively to chains: $\partial(\alpha_0 + \alpha_1) = \partial\alpha_0 + \partial\alpha_1$. For $q = 0$, $\partial_0(v) = 1$, and extend linearly.[5] (We sometimes omit subscripts from boundary operators.)

The *chain complex* $C(K)$ is the sequence of groups and homomorphisms $\{C_q(K), \partial_q\}$.

[5] Munkres [24] uses ϵ for ∂_0.

Let $C(K) = \{C_q(K), \partial_q\}$ and $C(L) = \{C_q(L), \partial'_q\}$ be chain complexes for simplicial complexes K and L. An *augmentation-preserving chain map* (or chain map) ϕ is a family of homomorphisms.

$$\phi_q : C_q(K) \to C_q(L)$$

such that $\partial'_q \circ \phi_q = \phi_{q-1} \circ \partial_q$.

Let $S^n = M(\sigma^n)$, for a simplex σ^n, and let $\mathcal{P}^n$ be a divided image of S^n under ψ. An *algebraic span* is a chain map $\phi : C(S^n) \to C(\mathcal{P}^n)$ that is carried by ψ. The chain map ϕ is *carried* by ψ if each simplex appearing with a nonzero coefficient in $\phi(\sigma^i)$ is in the subcomplex $\psi(\sigma^i)$.

A properly colored m-pseudomanifold K^m is *orientable* if its m-simplexes can be oriented so that two m-simplexes sharing an $(m-1)$-face give opposite orientations to the face. That is, the chain $\partial(\sigma_1 + \sigma_2)$ has zero coefficient on the face that σ_1 and σ_2 share. Such an orientation is called *coherent,* and partitions the m-simplexes in two classes, so that simplexes sharing an $(m-1)$-face are in different classes. The orientation of simplexes of dimension other than m can be arbitrary. Also, if K^m is orientable, then it has exactly two coherent orientations.

The next proposition follows directly from the definition of pseudomanifold and of orientability. Indeed, in an m-pseudomanifold every $(m-1)$-simplex is contained in either one or two m-simplexes. For any $(m-1)$-simplex σ contained in two m-simplexes σ_1, σ_2, the chain $\partial(\sigma_1 + \sigma_2)$ has zero coefficient on σ.

PROPOSITION B.1. *Let K^m be an m-pseudomanifold with a coherent orientation and c the chain which is the sum of all its m-simplexes. Then $\partial(c)$ is a chain of $(m-1)$-simplexes, which as a set is equal to bound(K^m).*

In the renaming applications we assume a divided image with structural symmetry, in order to use Theorem 6.2. In such cases we assume that the corresponding algebraic span preserves the symmetry, that is, that symmetric simplexes have associated symmetric simplexes under the chain map. Let $\mathcal{E}^n$ be the ISE complex on input σ^n, a divided image of σ^n under ψ, where each process executes a fixed number of steps.

THEOREM B.2. *There is an algebraic span from $M(\sigma^n)$ to $\mathcal{E}^n$. Moreover, if $\mathcal{E}^n$ has structural symmetry, then the algebraic span preserves it.*

Proof. Give an arbitrary orientation to the simplexes of $M(\sigma^n)$ to obtain a chain complex $C(M(\sigma^n))$.

By Theorem 5.10 we know that $\mathcal{E}^n$ is a chromatic divided image of $M(\sigma^n)$ under ψ. By Theorem 5.13 we can give to every pseudomanifold $\psi(\sigma)$ a coherent orientation and obtain the chain complex $C(\mathcal{E}^n)$.

Let $\dot{\psi}$ be the chain map that associates to $\sigma^i \in M(\sigma^n)$, the chain which is the sum of the i-simplexes that appear in $\psi(\sigma^i)$, with sign defined as follows: $\dot{\psi}(\sigma^n)$ is the sum of the n-simplexes in $\psi(\sigma^n)$. If $\dot{\psi}(\sigma^i)$ has been defined, then $\dot{\psi}(\text{face}_j(\sigma^i))$ is the sum of the $(i-1)$-simplexes in $\psi(\text{face}_j(\sigma^i))$ with sign $(-1)^j$.

We proceed to prove that $\dot{\psi}$ is a chain map. First, $\dot{\psi}_q$ is a well-defined homomorphism, since it is defined on simplexes (the generators of the group $C_q(M(\sigma^n))$) and extended linearly. Now, $\dot{\psi}_q(\sigma^q) = c$ satisfies the hypothesis of Proposition B.1, and hence $\partial'_q(c)$ is equal to *bound*$(\psi(\sigma^q))$, as a set. Since $\partial_q(\sigma^q)$ is equal to $\sum_{i=0}^{q}(-1)^i \cdot \text{face}_i(\sigma^q)$, and equal to *bound*$(\sigma^q)$ as a set, and by the last requirement of the definition of a divided image, we have that $\partial'_q \circ \phi_q = \phi_{q-1} \circ \partial_q$, since the orientations correspond. Notice that with $q = 0$ we use condition 3 of Definition 4.1. ∎

Acknowledgments. We thank Javier Bracho, Eli Gafni, John Havlicek, Maurice Herlihy, Guy Rey, Nir Shavit, Ronit Teplixke, and Mark Tuttle for comments on the paper and very useful discussions. We especially thank an anonymous referee for many comments that allowed us to correct errors and improve the overall presentation.

REFERENCES

[1] H. Attiya, N. Lynch, and N. Shavit, *Are wait-free algorithms fast?*, J. ACM, 41 (1994), pp. 725–763.
[2] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk, *Renaming in an asynchronous environment*, J. ACM, 37 (1990), pp. 524–548.
[3] H. Attiya and S. Rajsbaum, *The combinatorial structure of wait-free solvable tasks*, in Proceedings of the 10th International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 1151, O. Babaoglu and K. Marzullo, eds., Springer-Verlag, Berlin, 1996, pp. 321–343.
[4] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications,* Elsevier, New York, 1976.
[5] J. Bracho and L. Montejano, *The combinatorics of colored triangulations of manifolds*, Geom. Dedicata, 22 (1987), pp. 303–328.
[6] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed* 1*-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.
[7] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for* t*-resilient asynchronous computations*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, 1993, pp. 91–100.
[8] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 41–51.
[9] E. Borowsky and E. Gafni, *A simple algorithmically reasoned characterization of wait-free computations*, in Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, pp. 189–198.
[10] E. Borowsky and E. Gafni, *The Implication of the Borowsky-Gafni Simulation on the Set Consensus Hierarchy*, Technical report 930021, Computer Science Department, UCLA, Los Angeles, CA, 1993.
[11] S. Chaudhuri, *More choices allow more faults: Set consensus problems in totally asynchronous systems*, Inform. and Comput., 105 (1993), pp. 132–158.
[12] S. Chaudhuri, M. Herlihy, N. Lynch, and M. Tuttle, *A tight lower bound for* k*-set agreement*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 206–217.
[13] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl, *Reaching approximate agreement in the presence of faults*, J. ACM, 33 (1986), pp. 499–516.
[14] D. Dolev and H. R. Strong, *Authenticated algorithms for Byzantine agreement*, SIAM J. Comput., 12 (1983), pp. 656–666.
[15] M. Fischer, N. Lynch, and M Paterson, *Impossibility of distributed commit with one faulty process*, J. ACM, 32 (1985), pp. 374–382.
[16] E. Gafni and E. Koutsoupias, *Three-processor tasks are undecidable*, SIAM J. Comput., 28 (1999), pp. 970–983.
[17] M. Henle, *A Combinatorial Introduction to Topology,* Dover, New York, 1994.
[18] M. Herlihy, *A Tutorial on Algebraic Topology and Distributed Computation*, manuscript, 1994.
[19] M. Herlihy and S. Rajsbaum, *Set consensus using arbitrary objects* in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, 1994, pp. 324–333.
[20] M. Herlihy and S. Rajsbaum, *Algebraic spans*, Math. Structures Comput. Sci., 10 (2000), pp. 549–573.
[21] M. Herlihy and S. Rajsbaum, *On the decidability of distributed decision tasks*, in Proceedings of the 29th ACM Symposium on Theory of Computation, El Paso, TX, 1997, pp. 111–120.
[22] M. Herlihy and N. Shavit, *The asynchronous computability theorem for* t*-resilient tasks*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 111–120.
[23] M. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free compu-*

tation, in Proceedings of the 1994 ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 1994, pp. 243–252; full version of [22] and [23] appeared in J. ACM, 46 (1999), pp. 858–923.

[24] J. R. Munkres, *Elements of Algebraic Topology*, Addison-Wesley, Menlo Park, CA, 1993.

[25] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 101–110.

[26] E. H. Spanier, *Algebraic Topology*, Springer-Verlag, New York, 1966.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1315–1338

RECOGNIZING DART-FREE PERFECT GRAPHS*

V. CHVÁTAL†, J. FONLUPT‡, L. SUN§, AND A. ZEMIRLINE¶

Abstract. A graph G is called a *Berge graph* if neither G nor its complement contains a chordless cycle whose length is odd and at least five; what we call a *dart* is the graph with vertices u, v, w, x, y and edges uv, vw, uy, vy, wy, xy; a graph is called *dart-free* if it has no induced subgraph isomorphic to the dart. We present a polynomial-time algorithm to recognize dart-free Berge graphs; this algorithm uses as a subroutine the polynomial-time algorithm for recognizing claw-free Berge graphs designed previously by Chvátal and Sbihi [*J. Combin. Theory Ser. B*, 44 (1988), pp. 154–176].

Key words. perfect graphs

AMS subject classifications. 05C17, 05C75, 68W05

PII. S0097539799354771

1. Introduction.

Claude Berge proposed the notion of a *perfect graph*, which is a graph G such that, for each induced subgraph F of G, the chromatic number of F equals the largest number of pairwise adjacent vertices in F; he made the following conjecture.

CONJECTURE 1.1 (The Strong Perfect Graph Conjecture). *A graph is perfect if and only if it contains, as an induced subgraph, no odd hole chordless cycle whose number of vertices is odd and at least five and no complement of such a cycle.*

Berge publicized the Strong Perfect Graph Conjecture as early as April 1960—at the Second International Meeting on Graph Theory, organized by Horst Sachs at the University of Halle-Wittenberg—but published it only three years later [1]; the first widely available reference to it is [2]. For an account of the early history of the conjecture, see [3, 4].

A *hole* is a chordless cycle with at least four vertices; an *antihole* is the complement of a hole; holes and antiholes are called *even* or *odd* according to the parity of their number of vertices. Following Chvátal and Sbihi [6], we shall call a graph a *Berge graph* if it contains no odd hole and no odd antihole: in these terms, the Strong Perfect Graph Conjecture asserts that a graph is perfect if and only if it is a Berge graph. Its "only if" part is trivial and its "if" part remains open.

Progress towards proving the Strong Perfect Graph Conjecture is often made by proving that all graphs in some restricted class of Berge graphs are perfect. A popular way of creating restricted classes of Berge graphs for this purpose is to forbid a single induced subgraph F: the resulting class consists of all Berge graphs that are *F-free* in the sense of containing no induced subgraph isomorphic to F. In particular, with a *claw* defined as the first graph in Figure 1.1, Parathasarathy and Ravindra [10]

*Received by the editors March 24, 1999; accepted for publication (in revised form) June 1, 2000; published electronically May 29, 2002. An extended abstract of this paper has appeared in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 2000, pp. 50–53.

http://www.siam.org/journals/sicomp/31-5/35477.html

†Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019 (chvatal@cs.rutgers.edu).

‡Équipe Combinatoire, Université Pierre et Marie Curie, U.F.R. 921, 75252 Paris Cedex 05, France (Jean.Fonlupt@ecp6.jussieu.fr).

§The author is deceased.

¶Département d'Informatique, Université de Bretagne Occidentale, BP 809, 29285 Brest Cedex, France (Abdallah.Zemirline@univ-brest.fr).

proved that all claw-free Berge graphs are perfect; with a *diamond* defined as the second graph in Figure 1.1, Tucker [12] proved that all diamond-free Berge graphs are perfect; Sun [11] strengthened both of these results by proving that, with a *dart* defined as the third graph in Figure 1.1, all dart-free Berge graphs are perfect.

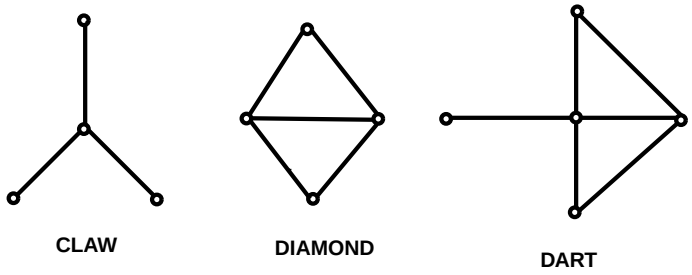


FIG. 1.1. *The claw, the diamond, and the dart.*

Another fundamental conjecture concerning perfect graphs is the following.

CONJECTURE 1.2. *There is a polynomial-time algorithm to recognize perfect graphs.*

Again, progress towards proving this conjecture is often made by designing a polynomial-time algorithm to recognize members of some restricted class of perfect graphs. In particular, Chvátal and Sbihi [6] designed a polynomial-time algorithm to recognize claw-free perfect graphs; Fonlupt and Zemirline [8, 9] designed a polynomial-time algorithm to recognize diamond-free perfect graphs; the subject of the present paper is a polynomial-time algorithm to recognize dart-free Berge graphs.

For a recent survey of results on perfect graphs and related subjects, see [5].

2. Theorems. Many theorems elucidate the structure of objects in some class $\mathcal{C}$ in terms of some proper subclass $\mathcal{C}_0$ of $\mathcal{C}$: they assert that, unless an object in $\mathcal{C}$ is primitive in the sense of belonging to $\mathcal{C}_0$, it must have a structural fault of a prescribed type. We follow this paradigm in three iterations.

First, let us call a graph *friendly* if, for each of its vertices x such that x is the center of a claw, the subgraph of G induced by the neighbors of x consists of vertex-disjoint cliques. Trivially, every friendly graph is dart-free; however, the converse is false (see either of the two graphs in Figure 2.1).

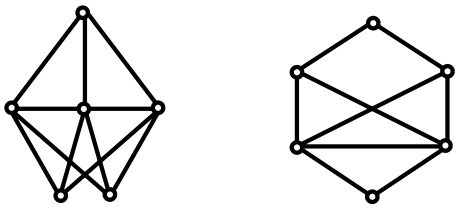


FIG. 2.1. *Two unfriendly dart-free graphs.*

Our first theorem specifies structural faults appearing in every dart-free graph that is not friendly. One of these faults is the presence of *adjacent twins*, meaning two adjacent vertices such that no vertex distinct from both is adjacent to precisely one of them.

THEOREM 2.1. *Unless a dart-free graph is friendly, it has at least one of the following properties:*

(i) *it is disconnected;*
(ii) *its complement is disconnected;*
(iii) *it contains adjacent twins.*

Next, let us call a graph a *bat* if it consists of a chordless path $a_1a_2 \dots a_m$ and an additional vertex z that is adjacent to a_1, a_i, a_{i+1}, a_m for some i with $3 \leq i \leq m-3$ and adjacent to no other a_j. Occasionally, we shall refer to z as the *head* of the bat and to a_1, a_m as its *wing-tips*; see Figure 2.2.

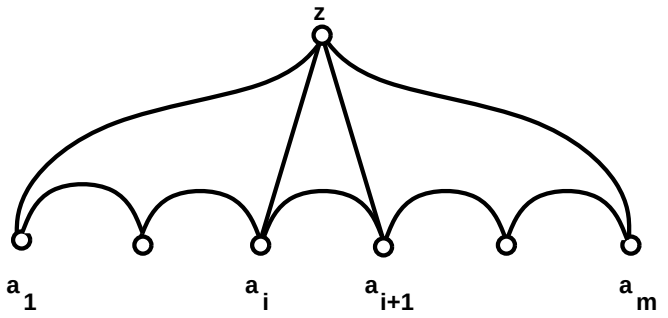


FIG. 2.2. *A bat.*

There are infinitely many bats; we call a graph *bat-free* if it contains none of them (as an induced subgraph). Our second theorem specifies structural faults appearing in every friendly Berge graph that is not bat-free. One of these faults is a clique-cutset; to describe the other fault, we need a few definitions. By a *z-edge*, we mean any edge whose endpoints are both adjacent to a vertex z; for any graph G and any vertex z of G, we let $G*z$ denote the graph obtained from $G - z$ by removing all the z-edges; we say that G has a *rosette* centered at z if $G*z$ is disconnected and the subgraph of G induced by all the neighbors of z consists of vertex-disjoint cliques.

THEOREM 2.2. *Every friendly graph containing no odd hole has at least one of the following properties:*

(i) *it is bat-free;*
(ii) *it has a clique-cutset that is a maximal clique;*
(iii) *it has a rosette.*

Any of the three graphs in Figure 2.4 shows that property (i) in this theorem cannot be dropped; the two graphs in Figure 2.3 show that neither (ii) nor (iii) can be dropped.

The two notions of a clique-cutset and a rosette may appear unrelated in terms of G; in terms of a certain graph which we call the *clique graph of* G, they are nearly dual to each other. The clique graph of G is bipartite; its white vertices are the vertices of G and its red vertices are the maximal cliques in G (here, as usual, "maximal" is meant with respect to set-inclusion rather than size); a white vertex z is adjacent to a red vertex C if and only if $z \in C$. Trivially, the removal of a red vertex C and all its neighbors disconnects the clique graph of G if and only if $G - C$ is disconnected; trivially, the removal of a white vertex z and all its neighbors disconnects the clique graph of G if and only if $G*z$ is disconnected.

Theorem 2.2 is closely related to the following theorem of Conforti and Rao [7]: *If a bipartite graph H containing no cycle of length four and no chordless cycle of*

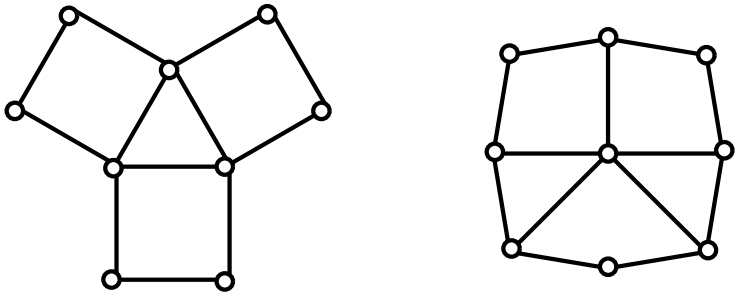


FIG. 2.3. *Neither* (ii) *nor* (iii) *in Theorem* 2.2 *can be dropped.*

length congruent to 2 *modulo* 4 *contains a cycle of length congruent to* 2 *modulo* 4*, then* H *has a vertex* v *such that the removal of* v *and all its neighbors disconnects* H*.* We are going to sketch a derivation of the Conforti–Rao theorem from a weaker form of Theorem 2.2, where "friendly" is replaced by "diamond-free."

Given a bipartite graph H that satisfies the hypothesis of the Conforti–Rao theorem, color the vertices in one part of H red, color the vertices in the other part of H white, and consider the graph G defined as follows: the vertices of G are the white vertices of H and two vertices of G are adjacent if and only if they have a common neighbor in H. Since H contains no chordless cycle of length six, H is the clique graph of G; since H contains no cycle of length four, G is diamond-free; since H contains no chordless cycle of length congruent to 2 modulo 4, G contains no odd hole. The shortest cycle of length congruent to 2 modulo 4 in H has precisely one chord (cf. Lemma 3.1 of Conforti and Rao [7]) and induces a bat in G. Now Theorem 2.2 (with "friendly" replaced by "diamond-free") guarantees that H has a vertex v such that the removal of v and all its neighbors disconnects H.

Our third theorem specifies a structural fault appearing in every friendly Berge graph that is neither bipartite nor claw-free: by a *separator* in a graph G, we mean any cutset consisting of at most two vertices except cutsets S such that S consists of two nonadjacent vertices, $G - S$ has precisely two components, and one of these components consists of a single vertex.

THEOREM 2.3. *Every bat-free friendly graph containing no odd hole has at least one of the following properties:*

(i) *it is bipartite;*
(ii) *it is claw-free;*
(iii) *it has a separator.*

Again, none of the three properties in this theorem can be dropped: see the three graphs in Figure 2.4.

Our polynomial-time algorithm to recognize friendly Berge graphs evolves from Theorems 2.1, 2.2, and 2.3; in analyzing it, we shall use a strengthening of Theorem 2.2. There, a bat in a friendly graph G is called *fragile* if, with C standing for the unique maximal clique of G that contains the triangle of the bat, the two wing-tips of the bat belong to distinct components of $G - C$.

THEOREM 2.4. *Let* G *be a friendly graph containing no odd hole. If* G *contains a bat with head* z *such that the two wing-tips of the bat belong to the same component of* $G * z$*, then* G *and* z *have at least one of the following properties:*

(i) G *contains a fragile bat with head* z*;*
(ii) G *contains a clique-cutset* C *such that* $z \in C$ *and some component of* $G - C$

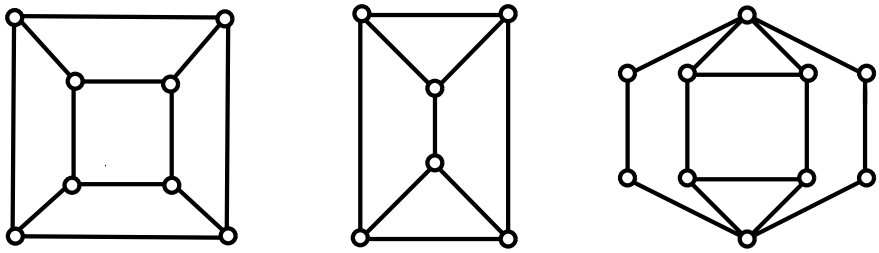


FIG. 2.4. *None of the three properties in Theorem* 2.3 *can be dropped.*

includes no neighbor of z.

Neither of the two properties in this theorem can be dropped: see the two graphs in Figure 2.5.

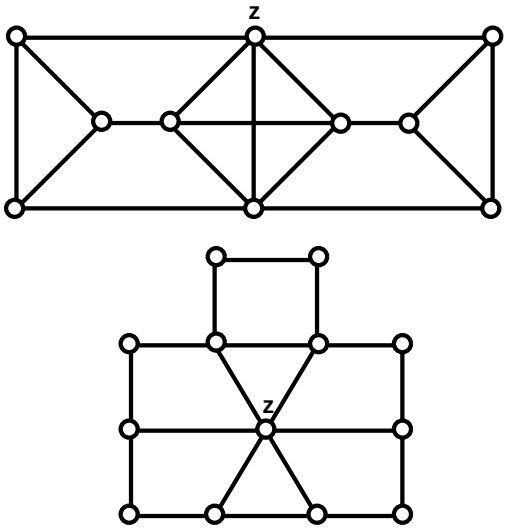


FIG. 2.5. *Neither of the two properties in Theorem* 2.3 *can be dropped.*

To derive Theorem 2.2 from Theorem 2.4, consider an arbitrary friendly graph G containing no odd hole. We may assume that G contains a bat (else it has property (i) of Theorem 2.2) and that, with z standing for the head of the bat, the two wing-tips of the bat belong to the same component of $G*z$ (else G has property (iii) of Theorem 2.2). Now G has property (i) of Theorem 2.4 or else it has property (ii) of Theorem 2.4. In the former case, the unique maximal clique of G that contains the triangle of the bat is a cutset of G; in the latter case, the unique maximal clique of G that contains C is a cutset of G.

Theorems 2.1, 2.3, and 2.4 will be proved in section 3; the algorithm and its analysis are the subject of section 4.

3. Proofs. All our subgraphs are induced. When A is a path with vertices $a_1, a_2, \ldots, a_m$ and edges $a_1a_2, a_2a_3, \ldots, a_{m-1}a_m$, we write $A = a_1a_2 \ldots a_m$ and borrow

the notation used for intervals: $A[a_i, a_j] = a_i a_{i+1} \dots a_j$, $A[a_i, a_j) = a_i a_{i+1} \dots a_{j-1}$, etc. Any path of the form $a_1 a_2 \dots a_k$ will be called a *prefix* of A and any path of the form $a_k a_{k+1} \dots a_m$ will be called a *suffix* of A.

3.1. Proof of Theorem 2.1. Consider an arbitrary graph G such that G is not friendly, G is connected, its complement $\overline{G}$ is connected, and G contains no adjacent twins. We shall find a dart in G.

It is a routine matter to verify that a graph is not friendly if and only if it contains (as an induced subgraph) either a dart or else the graph with vertices u_1, u_2, u_3, v_1, v_2 and the seven edges v_1v_2, u_iv_j $(i = 1, 2, 3; j = 1, 2)$. In particular, we may assume that our G contains an induced subgraph of the second kind. Now consider the subgraph of G induced by all the common neighbors of u_1, u_2, and u_3; let Q denote the component of this subgraph that contains v_1 and v_2.

CASE 1. *There are vertices x, w_1, w_2 such that $x \notin Q, w_1 \in Q, w_2 \in Q$, and $xw_1 \in E, xw_2 \notin E$.*

Since Q is connected, we may assume that $w_1w_2 \in E$; since $x \notin Q$, we have $xu_i \notin E$ for at least one i. Now the subgraph of G induced by $u_1, u_2, u_3, w_1, w_2, x$ contains an induced dart.

CASE 2. *The set of vertices of $G - Q$ partitions into sets A and B such that $xw \in E$ whenever $x \in A, w \in Q$ and such that $xw \notin E$ whenever $x \in B, w \in Q$.*

Since $\overline{G}$ is connected, we have $B \neq \emptyset$; in turn, since G is connected, there is an edge ab with $a \in A, b \in B$. Since v_1 and v_2 are not adjacent twins, some vertex v is adjacent to precisely one of them; note that $v \in Q$. Now a, b, v, v_1, v_2 induce a dart in G. □

3.2. Two simple lemmas.

LEMMA 3.1. *Let G be a graph with at least four vertices and with a unique triangle $v_1v_2v_3$. If all three graphs $G - \{v_1, v_2\}$, $G - \{v_1, v_3\}$, $G - \{v_2, v_3\}$ are connected, then G contains an odd hole.*

Proof. Let Q be any component of $G - \{v_1, v_2, v_3\}$. We may assume that Q is bipartite (else Q contains an odd hole and we are done); thus the set of vertices of Q splits into stable sets S_1 and S_2. Since each $G - \{v_i, v_j\}$ is connected, each of the three vertices v_k must have a neighbor in $S_1 \cup S_2$. Hence two of the three vertices, say v_1 and v_2, must have a neighbor in the same S_i. Since Q is connected, it follows that the subgraph of G induced by $Q \cup \{v_1, v_2\}$ is not bipartite, and so it contains an odd hole. □

LEMMA 3.2. *Let an edge e of a graph G extend into no triangle. If G is friendly, then $G-e$ is friendly; if G contains no odd hole, then $G-e$ contains no odd hole.* □

3.3. Proof of Theorem 2.3. By a *pseudobat*, we shall mean any graph with vertices $a_1, a_2, \dots, a_m$ and z such that

(i) $a_1a_2 \dots a_m$ is a (not necessarily chordless) path;

(ii) z is adjacent to a_1, a_i, a_{i+1} and a_m for some i with $3 \leq i \leq m - 3$ (and possibly to other vertices a_j);

(iii) $za_1a_ia_m$ is a claw.

(In a friendly graph, condition (iii) is equivalent to saying that both $za_1a_ia_m$ and $za_1a_{i+1}a_m$ are claws.)

LEMMA 3.3. *Let G be a friendly graph containing no odd hole; let z be a vertex of G that belongs to no triangle; let $u_1u_2 \dots u_r$, $v_1v_2 \dots v_s$, and $w_1w_2 \dots w_t$ be vertex-disjoint paths in $G - z$ such that u_1, v_1, and w_1 are neighbors of z and $u_rv_sw_t$ is a triangle. Then G contains an induced pseudobat.*

Proof. The proof is by induction on the number of vertices of G. The induction hypothesis allows us to assume that the three paths are chordless and that together they cover all the vertices of $G - z$. Lemma 3.1 guarantees that G contains a triangle T other than $u_r v_s w_t$. Trivially, T meets at least two of the three chordless paths; the induction hypothesis allows us to assume that T meets at most two of the paths. Hence symmetry allows us to assume that T is $v_j w_k w_{k+1}$ for some j and k; in turn, the induction hypothesis allows us to assume that $j = s$ (consider $G - v_s$).

CASE 1. $v_s w_{t-1} \in E$.

Since G is friendly, at least two of the three vertices u_r, v_{s-1}, w_{t-1} must be adjacent (consider the neighborhood of v_s), and so we are done by the induction hypothesis applied to $G - w_t$.

CASE 2. $v_s w_{t-1} \notin E$.

Write $v_0 = z$. The induction hypothesis allows us to assume that v_{s-1} is adjacent neither to w_k nor to w_t (consider $G - w_{t-1}$). Thus the path $v_{s-1} \dots v_2 v_1 v_0 w_1 w_2 \dots w_t$ along with the additional vertex v_s induces a pseudobat in G. □

LEMMA 3.4. *Let G be a friendly graph containing no odd hole. If G contains an induced pseudobat, then it contains an induced bat.*

Proof. The proof is by induction on the number of edges of G. Let $a_1, a_2, \dots, a_m$ and z be as in our definition of a pseudobat. Note that $a_1 a_{i+1} \notin E$ and $a_{i+1} a_m \notin E$ since G is friendly (consider the neighborhood of z). The induction hypothesis allows us to assume that G is the pseudobat and that the two paths $a_1 a_2 \dots a_i$ and $a_{i+1} a_{i+2} \dots a_m$ are chordless.

CASE 1. *z is adjacent to some a_j other than a_1, a_i, a_{i+1}, a_m.*

Symmetry allows us to assume that $2 \le j \le i - 1$. If $j = i - 1$, then $a_{i-1} a_{i+1} \in E, a_1 a_{i-1} \notin E, a_{i-1} a_m \notin E$ (consider the neighborhood of z), and we are done by the induction hypothesis applied to $G - a_i$. Thus we may assume that $j \le i - 2$; in particular, $a_j a_i \notin E$ (since the path $a_1 a_2 \dots a_i$ is chordless). If $a_j a_m \notin E$, then we are done by the induction hypothesis applied to the pseudobat induced in G by $a_j a_{j+1} \dots a_m$ and z; if $a_j a_m \in E$, then $a_1 a_j \notin E$, $a_j a_{i+1} \notin E$, $a_1 a_{i+1} \notin E$ (consider the neighborhood of z), and we are done by the induction hypothesis applied to the pseudobat induced in G by $a_1 a_2 \dots a_j a_m a_{m-1} \dots a_{i+1}$ and z.

CASE 2. *z is adjacent to no a_j other than a_1, a_i, a_{i+1}, a_m.*

We may assume that the path $a_1 a_2 \dots a_m$ has at least one chord (else G is a bat and we are done). If no chord of this path extends into a triangle, then G has an odd hole since conditions of Lemma 3.1 are satisfied for G or for the subgraph induced in G by $a_1 a_2 \dots a_i a_{i+1}$ and z or for the subgraph induced in G by $a_i a_{i+1} \dots a_m$ and z. Thus we may assume that a chord e extends into a triangle, T; since $z \notin T$, symmetry allows us to assume that T is $a_j a_k a_{k+1}$ for some j and k with $1 \le j \le i$ and $i + 1 \le k \le m - 1$. Among all such triangles, choose one with j as small as possible. If $j \le i - 1$, then Lemma 3.3 guarantees that $G - a_i$ contains an induced pseudobat (in which case we are done by the induction hypothesis). Thus we may assume that $j = i$.

SUBCASE 2.1. $a_i a_{i+2} \notin E$.

Now $k \ge i+3$ and (since $z a_1 a_i a_m$ is a claw) $k \le m-2$, and so $z a_k \notin E, z a_{k+1} \notin E$; minimality of j implies that $a_{i-1} a_k \notin E$ or $a_{i-1} a_{k+1} \notin E$ (or both); hence $a_i a_{i-1} z a_k$ or $a_i a_{i-1} z a_{k+1}$ is a claw. It follows that $a_i a_{i-1} z a_k$ is a claw (consider the neighborhood of a_i), and so we are done by the induction hypothesis applied to the pseudobat induced in G by $a_{i-1} \dots a_2 a_1 z a_{i+1} a_{i+2} \dots a_k$ and a_i.

SUBCASE 2.2. $a_i a_{i+2} \in E$.

Now $a_{i-1}a_{i+2} \in E$ (consider the neighborhood of a_i), and so minimality of j implies $a_{i-1}a_{i+1} \notin E$, $a_{i-1}a_{i+3} \notin E$; since the path $a_{i+1}a_{i+2}\dots a_m$ is chordless, $a_{i+1}a_{i+3} \notin E$. Thus G is not friendly (consider the neighborhood of a_{i+2}), a contradiction. □

Proof of Theorem 2.3. Consider an arbitrary bat-free friendly graph G that contains no odd hole, is not bipartite, and is not claw-free. We shall find a separator in G.

For each vertex x of G, write $x \in A$ if x is in a triangle; write $x \in B$ if x is the center of a claw. By assumption, $A \neq \emptyset$ and $B \neq \emptyset$.

CASE 1. $A \cap B \neq \emptyset$.

Let z be a vertex in $A \cap B$. Since G is friendly, z has neighbors c_1, c_2, d_1, d_2 such that $c_1c_2 \notin E, d_1d_2$ and $c_id_j \notin E$ for all choices of i and j. Lemma 3.4 guarantees that $G - z$ contains no two vertex-disjoint paths from $\{c_1, c_2\}$ to $\{d_1, d_2\}$; now, by Menger's theorem, $G - z$ contains a vertex w such that $G - \{z, w\}$ contains no path from $\{c_1, c_2\}$ to $\{d_1, d_2\}$. Observe that $\{z, w\}$ is a separator in G.

CASE 2. $A \cap B = \emptyset$.

Let T be a triangle in G, let z be a vertex in B and let N denote the neighborhood of z. Lemmas 3.3 and 3.4 guarantee that G contains no three vertex-disjoint paths from N to T; now, by Menger's theorem, G contains a set S of at most two vertices such that $G - S$ contains no path from N to T. Observe that S is a separator in G. □

3.4. Proof of Theorem 2.4. The following simple fact will be used quite a few times.

LEMMA 3.5. *Let G be a graph with a chordless path $x_1x_2\dots x_n$ such that $n \geq 5$; let z be a vertex of G such that*

- *$zx_j \in E$ if and only if j is one of $1, 2, n-1, n$*

and such that

- *$x_2x_3\dots x_{n-1}$ is a connected component of $G - \{z, x_1, x_n\}$.*

If G is friendly, then $G - \{zx_1, zx_2, zx_{n-1}, zx_n\}$ is friendly; if G contains no odd hole, then $G - \{zx_1, zx_2, zx_{n-1}, zx_n\}$ contains no odd hole.

Proof. First of all, note that

(i) zx_1x_2 is the only triangle containing zx_1

(any other triangle zx_1y would form a dart with x_2 and x_{n-1}); similarly,

(ii) zx_nx_{n-1} is the only triangle containing zx_n.

Now write $F = G - \{zx_2, zx_{n-1}\}$. It is a routine matter to verify, using (i) and (ii), that F is friendly. The only hole in F that has a chord in G is $zx_1x_2\dots x_n$; if this hole is odd, then $zx_2x_3\dots x_{n-1}$ is an odd hole in G. The rest follows from (i) and (ii) by two applications of Lemma 3.2. □

Unless indicated otherwise, we shall use $a_1, \dots, a_i, a_{i+1}, \dots, a_m$ and z to denote the vertices of a generic bat just as we did in its definition: the bat consists of a chordless path $a_1a_2\dots a_m$ and an additional vertex z that is adjacent to a_1, a_i, a_{i+1}, a_m for some i with $3 \leq i \leq m-3$ and to no other a_j. Trivially, if a friendly graph contains a bat, then the triangle za_ia_{i+1} in the bat extends into a unique maximal clique; we shall often rely tacitly on a lemma that guarantees a stronger conclusion under the additional assumption that the friendly graph contains no odd hole.

LEMMA 3.6. *If a friendly graph G containing no odd hole contains a bat, then each edge of the triangle in the bat extends into a* unique *maximal clique of G.*

Proof. Since za_ia_{i+1} extends into a unique maximal clique, our task reduces to proving that no vertex of G other than z, a_i, a_{i+1} is adjacent to precisely two of

z, a_i, a_{i+1}. Assume the contrary. Symmetry allows us to distinguish between two cases.

CASE 1. *G includes a vertex x such that $xz \in E$, $xa_i \in E$, $xa_{i+1} \notin E$, and $x \neq a_{i+1}$.*

We must first have $xa_{i-1} \in E$ (else $a_{i-1}a_ia_{i+1}zx$ is a dart) and $xa_1 \in E$ (else $a_1a_ia_{i+1}zx$ is a dart) and $xa_m \in E$ (else $a_mzxa_ia_{i+1}$ is a dart), and then $i = 3$ (else $a_1a_{i-1}a_mzx$ is a dart). However, then $a_1a_2a_3a_mx$ is a dart, a contradiction.

CASE 2. *G includes a vertex x such that $xa_i \in E$, $xa_{i+1} \in E$, $xz \notin E$, and $x \neq z$.*

We must first have $xa_{i-1} \in E$ (else $a_{i-1}a_ia_{i+1}xz$ is a dart) and $xa_{i+2} \in E$ (else $a_{i+2}a_{i+1}a_ixz$ is a dart), and then $xa_j \notin E$ whenever $1 \leq j \leq i-2$ (else $a_jxa_{i+1}a_ia_{i+2}$ is a dart). However, then one of the holes $za_1a_2 \ldots a_iz$ and $za_1a_2 \ldots a_{i-1}xa_{i+1}z$ is odd, a contradiction. □

A path in G will be called *x-sparse* if it is chordless and contains no x-edge.

LEMMA 3.7. *Let a friendly graph containing no odd hole contain a bat. Then every z-sparse path joining the two wing-tips of the bat passes through the maximal clique that contains the triangle of the bat.*

Proof. Consider a counterexample G with the smallest number of edges. Some portion $b_1b_2 \ldots b_n$ of the z-sparse path is vertex-disjoint from the bat, with b_1 having at least one neighbor in $A[a_1, a_{i-1}]$ and with b_n having at least one neighbor in $A[a_{i+2}, a_m]$. Write $B = b_1b_2 \ldots b_n$. Since B is a portion of a z-sparse path from a_1 to a_m, neither a_1b_1 nor b_na_m is a z-edge. Minimality of G guarantees that

- every vertex of G belongs either to the bat or to B;
- no edges of G go from $A[a_1, a_{i-1}]$ to $B - b_1$;
- no edges of G go from $A[a_{i+2}, a_m]$ to $B - b_n$.

Furthermore, Lemma 3.2 and minimality of G guarantee that

- every edge of G that belongs neither to the bat nor to B has at least one of the four vertices a_i, a_{i+1}, b_1, b_n for an endpoint:

all other edges would have the form zb_j with $1 < j < n$, and so they could be removed.

CASE 1. *There are subscripts s, t such that $s < t$, $a_ib_t \in E$, $a_{i+1}b_s \in E$.*

Let t be as small as possible subject to the assumption of this case; once t has been fixed, let s be as large as possible subject to the assumption of this case. By Lemma 3.2 and minimality of G, each of the two edges $a_ib_t, a_{i+1}b_s$ extends into a triangle; since $a_ib_s \notin E$ and $a_{i+1}b_t \notin E$, it follows that $a_{i+1}b_{s-1}, a_ib_{t+1} \in E$.

SUBCASE 1.1. *$a_{i+1}b_r \in E$ for some r with $r < s - 1$.*

Let r be as large as possible subject to the assumption of this case. Since G is friendly, we have $r \leq s - 3$ (consider the neighborhood of a_{i+1}); maximality of r guarantees that $a_{i+1}b_j \notin E$ whenever $r < j < s - 1$; minimality of t guarantees that $a_ib_j \notin E$ whenever $r < j < s-1$. By Lemma 3.2 and minimality of G, the edge $a_{i+1}b_r$ extends into a triangle; since $a_ib_r \notin E$, it follows that $a_{i+1}b_{r-1} \in E$. However, then $G, B[b_{r-1}, b_s]$, and a_{i+1} satisfy the hypothesis of Lemma 3.5, and so the minimality of G is contradicted.

SUBCASE 1.2. *$a_{i+1}b_r \in E$ for no r with $r < s - 1$.*

Let F denote the subgraph of G induced by $A[a_1, a_i]$ and $B[b_1, b_{s-1}]$. By assumption of this case, no vertex of F other than a_i and b_{s-1} is adjacent to a_{i+1}; trivially, F contains a chordless path P from a_i to b_{s-1}. By Lemma 3.1, the subgraph of G induced by the union of P, a_{i+1}, and $B[b_s, b_t]$ contains an odd hole.

CASE 2. *There are no subscripts s, t such that $s < t$, $a_ib_t \in E$, $a_{i+1}b_s \in E$.*

By assumption of this case, there is a subscript j such that a_i is adjacent to none of $b_j, b_{j+1}, \ldots, b_n$ and such that a_{i+1} is adjacent to none of $b_1, b_2, \ldots, b_{j-1}$. Trivially,

$G - \{z, a_i\}$ contains a chordless path C from a_1 to a_{i+1} and $G - \{z, a_{i+1}\}$ contains a chordless path D from a_i to a_m. Since a_1 and a_{i+1} belong to distinct components of $G - \{z, a_i, b_j\}$, path C must pass through b_j; since a_i and a_m belong to distinct components of $G - \{z, a_{i+1}, b_j\}$, path D must also pass through b_j. Write

$$C_1 = C[a_1, b_j], \quad C_2 = C[b_j, a_{i+1}], \quad D_1 = D[a_i, b_j], \quad D_2 = D[b_j, a_m];$$

let c_k and d_k denote the number of edges in C_k and D_k, respectively. Since at least one of the four numbers $c_1 + c_2 + 2$, $d_1 + d_2 + 2$, $c_1 + d_2 + 2$, $d_1 + c_2 + 1$ is odd, at least one of the four cycles Cz, Dz, C_1D_2z, D_1C_2 is odd. Since none of the four subgraphs of G induced by these four cycles contains a triangle (in particular, neither za_ib_1 nor $za_{i+1}b_n$ is a triangle as the neighborhood of z consists of vertex-disjoint cliques), it follows that G contains an odd hole. □

By a *z-splitter*, we mean a bat (with head z) along with a z-sparse path between the two wing-tips of the bat that passes through one of a_i, a_{i+1}.

LEMMA 3.8. *Let G be a friendly graph containing no odd hole and let z be a vertex of G such that G contains a z-splitter. Then G contains a fragile bat with head z.*

Proof. Among all counterexamples (G, z), choose one with G having as few edges as possible. In this G and for this z, consider a z-splitter with as few vertices as possible.

Symmetry allows us to assume that the z-sparse path between the two wing-tips of the bat in the z-splitter passes through a_{i+1}; now minimality of the z-splitter implies that this path has the form $XBA[a_{i+1}, a_m]$ with X a prefix of $A[a_1, a_i)$. Write $B = b_1b_2 \dots b_n$. Minimality of the z-splitter guarantees that

- B is chordless and vertex-disjoint from A;
- no vertex in $A[a_1, a_i)$ has a neighbor in $B(b_1, b_n]$;
- a_{i+1} is adjacent to no vertex in $B[b_1, b_n)$.

We claim that

(α) some vertex in $A(a_1, a_i)$ is adjacent to b_1.

To justify this claim, assume the contrary: no vertex in $A(a_1, a_i)$ has a neighbor in B. With F standing for the graph induced in G by $\{z\} \cup A[a_1, a_{i+1}] \cup B$, Lemma 3.1 guarantees that za_ia_{i+1} is not the only triangle in F; as $XBA[a_{i+1}, a_m]$ is a z-sparse path, all other triangles in F must have the form $a_ib_kb_{k+1}$ with $1 \le k < n$. Let k be the smallest subscript such that $a_ib_k \in E$, write $b_0 = a_1$, and let j be the largest subscript such that $zb_j \in E$ and $0 \le j < k$. Now the path $b_j \dots b_ka_i \dots a_m$ and z induce in G a bat with head z, whose wing-tips b_j and a_m are joined by the z-sparse path $b_j \dots b_na_{i+1} \dots a_m$. Hence minimality of our z-splitter is contradicted: a proper subset of its vertex-set (a_2 is missing) induces another z-splitter. This contradiction completes the justification of (α).

Furthermore, we claim that

(β) z has no neighbor in B.

To justify this claim, assume the contrary and let k be the smallest subscript such that $zb_k \in E$; note that (since $XBA[a_{i+1}, a_m]$ is z-sparse) $k < n$. If $a_ib_s \in E$ for some s such that $1 \le s \le k$, then let s be the largest subscript with this property, note that (since $a_{i+1}b_k \notin E$) $s < k$, and set $r = i$; else let r be the largest subscript such that $a_rb_1 \in E$ and set $s = 1$. In either case, the path $b_k \dots b_sa_r \dots a_m$ and z induce in G a bat with head z, whose wing-tips b_k and a_m are joined by the z-sparse path $b_k \dots b_na_{i+1} \dots a_m$. Hence minimality of our z-splitter is contradicted: a proper

subset of its vertex-set (a_1 is missing) induces another z-splitter. This contradiction completes the justification of (β).

Lemma 3.7 guarantees that

- no edge of G goes from $A[a_1, a_i) \cup B$ to $A(a_{i+1}, a_m]$.

Let K denote the maximal clique in G that contains the triangle za_ia_{i+1}. Since (G, z) is a counterexample, a_1 and a_m belong to the same component of $G - K$. Hence there is a path $c_1c_2 \dots c_p$ in $G - K$ such that c_1 has a neighbor in $A[a_1, a_i) \cup B$ and c_p has a neighbor in $A(a_{i+1}, a_m]$; among all such paths, choose one with p as small as possible and write $C = c_1c_2 \dots c_p$. Minimality of p guarantees that

- C is chordless and vertex-disjoint from $A[a_1, a_i) \cup B \cup A(a_{i+1}, a_m]$;
- no vertex of $C(c_1, c_p]$ has a neighbor in $A[a_1, a_i) \cup B$;
- no vertex of $C[c_1, c_p)$ has a neighbor in $A(a_{i+1}, a_m]$.

With G_0 standing for the subgraph of G induced by the union of the splitter and C, observe that G_0 has no clique-cutset K' such that $z \in K'$. Hence (G_0, z) is also a counterexample; now mimimality of G guarantees that

- G has no vertices outside the splitter and C.

Lemma 3.2 and minimality of G guarantee that

- every edge a_ic_j extends into a triangle;
- every edge $a_{i+1}c_j$ extends into a triangle;

note that

- if a_ic_jx is a triangle and $j > 1$, then $x = c_{j-1}$ or $x = c_{j+1}$;
- if $a_{i+1}c_jx$ is a triangle and $1 < j < p$, then $x = c_{j-1}$ or $x = c_{j+1}$.

There is a chordless path from a_1 to a_m of the form YCZ such that Y is a prefix of XB and Z is a suffix of $A(a_{i+1}, a_m]$; Lemma 3.7 guarantees that this path is not z-sparse; since neither Y nor Z contains a z-edge, it follows that C includes at least one neighbor of z. Let v denote the first neighbor of z on C.

We claim that

(γ) no edge of G goes from $A[a_1, a_{i+1}) \cup B \cup C[c_1, v)$ to $A(a_{i+1}, a_m] \cup C(v, c_p]$.

To justify this claim, assume the contrary. This assumption implies that some edge of G joins a_i to a vertex in $C(v, c_p]$; it follows that $C[v, c_p]$ contains an a_i-edge. There is a suffix Y of $A(a_{i+1}, a_m]$ such that $C[v, c_p]Y$ is a chordless path; write $D = C[v, c_p]Y$ and note that (as D is vertex-disjoint from K) each vertex of D is adjacent to at most one of z, a_i, and a_{i+1}. Consider a minimal portion $D[x, y]$ of D such that x, y are neighbors of z and such that $D[x, y]$ contains an a_i-edge; note that no vertex of $D(x, y)$ is adjacent to z. If $D[x, y]$ contains precisely one a_i-edge, then the subgraph of G induced by $D[x, y] \cup \{z, a_i\}$ satisfies the hypothesis of Lemma 3.1, and so G contains an odd hole. Hence $D[x, y]$ contains at least two a_i-edges. These two a_i-edges lie on $C[v, c_p]$; consider a minimal portion $C[c_r, c_s]$ of $C[v, c_p] \cap D(x, y)$ that contains two a_k-edges with $k = i$ or $k = i + 1$. Lemma 3.5 and minimality of G guarantee that $C(c_r, c_s)$ is not a connected component of $G - \{a_k, c_r, c_s\}$; it follows that some vertex of $C(c_r, c_s)$ is adjacent to a_t with $\{t, k\} = \{i, i + 1\}$; now $C(c_r, c_s)$ contains precisely one a_t-edge and no a_k-edge. However, then Lemma 3.1 guarantees that the subgraph of G induced by $C(c_r, c_s) \cup \{a_i, a_{i+1}\}$ contains an odd hole; this contradiction completes the justification of (γ).

The subgraph of G induced by $A[a_{i+1}, a_m] \cup C[v, c_p]$ contains a chordless path from a_{i+1} to v; let P denote this path. We propose to find, in the subgraph of G induced by $A[a_1, a_{i+1}] \cup B \cup C[c_1, v]$, chordless paths P_0, P_1 from v to a_{i+1} such that P_0 has an even number of edges and P_1 has an odd number of edges. This will complete

the proof of Lemma 3.8: one of the cycles P_0P and P_1P is odd and, by virtue of (γ), chordless.

In finding P_0 and P_1, we shall rely tacitly on the fact that

- the subgraph of G induced by $A(a_1, a_i) \cup B$ is connected and includes no neighbor of z,

which is guaranteed by (α) and (β).

CASE 1. *At least one of a_i and a_{i+1} has a neighbor in $C[c_1, v)$.*

Since the subgraph of G induced by $A[a_1, a_i) \cup B \cup C[c_1, v] \cup \{a_{i+1}\}$ is connected, it contains a chordless path P_0 from v to a_{i+1}; by assumption of this case, $c_1 \neq v$, and so $a_1v \notin E$; it follows that P_0 is z-sparse. Since the subgraph of G induced by P_0 and z contains no triangle, it must be bipartite; in particular, the cycle P_0z must be even; hence P_0 has an even number of edges.

Let u be the last vertex on $C[c_1, v)$ adjacent to one of a_i, a_{i+1}. If $ua_{i+1} \in E$ and u has a neighbor in $A[a_1, a_i)$, then $u = c_1$; in this case, let w denote the last neighbor of u on $A[a_1, a_i]$; Lemma 3.1 guarantees that the subgraph of G induced by z, $A[w, a_{i+1}]$, and $C[u, v]$ contains an odd hole. If $ua_{i+1} \in E$ and u has no neighbor in $A[a_1, a_i)$, then $A[a_1, a_{i+1}]C[u, v]$ and z induce a bat whose wing-tips are joined by a z-sparse path in the subgraph of G induced by $A[a_1, a_i) \cup B \cup C[c_1, v]$, contradicting Lemma 3.7. Hence $ua_i \in E$. With Q standing for $C[u, v]$ reversed, write $P_1 = Qa_ia_{i+1}$; since $P_1[v, a_i]z$ is a hole, P_1 has an odd number of edges.

CASE 2. *Neither a_i nor a_{i+1} has a neighbor in $C[c_1, v)$.*

SUBCASE 2.1. *c_1 has a neighbor in $A(a_1, a_i) \cup B[b_1, b_n)$.*

By assumption of this subcase, the subgraph of G induced by $A(a_1, a_i) \cup B \cup \{a_{i+1}\} \cup C[c_1, v]$ contains a chordless path P_0 from v to a_{i+1} and the subgraph of G induced by $A(a_1, a_i] \cup B[b_1, b_n) \cup C[c_1, v]$ contains a chordless path Q from v to a_i. Observe that each of P_0z, Qz is a hole, and so each of P_0, Q has an even number of edges. Set $P_1 = Qa_{i+1}$ and observe that P_1 is a chordless path.

SUBCASE 2.2. *c_1 has no neighbor in $A(a_1, a_i) \cup B[b_1, b_n)$.*

The subgraph of G induced by $A(a_1, a_i] \cup B$ contains a chordless path Q from b_n to a_i. Since G contains no odd hole, its subgraph induced by $Q \cup C[c_1, v] \cup \{z, a_{i+1}\}$ must not satisfy the hypothesis of Lemma 3.1; under the assumption of this subcase, this means $c_1b_n \notin E$. Hence a_1 is the unique neighbor of c_1 in $A[a_1, a_i) \cup B$.

The subgraph of G induced by $A[a_1, a_i) \cup B \cup \{a_{i+1}\}$ contains a chordless path S from a_1 to a_{i+1}. Observe that each of $A[a_1, a_i]z$, Sz is a hole, and so each of $A[a_1, a_i]$, S has an even number of edges. With R standing for $C[c_1, v]$ reversed, each of $RA[a_1, a_{i+1}]$ and RS is a chordless path; one of these paths has an even number of edges and the other has an odd number of edges. □

A path $w_1w_2 \dots w_m$ in G will be called *z-special* if

- $w_1w_2 \dots w_m$ is a chordless path in $G * z$,
- w_1 and w_m are neighbors of z,
- z has at least two neighbors in $w_2w_3 \dots w_{m-1}$, the set N of these neighbors forms a clique, and $w_1w \notin E, w_mw \notin E$ whenever $w \in N$.

(Note that w_1 and w_m may or may not be adjacent.)

LEMMA 3.9. *Let G be a friendly graph containing no odd hole and let z be a vertex of G such that z is the center of a claw. If G contains a z-special path, then G and z have at least one of the following properties:*

(i) *G contains a fragile bat with head z;*

(ii) *G contains a clique-cutset K such that $z \in K$ and some component of $G - K$ includes no neighbor of z.*

Proof. Note that the neighborhood of z consists of vertex-disjoint cliques. We shall proceed by induction on the number of neighbors of z on the z-special path. Let A denote the z-special path; write $A = a_1 a_2 \dots a_m$; let a_i be the first neighbor of z on $A(a_1, a_m)$ and let a_j be the last neighbor of z on $A(a_1, a_m)$; let K denote the maximal clique in G that contains z and all its neighbors on $A(a_1, a_m)$. Write

$$A_O = A[a_1, a_i) \cup A(a_j, a_m] \text{ and } A_I = A(a_i, a_j) - K.$$

(The subscripts O and I are mnemonic for "outside" and "inside".) If A_O and A_I belong to the same connected component of $G - K$, then $G - K$ contains a path $b_1 b_2 \dots b_n$ such that b_1 has a neighbor in A_O and b_n has a neighbor in A_I. If A_O and A_I belong to distinct connected components of $G - K$, then consider a component Q of $G - K$ that includes a vertex of A_I: unless (ii) holds (in which case we are done), Q includes a neighbor of z, and so Q contains a path $b_1 b_2 \dots b_n$ such that b_1 is a neighbor of z and b_n has a neighbor in A_I. Hence we may assume in any case that $G - K$ contains a path $b_1 b_2 \dots b_n$ such that

- b_1 has a neighbor in $A_O \cup \{z\}$;
- b_n has a neighbor in A_I;

taking n as small as possible, we may assume further that

- $b_1 b_2 \dots b_n$ is chordless and vertex-disjoint from A;
- no b_k with $k > 1$ has a neighbor in $A_O \cup \{z\}$;
- no b_k with $k < n$ has a neighbor in A_I.

Write $B = b_1 b_2 \dots b_n$ and note that

- every neighbor of z in $A \cup B$ belongs to $K \cup \{a_1, a_m, b_1\}$.

Let F denote the subgraph of $G * z$ induced by $A \cup B$.

Case 1. *b_1 has no neighbor in A_O.*

F contains a chordless path C from b_1 to a_1 that avoids $A[a_j, a_m]$; by assumption of this case, C must pass through a_i; if some other interior vertex of C also belongs to K, then C is z-special and we are done by the induction hypothesis; hence we may assume that b_1, a_i, a_1 are the only neighbors of z on C; in particular, C is z-sparse. Similarly, F contains a z-sparse path D from b_1 to a_m such that D avoids $A[a_1, a_i]$ and such that b_1, a_j, a_m are the only neighbors of z on D.

If a_j has no neighbor on $C[b_1, a_i)$, then $C[b_1, a_i]$, $A[a_j, a_m]$, and z induce in G a bat with head z, whose wing-tips are joined by the z-sparse path D, and so (i) follows by Lemma 3.8. If a_j has a neighbor on $C[b_1, a_i)$, then let x be the first neighbor of a_j on $C[b_1, a_i)$; note that x is not the last vertex on $C[b_1, a_i)$ (else $x a_i z a_j a_{j+1}$ would be a dart); now $C[b_1, x] a_j$, $A[a_1, a_i]$, and z induce in G a bat with head z, whose wing-tips are joined by the z-sparse path C, and so (i) follows again by Lemma 3.8.

Case 2. *At least one of $z b_1 a_1$ and $z b_1 a_m$ is a triangle.*

Symmetry allows us to assume that $z b_1 a_1$ is a triangle.

Subcase 2.1. *b_1 has no neighbor in $A(a_1, a_i)$.*

F contains a chordless path C from b_1 to a_1 that avoids $A[a_j, a_m]$; by assumption of this subcase, C must pass through a_i. Lemma 3.1 applied to the subgraph of G induced by $C \cup \{z\}$ guarantees that a_i is not the only interior vertex of C that belongs to K. Hence C is z-special and we are done by the induction hypothesis.

Subcase 2.2. *b_1 has a neighbor in $A(a_1, a_i)$ and it has a neighbor in $A(a_j, a_m]$.*

Let u be the last neighbor of b_1 on $A(a_1, a_i)$ and let v be the first neighbor of b_1 on $A(a_j, a_m]$; Lemma 3.1 guarantees that $z a_i a_j$ is not the only triangle in the subgraph

of G induced by $A[u, a_i] \cup A[a_j, v] \cup \{b_1, z\}$. Hence $v = a_m$, which brings us back to a mirror image of Subcase 2.1.

SUBCASE 2.3. *b_1 has a neighbor in $A(a_1, a_i)$ and it has no neighbor in $A(a_j, a_m]$.*

F contains a chordless path C from b_1 to a_m that avoids $A[a_1, a_i]$; by assumption of this subcase, C must pass through a_j. If C is z-special, then we are done by the induction hypothesis; hence we may assume that a_j is the only interior vertex of C that belongs to K. With x standing for the last neighbor of b_1 on $A(a_1, a_i)$, observe that $b_1A[x, a_i]A[a_j, a_m]$ and z induce a bat in G, whose wing-tips are joined by the z-sparse path C; hence (i) follows by Lemma 3.8.

CASE 3. *b_1 has a neighbor in A_O; neither zb_1a_1 nor zb_1a_m is a triangle.*

By assumption of this case, F contains a chordless path C from a_1 to a_m that avoids at least one of a_i and a_j and has the following property: *If b_1 appears on C at all, then it appears either before all the vertices of $C \cap K$ or after all the vertices of $C \cap K$.* If C includes at least two vertices of K, then C is z-special or else one of $C[a_1, b_1]$, $C[b_1, a_m]$ is z-special and we are done by the induction hypothesis; hence we may assume that

- C includes at most one vertex of K.

Note that

- if C includes a vertex of K, then this vertex is a_i or a_j.

The assumption of this case guarantees that

- C has no chords in G except possibly a_1a_m.

If $a_1a_m \notin E$, then $A[a_1, a_i]$, $A[a_j, a_m]$, and z induce a bat with head z, whose wing-tips are joined by the z-sparse path C; Lemma 3.7 guarantees that C includes a vertex of K; hence the bat and C constitute a splitter and (i) follows by Lemma 3.8. We propose to complete the proof by deriving a contradiction from the assumption that

- $a_1a_m \in E$.

Since C includes no z-edge, Lemma 3.1 applied to the subgraph of G induced by $C \cup \{z\}$ guarantees that

- no interior vertex of C is adjacent to z.

Now all the vertices of C come from $(A \cup B) - K$, which is $A_O \cup A_I \cup B$. With F_0 standing for the subgraph of $G * z$ induced by $A_O \cup A_I \cup B$, observe that $A[a_1, a_i)$ is a connected component of $F_0 - b_1$ and that $A(a_j, a_m]$ is another; it follows that

- $C = A[a_1, x]b_1A[y, a_m]$

for some x on $A[a_1, a_i)$ and some y on $A(a_j, a_m]$.

SUBCASE 3.1. *$x = a_{i-1}$ and $y = a_{j+1}$.*

Let H_1 denote the subgraph of G induced by $A[a_1, a_i] \cup A[a_j, a_m]$ and let H_2 denote the subgraph of G induced by C. By definition, H_1 is a hole of length at least six; by assumption of this subcase, H_2 is a hole whose length is one less than the length of H_1. Hence one of H_1, H_2 is odd, a contradiction.

SUBCASE 3.2. *$x \neq a_{i-1}$ or $y \neq a_{j+1}$.*

Symmetry allows us to assume that $x \neq a_{i-1}$. Observe that B extends into a chordless path D in F such that D leads from b_1 to some vertex in $A \cap K$ other than a_j and such that no interior vertex of D has a neighbor in $A_O \cup \{z\}$. Lemma 3.1 is contradicted by the subgraph of G induced by C, z, and D. □

LEMMA 3.10. *Let G be a friendly graph containing no odd hole and let xyz be a triangle in G such that z is the center of a claw; let K be the maximal clique in G that contains the triangle xyz. If $G * z$ contains a chordless path P from x to y such that some neighbor of z on P does not belong to K, then G and z have at least one of the following properties:*

(i) *G contains a fragile bat with head z;*

(ii) *G contains a clique-cutset K' such that $z \in K'$ and some component of $G - K'$ includes no neighbor of z.*

Proof. The neighborhood of z consists of vertex-disjoint cliques $K_1, K_2, \ldots, K_m$ such that $K = K_1$. By assumption, $G * z$ contains a chordless path P from x to y such that some neighbor of z on P does not belong to K_1. Let $P[a, f]$ be a minimal segment of P such that a, f belong to the same K_r and some vertex of $P(a, f)$ belongs to a K_s with $s \neq r$; write $Q = P[a, f]$. By minimality of Q, no interior vertex of Q belongs to K_r; Lemma 3.1 guarantees that the subgraph of G induced by Q and z contains a triangle other than zaf; hence some two interior vertices of Q belong to the same K_t. Let c and d be the first and the last vertex in Q that belong to K_t; note that (by minimality of Q) all the neighbors of z in $Q[c, d]$ belong to K_t. Let b be the last neighbor of z in $Q[a, c)$ and let e be the first neighbor of z in $Q(d, f]$. Then $Q[b, e]$ is a z-special path and the desired conclusion follows by Lemma 3.9. ∎

Proof of Theorem 2.4. Note that the assumption

"a_1 and a_m belong to the same component of $G * z$"

is equivalent to the assumption

"a_i and a_{i+1} belong to the same component of $G * z$."

Write $A = a_1 a_2 \ldots a_m$ and let K denote the maximal clique in G that contains the triangle za_ia_{i+1}. We may assume that the bat is not fragile: $G - K$ contains a path from a_1 to a_m. Hence $G - a_ia_{i+1}$ contains a path P from a_i to a_{i+1} with all interior vertices outside K; write $P = p_1 p_2 \ldots p_t$.

Case 1. *P contains no z-edge.*

Now P is a path in $G * z$; we may assume that P is a chordless path in $G * z$ (but not necessarily a chordless path in $G - a_ia_{i+1}$); then Lemma 3.10 allows us to assume that no interior vertex of P is adjacent to z. Since $G * z$ contains the path $A[a_1, a_i]PA[a_{i+1}, a_m]$, it contains a chordless path Q from a_1 to a_m such that all neighbors of z on Q come from $\{a_1, a_i, a_{i+1}, a_m\}$; Lemma 3.7 guarantees that Q contains at least one of a_i, a_{i+1}. If Q contains precisely one of a_i, a_{i+1}, then the desired conclusion follows from Lemma 3.8; else the desired conclusion follows from Lemma 3.9.

Case 2. *P contains precisely one z-edge.*

Let p_jp_{j+1} be the unique z-edge on P and let K' denote the maximal clique in G that contains the triangle zp_jp_{j+1}. Since $G * z$ contains $P[p_1, p_j]$, $P[p_{j+1}, p_t]$, and a path from p_1 to p_t, it contains a chordless path Q from p_j to p_{j+1}. Lemma 3.10 allows us to assume that all the neighbors of z on Q come from K'. However, then the walk $P[p_1, p_{j-1}]QP[p_{j+2}, p_t]$ in $G * z$ contains no vertices of K other than a_i and a_{i+1}, which brings us back to Case 1.

Case 3. *P contains at least two z-edges.*

We may assume that P is a chordless path in $G - a_ia_{i+1}$. By assumption of this case, z has at least four neighbors on $P(p_1, p_t)$; with p_r standing for the first neighbor of z on $P(p_1, p_t)$ and with p_s standing for the last neighbor of z on $P(p_1, p_t)$, let us write

$$B = b_1 b_2 \ldots b_n = p_r p_{r-1} \ldots p_1 p_t p_{t-1} \ldots p_s \quad \text{and} \quad b_j = p_1, b_{j+1} = p_t.$$

Now B and z induce in G a bat; note that $b_j = a_i$, $b_{j+1} = a_{i+1}$, and so K is the maximal clique in G that contains the triangle zb_jb_{j+1} in this new bat.

By assumption, $G * z$ contains a chordless path Q from b_j to b_{j+1}; Lemma 3.10 allows us to assume that all the neighbors of z on Q come from K. Since $G * z$ contains

the path $B[b_1, b_j]QB[b_{j+1}, b_n]$, it contains a chordless path C from b_1 to b_n such that all vertices of C come from $B \cup Q$; write $C = c_1c_2 \dots c_p$. Lemma 3.7 allows us to assume that C contains at least one vertex of K; Lemma 3.9 allows us to assume that C contains at most one vertex of K; let x be the unique vertex in $C \cap K$.

Lemma 3.7 allows us to assume that

- ∘ no vertex in $C[c_1, x)$ has a neighbor in $B(b_{j+1}, b_n]$;
- ∘ no vertex in $C(x, c_p]$ has a neighbor in $B[b_1, b_j)$;

in turn, Lemma 3.8 allows us to assume that

- • x is distinct from b_j and b_{j+1};
- ∘ no vertex in $C[c_1, x)$ is adjacent to b_{j+1};
- ∘ no vertex in $C(x, c_p]$ is adjacent to b_j.

Now if x has no neighbor in $B(b_1, b_j)$, then $B[b_1, b_j]C[x, c_p]$ and z induce a bat; since this bat and C constitute a z-splitter, Lemma 3.8 allows us to assume that

- ∘ x has a neighbor in $B(b_1, b_j)$;
- ∘ x has a neighbor in $B(b_{j+1}, b_n)$.

Letting u denote the first neighbor of x in $B(b_1, b_j)$ and letting v denote the last neighbor of x in $B(b_{j+1}, b_n)$, we may just as well assume that

- • $C = B[b_1, u]xB[v, b_n]$.

Since $C \cup \{z\}$ induces a cycle in G and since zx is the unique chord of this cycle, C has an even number of edges; $B \cup \{z\}$ induces a bat in G, and so B has an odd number of edges; let us note for future reference that

- • the lengths of B and C differ in parity.

Recall that the vertex-set of P induces in G a hole H such that

- • B is a segment of H.

We propose to complete the proof by contradicting the assumption that the subgraph of G induced by $H \cup \{z, x\}$ is friendly and contains no odd hole.

For this purpose, first note that (since the neighborhood of z consists of vertex-disjoint cliques and since b_j, b_{j+1} are the only vertices of K on H),

- • z and x have no common neighbor on H other than b_j and b_{j+1}.

H is a concatenation of two paths: path B from b_1 to b_n and its counterpart $\overline{B}$ $(= H - B(b_1, b_n))$ from b_n to b_1. Starting with the subgraph F of G induced by $H \cup \{z, x\}$, keep reducing F as long as possible by the following two operations:

(α) If $\overline{B}$ contains a vertex w such that
w is adjacent to x and the edge xw extends into no triangle,
then remove the edge xw from F.

(β) If $\overline{B}$ contains a segment $w_1w_2 \dots w_d$ such that
w_1, w_2, w_{d-1}, w_d are adjacent to x,
none of $w_3, \dots, w_{d-2}$ are adjacent to x,
and none of $w_1, w_2, \dots, w_d$ are adjacent to z,
then remove the four edges $xw_1, xw_2, xw_{d-1}, xw_d$ from F.

Lemma 3.2 guarantees that the invariant

- • F is a friendly graph containing no odd hole

is maintained after each application of (α); Lemma 3.5 guarantees that this invariant is maintained after each application of (β). Now let us distinguish between two subcases.

SUBCASE 3.1. *x has a neighbor on $\overline{B}$.*

Since transformation (α) is not (any more) applicable to F, the assumption of this subcase guarantees that $\overline{B}$ contains an x-edge, e. Let $\overline{B}_0$ be a minimal segment of $\overline{B}$ that contains e, begins at a neighbor of z, and ends at a neighbor of z. Since

transformation (β) is not (any more) applicable to F, edge e is the only x-edge in $\overline{B}_0$. However, then the subgraph of F induced by $\overline{B}_0 \cup \{z, x\}$ contradicts Lemma 3.1.

SUBCASE 3.2. *x has no neighbor on $\overline{B}$.*

The concatenation of B and $\overline{B}$ is H; by assumption of this subcase, the concatenation of C and $\overline{B}$ is another hole; since the lengths of B and C differ in parity, one of these two holes is odd, a contradiction. □

4. The algorithm. Theorem 2.3 suggests a recursive algorithm that, given any friendly bat-free graph G, finds out whether or not G is a Berge graph. This algorithm, TEST(G), uses a procedure DECOMP1(G; S) that, given any friendly graph G and a separator S in G, either finds an odd hole in G at once or else constructs friendly graphs G_1 and G_2 such that

- if G is a bat-free Berge graph,
  then both of G_1 and G_2 are bat-free Berge graphs;
- if G is not a Berge graph,
  then at least one of G_1 and G_2 is not a Berge graph.

DECOMP1(G; S) goes as follows.

By definition, the set of vertices of $G - S$ splits into disjoint nonempty sets V_1 and V_2 such that no edge of G joins a vertex in V_1 to a vertex in V_2; if S consists of two nonadjacent vertices, then each V_i includes at least two vertices. Let F_i denote the subgraph of G induced by $V_i \cup S$. If S is a clique, then DECOMP1(G; S) simply returns F_1 and F_2. If S consists of nonadjacent vertices u and v, then we may assume that both $G - u$ and $G - v$ are connected (else a one-point separator could be used in place of S) and find a chordless path P_i from u to v in each F_i; now if both P_i have an odd number of edges, then DECOMP1(G; S) returns graphs G_1 and G_2 such that each G_i is F_i with one additional edge, uv; if both P_i have an even number of edges, then DECOMP1(G; S) returns graphs G_1 and G_2 such that each G_i is F_i with one additional vertex, w, and two additional edges, uw and vw; if one P_i has an odd number of edges and the other P_i has an even number of edges, then DECOMP1(G; S) returns the odd hole $P_1 \cup P_2$.

```
TEST(G):
(Step 1) if G is bipartite then return true end
(Step 2) if      G is claw-free
         then if G is a Berge graph then return true else return false end
         end
(Step 3) if G has no separator then return false end
(Step 4) S = separator in G;
         if      DECOMP1(G; S) returns graphs G1 and G2
         then return TEST(G1)∧TEST(G2);
         else    return false;
         end
```

LEMMA 4.1. *Let G be a friendly graph. If* TEST(G) *returns* true*, then G is a Berge graph; if* TEST(G) *returns* false*, then G is not a Berge graph or else G contains a bat.*

Proof. We use induction on the number of vertices of G. If TEST(G) returns *false* in Step 3, then Theorem 2.3 guarantees that G contains an odd hole or a bat. If DECOMP1(G; S) returns graphs G_1 and G_2 in Step 4, then G_1 and G_2 are both friendly; if both of them are Berge graphs, then G is a Berge graph; if at least one of them is not a Berge graph, then G is not a Berge graph; if at least one of them

contains a bat, then G contains a bat. (When S consists of vertices u and v that are nonadjacent in G and adjacent in G_i, it may help to keep in mind that u and v have no common neighbor.) □

LEMMA 4.2. TEST(G) *runs in polynomial time.*

Proof. Each step of the algorithm, except for the recursive calls to evaluate TEST(G_1) and TEST(G_2) in Step 4, can be executed in polynomial time; in particular, Step 2 can be executed in polynomial time by the algorithm of Chvátal and Sbihi [6]. Thus our task reduces to showing that the number $t(G)$ of nodes in the recursion tree of TEST(G) does not exceed some polynomial in n, the number of vertices of the input graph G.

For this purpose, first note that

$$t(G) = 1 \text{ whenever } n \leq 4$$

(since every graph with at most four vertices is bipartite or claw-free). Now induction on n shows that

$$t(G) \leq 4n - 17 \text{ whenever } n \geq 5 :$$

if G_1, G_2 are the graphs returned by DECOMP1(G;S) in Step 4, then

$$t(G) = 1 + t(G_1) + t(G_2)$$

and, with n_i standing for the number of vertices of G_i,

$$n_1 \leq n - 1,\ n_2 \leq n - 1,\ \text{and } n_1 + n_2 \leq n + 4. \qquad \square$$

Our main algorithm, given any friendly graph G, returns *true* if and only if G is a Berge graph. This algorithm, BERGE(G), is also recursive; it evolves from Theorem 2.2 much as Algorithm TEST(G) evolves from Theorem 2.3.

If G, the input of BERGE(G), contains a clique-cutset C, then we call a procedure DECOMP2(G; C) which goes as follows: Let $F_1, F_2, \ldots, F_k$ be the components of $G-C$; let G_i denote the subgraph of G induced by $F_i \cup C$; return $G_1, G_2, \ldots, G_k$.

If G contains a rosette centered at z (but G contains no clique-cutset), then we call a more complicated procedure DECOMP3(G; z) which goes as follows: Let $F_1, F_2, \ldots, F_k$ be the components of $G * z$; let G_i denote the subgraph of G induced by $F_i \cup \{z\}$. Let N denote the subgraph of G induced by all the neighbors of z; let H be the graph obtained from N by adding pairwise nonadjacent vertices $w_1, w_2, \ldots, w_k$ along with all the edges xw_i such that $x \in N \cap F_i$. Return $G_1, G_2, \ldots, G_k$ and H.

DECOMP3 is illustrated in Figure 4.1. There, G is not a Berge graph but G_1, G_2, G_3 are; the unique odd hole in G is contained in none of G_1, G_2, G_3, but it reappears in H.

In general, H may be seen as a device for detecting those odd holes in G that are contained in none of $G_1, G_2, \ldots, G_k$: at first, one might be tempted to conjecture that G is a Berge graph if and only if $G_1, G_2, \ldots, G_k$ and H are all Berge graphs. Unfortunately, this is not the case: two counterexamples are shown in Figure 4.2. Fortunately, these two counterexamples are harmless: each of them has a clique-cutset, and so it can be subjected to DECOMP2 instead of DECOMP3.

We make it our policy to subject G to DECOMP3 only if it has no clique-cutset. Under this assumption (as we shall prove later), G is a Berge graph if and only if $G_1, G_2, \ldots, G_k$ and H are all Berge graphs *and* none of $G_1, G_2, \ldots, G_k$ contains a fragile bat with head z. The proviso involving fragile bats cannot be dropped: see Figure 4.3.

A straightforward algorithm NoFB$(F; z)$, given any graph F along with a vertex z of F such that the neighborhood of z in F consists of vertex-disjoint cliques, returns *true* if and only if F contains no fragile bat with head z. There, as usual, $N(v)$ denotes the neighborhood of v.

NoFB$(F; z)$:
for all z-edges xy
do $C =$ the maximal clique of F that contains xyz;
  for all choices of distinct components A, B of $F - C$
  do if $A - N(y)$ contains a path from $N(z)$ to $N(x)$ and
    $B - N(x)$ contains a path from $N(z)$ to $N(y)$
    then return *false*;
    end
  end
end
return *true*;

Berge(G):
(Step 1) **if** G has a clique-cutset, C
  then $G_1, G_2, \ldots, G_k =$ the output of Decomp2$(G; C)$;
    return Berge$(G_1) \wedge$ Berge$(G_2) \wedge \cdots \wedge$ Berge(G_k);
  end
(Step 2) **if** G has a rosette, centered at some vertex z
  then $G_1, G_2, \ldots, G_k, H =$ the output of Decomp3$(G; z)$;
    return Berge$(G_1) \wedge$ Berge$(G_2) \wedge \cdots \wedge$ Berge(G_k)
      $\wedge$Test(H)
      $\wedge$NoFB$(G_1; z) \wedge$ NoFB$(G_2; z) \wedge \ldots$ NoFB$(G_k; z)$;
  end
(Step 3) return Test(G);

Theorem 4.3. *Given any friendly graph G, algorithm* Berge(G) *returns* true *if and only if G is a Berge graph.*

Proof. We use induction on the number of vertices of G.

Case 1. Berge(G) *returns in Step* 1.

This case is trivial.

Case 2. Berge(G) *returns in Step* 2.

It is easy to see that H is friendly and bat-free; hence Lemma 4.1 guarantees that H is a Berge graph if and only if Test$(H) =$ *true*. Thus our task reduces to proving the following statements:

(i) If G contains an odd hole, then one of $G_1, \ldots, G_k, H$ contains an odd hole or else one of $G_1, \ldots, G_k$ contains a bat with head z.

(ii) If some G_i contains a bat with head z, then it contains an odd hole or a fragile bat with head z.

(iii) Every antihole of length at least seven in G is contained in one of $G_1, \ldots, G_k$.

(iv) If H contains an odd hole, then G contains an odd hole.

(v) H contains no antihole of length at least seven.

(vi) If some G_i contains a bat with head z, then G contains an odd hole.

Proof of (i). Let G contain an odd hole. If this hole contains at most one z-edge, then it is contained in one of $G_1, \ldots, G_k$. Thus we may assume that the hole has the form $e_1P_1e_2P_2 \ldots e_tP_t$ where $e_1, e_2, \ldots, e_t$ are z-edges, $t \geq 2$, and each P_i is a chordless path in G that contains no z-edge; since P_i does not pass through z, all its edges come from some $F_{j(i)}$, and so P_i is a chordless path in $G_{j(i)} - z$. If some P_i has

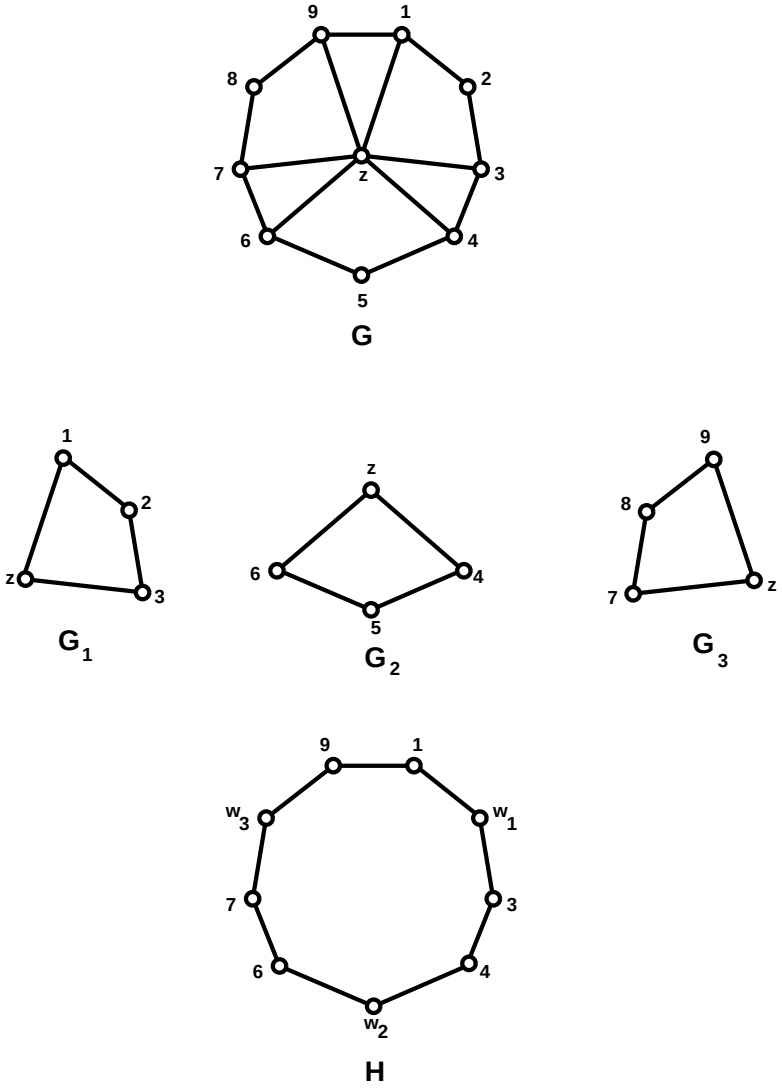


FIG. 4.1. DECOMP3 *illustrated.*

an odd number of edges, then the subgraph of G induced by $P_i \cup \{z\}$ is not bipartite; since it contains no triangle, it contains an odd hole and we are done. Thus we may assume that each P_i has an even number of edges. Now t is odd, and so the closed walk $e_1 w_{j(1)} e_2 w_{j(2)} \dots e_t w_{j(t)}$ in H has an odd length, $3t$; hence the subgraph H_0 of H induced by all the vertices of this walk is not bipartite. If H_0 contains no triangle, then it contains an odd hole and we are done. Thus we may assume that H_0 contains a triangle. It is easy to see that the triangle consists of some e_i and some w_j. However, then both endpoints of e_i belong to F_j, and so both P_{i-1} (with $P_0 = P_t$) and P_i are fully contained in F_j; the subgraph of G_j induced by $P_{i-1} \cup P_i \cup \{z\}$ contains a bat.

Proof of (ii). If some G_i contains a bat with head z, then—since $G_i * z$ is connected—Theorem 2.4 guarantees that G_i contains an odd hole (in which case we are done), or G_i contains a fragile bat with head z (in which case we are done

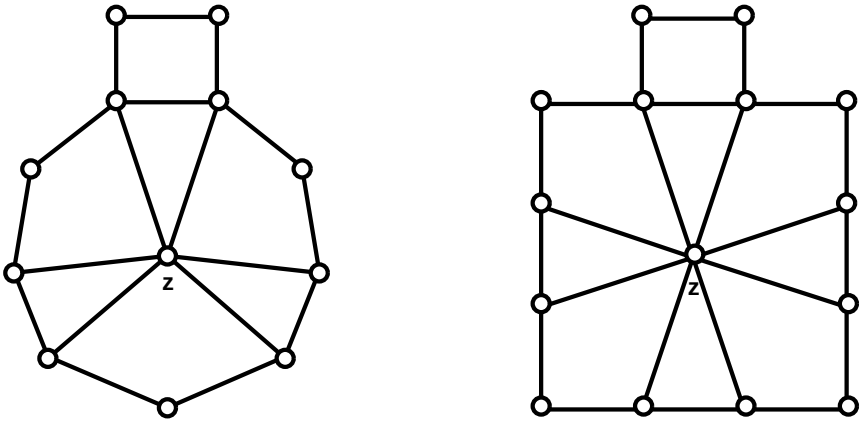


FIG. 4.2. *Clique-cutsets create counterexamples.*

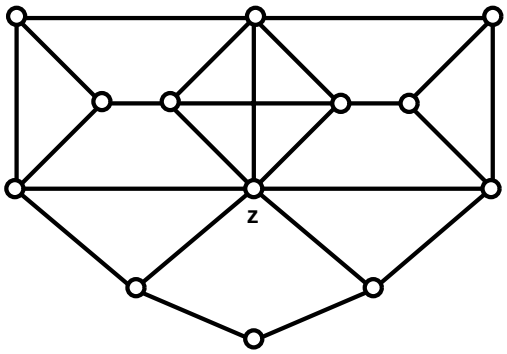


FIG. 4.3. *The significance of fragile bats.*

again), or G_i contains a clique-cutset C such that $z \in C$ and some component of G_i includes no neighbor of z (in which case C is a clique-cutset in G, contradicting our assumption that BERGE(G) does not return in Step 1).

Proof of (iii). Let A be an antihole of length at least seven in G. Observe that $z \notin A$ (since the neighborhood of z consists of vertex-disjoint cliques). If A is contained in one of $G_1, \ldots, G_k$, then we are done; else symmetry allows us to assume that A includes at least one vertex of G_1 and at least one vertex of G_2. Since A is connected, it follows that A contains an edge x_1x_2 with $x_1 \in G_1, x_2 \in G_2$. Note that x_1x_2 is a z-edge. In every antihole of length at least six, each edge is contained in a hole of length four; in particular, x_1x_2 is contained in a hole $x_1x_2y_2y_1$. Since the neighborhood of z consists of vertex-disjoint cliques and x_1, x_2 are adjacent to z, neither y_1 nor y_2 is adjacent to z. Hence $x_1y_1y_2x_2$ is a path in $G * z$ and yet its endpoints belong to distinct components of $G * z$, a contradiction.

Proof of (iv). Let H contain an odd hole. Since $w_1, w_2, \ldots, w_k$ have pairwise disjoint neighborhoods, this hole has the form

$$u_1v_1w_{j(1)}u_2v_2w_{j(2)} \ldots u_tv_tw_{j(t)}$$

such that u_1v_1, u_2v_2, $\ldots$, u_tv_t are z-edges in G and t is odd. Replacing each path $v_iw_{j(i)}u_{i+1}$ (here, $u_{t+1} = u_1$) by a chordless path P_i from v_i to u_{i+1} in $F_{j(i)}$, we obtain

a cycle C in $G - z$.

Note that each P_i, although chordless in $G * z$, may have chords in G. However, choosing C with as few edges as possible, we claim that each P_i is chordless in G or else G contains an odd hole (in which case we are done). To justify this claim, consider a chord xy of P_i in G. The assumption that C has as few edges as possible guarantees that neither x nor y equals v_i or u_{i+1} (if $x = v_i$, then we may replace v_i and P_i by y and $P[y, u_{i+1}]$); first choosing x as close to v_i as possible and then y as close to u_{i+1} as possible, we conclude that G_i contains a bat with head z. Since BERGE(G) did not return in Step 1, G has no clique-cutset; hence Theorem 2.4 guarantees that G contains an odd hole.

Now we may assume that each P_i is chordless in G. Let F denote the subgraph of G induced by the vertices of C; note that the edge-set of F partitions into edge-sets of $P_1, P_2, \ldots, P_t$ and edge-sets of pairwise vertex-disjoint cliques formed by z-edges.

If some P_i has an odd number of edges, then the subgraph of G induced by $P_i \cup \{z\}$ is not bipartite; since it contains no triangle, it contains an odd hole and we are done. Thus we may assume that each P_i has an even number of edges. Now C has an odd number of edges, and so F is not bipartite. If F contains no triangle, then it contains an odd hole and we are done. Thus we may assume that F contains a triangle.

Observe that at least one vertex of this triangle is an interior vertex of some P_i. Let u denote this vertex and let v, w denote the remaining two vertices of the triangle so that, proceeding from u along C in some cyclic order, we encounter first v and then w; let a denote the immediate predecessor of u in this order and let b denote the immediate successor of u.

Observe that uv, uw, vw are z-edges and au, ub are not; it follows that the five vertices u, v, w, a, b are distinct and (since the path P_i is chordless) the subgraph of G induced by them has no edges other than uv, uw, vw, au, ub. Hence Lemma 3.4 (with F in place of G) guarantees that F contains an odd hole (in which case we are done) or a bat. Thus we may assume that F contains a bat. The bat consists of a head x and a chordless path $a_1 a_2 \ldots a_m$ such that x is adjacent to a_1, a_i, a_{i+1}, a_m for some i with $3 \leq i \leq m - 3$ (and to no other a_j). Again, all three edges of the triangle $x a_i a_{i+1}$ must be z-edges. Since all the paths $P_1, P_2, \ldots, P_t$ are chordless in G, x and a_i cannot belong to the same P_j; hence the path $x a_1 a_2 \ldots a_i$ must involve at least one z-edge; similarly, the path $a_{i+1} a_{i+2} \ldots a_m x$ must involve at least one z-edge. Thus a_1 and a_m belong to the same component of $G * x$; since BERGE(G) did not return in Step 1, G has no clique-cutset; hence Theorem 2.4 guarantees that G contains an odd hole.

Proof of (v). Each vertex in an antihole of length at least seven has the property that its neighborhood contains a chordless path with three edges; no vertex of H has this property.

Proof of (vi). Let G_i contain a bat with head z. The two wing-tips of this bat belong to the same component of $G * z$ (this component is F_i). Now Theorem 2.4 guarantees that G contains an odd hole (in which case we are done) or G has a clique-cutset (contradicting our assumption that BERGE(G) does not return in Step 1).

CASE 3. BERGE(G) *returns in Step* 3.

Now G has neither a clique-cutset nor a rosette; hence Theorem 2.2 guarantees that G is bat-free or else G contains an odd hole; in turn, Lemma 4.1 guarantees that TEST(G) = *true* if and only if G is a Berge graph. □

THEOREM 4.4. BERGE(G) *runs in polynomial time.*

Proof. A clique-cutset in G can be found (or its absence established) in polynomial time by an algorithm designed by Whitesides [13]; evidently, a rosette in G can be found (or its absence established) in polynomial time; DECOMP2$(G; C)$, DECOMP3 $(G; z)$ and NOFB$(G_i; z)$ can be executed in polynomial time; by Lemma 4.2, TEST(H) in Step 2 and TEST(G) in Step 3 can be evaluated in polynomial time. Thus our task reduces to showing that the number $t(G)$ of nodes in the recursion tree of BERGE(G) does not exceed some polynomial in n, the number of vertices of the input graph G.

We are going to show that $t(G) \leq n^2$. More precisely, with $\overline{e}(G)$ standing for the number of edges in the complement of G, we claim that $t(G) \leq 2\overline{e}(G) + 1$. Justifying this claim by induction on $\overline{e}(G)$ amounts to proving that

$$2(\overline{e}(G_1) + \overline{e}(G_2) + \cdots + \overline{e}(G_k)) + k \leq 2\overline{e}(G)$$

whenever $G_1, \ldots, G_k$ are returned by DECOMP2$(G; C)$ or $G_1, \ldots, G_k, H$ are returned by DECOMP3$(G; z)$. A stronger inequality,

$$\overline{e}(G_1) + \overline{e}(G_2) + \cdots + \overline{e}(G_k) + \binom{k}{2} \leq \overline{e}(G),$$

follows from observing that (i) there is a clique C such that $G_i \cap G_j = C$ whenever $i \neq j$ and that (ii) for every choice of distinct i and j, there are nonadjacent vertices x_i and x_j with $x_i \in G_i - C$, $x_j \in G_j - C$. The first observation is trivial. (We have $C = \{z\}$ in case of DECOMP3.) To make the second observation in case of DECOMP2, choose any x_j in $G_j - C$. To make the second observation in case of DECOMP3, first recall that the subgraph N of G induced by the neighborhood of z consists of vertex-disjoint cliques and then note that (since G has no clique-cutset) each G_i meets at least two of these cliques; hence x_i and x_j can be chosen from two distinct cliques of N. □

Finally, Theorem 2.1 reduces the task of recognizing dart-free Berge graphs in polynomial time to the task of recognizing friendly Berge graphs in polynomial time: given any dart-free graph H, we can find in polynomial time a family $\mathbf{F}$ of pairwise vertex-disjoint friendly induced subgraphs of H such that H is a Berge graph if and only if all the members of $\mathbf{F}$ are Berge graphs. The procedure is obvious. To initialize, we set $\mathbf{F} = \{H\}$. While some member G of $\mathbf{F}$ has one of the properties (i), (ii), (iii) of Theorem 2.1, we replace it in $\mathbf{F}$ by its connected components (in case G is disconnected) or by graphs $G_1, G_2, \ldots, G_k$ such that $\overline{G}_1, \overline{G}_2, \ldots, \overline{G}_k$ are connected components of $\overline{G}$ (in case $\overline{G}$ is disconnected) or by $G - w$ such that w and some other vertex of G are adjacent twins. Upon termination, Theorem 2.1 guarantees that all the members of $\mathbf{F}$ are friendly.

Acknowledgment. We thank an anonymous referee for thoughtful comments that helped to improve the presentation of this paper.

REFERENCES

[1] C. BERGE, *Perfect graphs*, in Six Papers on Graph Theory, Indian Statistical Institute, Calcutta, India, 1963, pp. 1–21.

[2] C. BERGE, *Une application de la théorie des graphes à un problème de codage*, in Automata Theory, E.R. Caianiello, ed., Academic Press, New York, 1966, pp. 25–34.

[3] C. BERGE, *The history of the perfect graphs*, Southeast Asian Bull. Math., 20 (1996), pp. 5–10.

[4] C. BERGE, *Motivations and history of some of my conjectures*, Discrete Math., 165/166 (1997), pp. 61–70.

[5] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, PA, 1999.
[6] V. Chvátal and N. Sbihi, *Recognizing claw-free perfect graphs*, J. Combin. Theory Ser. B, 44 (1988), pp. 154–176.
[7] M. Conforti and M. R. Rao, *Structural properties and decomposition of linear balanced matrices*, Math. Programming, 55 (1992), pp. 129–168.
[8] J. Fonlupt and A. Zemirline, *A characterization of perfect K_4–$\{e\}$-free graphs*, Rev. Maghrébine Math., 1 (1992), pp. 167–202.
[9] J. Fonlupt and A. Zemirline, *A polynomial recognition algorithm for perfect K_4–$\{e\}$-free graphs*, Rev. Maghrébine Math., 2 (1993), pp. 1–26.
[10] K. R. Parathasarathy and G. Ravindra, *The strong perfect-graph conjecture is true for $K_{1,3}$-free graphs*, J. Combin. Theory Ser. B, 21 (1976), pp. 212–223.
[11] L. Sun, *Two classes of perfect graphs*, J. Combin. Theory Ser. B, 53 (1991), pp. 273–292.
[12] A. Tucker, *Coloring perfect K_4–$\{e\}$-free graphs*, J. Combin. Theory Ser. B, 42 (1987), pp. 313–318.
[13] S. H. Whitesides, *An algorithm for finding clique cut-sets*, Inform. Process. Lett., 12 (1981), pp. 31–32.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1339–1357

ALGORITHMS FOR A MINIMUM VOLUME ENCLOSING SIMPLEX IN THREE DIMENSIONS*

YUNHONG ZHOU† AND SUBHASH SURI‡

Abstract. We develop a combinatorial algorithm for determining a *minimum volume simplex* enclosing a set of points in $\mathcal{R}^3$. If the *convex hull* of the points has n vertices, then our algorithm takes $\Theta(n^4)$ time. Combining our exact but slow algorithm with a simple but crude approximation technique, we also develop an ε-approximation algorithm. The algorithm computes in $O(n + 1/\varepsilon^6)$ time a simplex whose volume is within $(1 + \varepsilon)$ factor of the optimal for any $\varepsilon > 0$.

Key words. shape approximation, bounding volumes, centroid

AMS subject classifications. 68Q25, 65D18, 52B10

PII. S0097539799363992

1. Introduction.

Approximating a geometric body by a combinatorially simpler shape is a problem with many applications. In computer graphics and robotics, for instance, checking for collision between complex geometric models is frequently a computational bottleneck. Therefore, collision detection packages commonly use simple bounding objects, such as axis-aligned bounding boxes [3, 14, 16], discrete oriented polytopes [9, 13], or spheres [10] to quickly eliminate pairs whose bounding objects are collision-free. Since intersecting simple bounding objects is computationally more efficient than checking objects themselves, this heuristic performs well in practice. (Suri, Hubbard, and Hughes [21] and Zhou and Suri [23] give theoretical proofs of these heuristics.) While in computer graphics and robotics rectangular boxes and spheres tend to be the approximating shapes of choice, in many applications, such as hyperspectral imaging and remote sensing, the natural object is a bounding *simplex.*

A key information processing task in hyperspectral imaging is to determine the parameters of a linear model for a set of multidimensional data vectors. These vectors could represent a set of spectral radiances or reflectances sampled finely in wavelength. A common model is to write these data vectors as a *convex* combination of certain extreme vectors, called *endmembers.* The *full unmixing* problem, as it is often called, is to determine both the endmembers and the associated proportions. The endmembers are defined as the corners of the *smallest volume simplex* enclosing the data vectors, and thus the problem of algorithmically fitting a simplex around a set of points has been considered by various researchers in Earth sciences [4, 5, 6, 20]. The method of Erlich and Full [5] works from "inside out"—it takes a set of extreme data points as initial guesses for the endmembers, then pushes the faces of the simplex out until all the data points are in the interior. Craig's method [4] works from outside in. Fuhrmann's method [6] uses a gradient descent approach. All of these methods are heuristics and do not always compute the smallest enclosing simplex. These heuris-

*Received by the editors November 11, 1999; accepted for publication (in revised form) January 18, 2002; published electronically May 29, 2002. The research in this paper was carried out when both the authors were at Washington University in St. Louis, and it was supported by National Science Foundation grants CCR-9901958 and ANI 9628190.

http://www.siam.org/journals/sicomp/31-5/36399.html

†Compaq Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301 (yzhou@pa.dec.com).

‡Computer Science Department, University of California, Santa Barbara, CA 93106 (suri@cs.ucsb.edu).

tics also lack any provable bound on the relative volume of the simplex found or the worst-case analysis of the running time.

Computing circumscribing or inscribed shapes of a given class, such as simplex, is also a fundamental problem in computational convexity. The paper by Gritzmann and Klee [8] gives a broad survey of results on the basic problems of computing, approximating, or measuring the smallest volume convex sets of a class containing a given convex body. In particular, Klee [11] proved the following useful *centroid property:* if T is a minimum volume simplex enclosing a convex body P, then the centroid of each facet of T touches P. Klee and Laskowski [12] used this centroid property to develop an efficient algorithm for computing the *smallest area triangle* containing a convex polygon in the plane. Their result was shortly improved to a linear time algorithm by O'Rourke et al. [17].

1.1. Related work. O'Rourke [18] has published a technical report claiming an $O(n^4)$ time algorithm for a minimum volume simplex in $\mathcal{R}^3$. Shortly after distributing the report in 1984, however, O'Rourke discovered a fatal flaw in a crucial lemma and has since retracted the claim [*personal communication, June* 1999]. The three-dimensional smallest simplex problem has since remained open.

Unlike the minimum volume simplex, efficient algorithms are known for other enclosing shapes such as bounding boxes, spheres, or ellipsoids in three, and sometimes in any fixed, dimensions. The axis-aligned bounding box is trivially computed in $O(n)$ time for any fixed dimension by computing the span of the enclosed polytope in each dimension and taking the cross product of these intervals. When the box can be arbitrarily oriented, O'Rourke [19] describes an $O(n^3)$ algorithm in three dimensions, where n is the number of vertices of the convex polytope.

The problem of computing the smallest ball enclosing a set of points is a classical one, dating back to Sylvester in 1857. In any fixed dimension, the smallest enclosing ball can be computed in optimal linear time using the fixed-dimensional linear programming algorithm of Megiddo [15]. Using an abstraction called "LP-type problems," the smallest enclosing ellipsoid can also be computed in (randomized) linear time [7]. Unlike spheres or ellipsoids, which have constant "combinatorial dimensions," simplices do not seem amenable to the abstract framework of "LP-type problems"—specifically, the simplex problem lacks the "locality" property exploited by the LP-type scheme.

A polynomial-time algorithm for the minimum volume simplex can be obtained by formulating the problem as an instance of *cubic programming* with linear constraints in 12 variables. The simplex is defined by four vertices and thus 12 variables. The objective function is the volume of the simplex, which is a cubic function of these variables. The condition that all points of the input lie inside the simplex can be written as linear constraints. If we adopt the model that a constant number of fixed-degree multinomials can be simultaneously solved in constant time, then this optimization problem can be solved in $O(n^6)$ time, since the feasible region defined by the constraints is a convex polytope in $\mathcal{R}^{12}$.

1.2. Our contribution. We develop an $O(n^4)$ time combinatorial algorithm for computing the smallest simplex enclosing a set of points in $\mathcal{R}^3$. The *algebraic complexity* of our algorithm is a significant improvement over the previous results. In all but one case, we need only to solve a *quadratic function in one variable,* which is trivially done. In the last case, we do require solving a degree ten polynomial; however, it is a single-variable polynomial, whose roots can be found efficiently by numerical methods. Building on the earlier work by Klee [11] and Vegter and Yap [22],

we further characterize the geometric properties of an optimal simplex and develop geometric procedures for computing the local optima of each combinatorial type. Let P be a convex polytope of n vertices in $\mathcal{R}^3$. (Given a set of points S, the minimum volume enclosing simplex for S is the same as the one enclosing the convex hull of S. Thus, we may assume from now on that the underlying body to be enclosed is a convex polytope.)

If T is the smallest simplex enclosing P, then each facet of T must touch P. The centroid property of Klee [11] says that each centroid p_i lies on P.[1] This alone is insufficient for determining the simplex, since the facets of T may have rotational degrees of freedom. In particular, each facet of T may contact P in a vertex, edge, or face of P. Let us denote these contact types as V (vertex), E (edge), or F (face). We may classify enclosing simplices according to the contact types of their facets. Since each facet of T has three possible contact types, altogether there are 15 such classes, which we call *combinatorial types.* Some example combinatorial types are $FFFF$, $FFEV$, and $VVEE$. Of these, only the type $FFFF$ is trivial, in the sense that given the contacts (actual faces of P), one can easily compute the unique simplex, if there is one. The other cases require nontrivial geometric reasoning to determine the smallest simplex.

We use a theorem of Vegter and Yap [22], which says that a locally minimal simplex does not have more than four degrees of freedom, to eliminate some of these combinatorial types. (A simplex's degree of freedom is the sum of its facets' degrees of freedom. A facet's degree of freedom is 0, 1, or 2, respectively, when its contact type is F, E, or V.) We add a new technical lemma to further reduce the possible combinatorial types to six plus the trivial type $FFFF$. For each of these six types, we develop geometric procedures to determine the unique simplex with the centroid property. In all but one case, our geometric procedures are highly efficient, requiring nothing more than solving a quadratic equation in one unknown. The combinatorial type $EEEE$ (all facets touching P at an edge),[2] however, proves to be quite difficult and requires computing the roots of a tenth degree polynomial. While this computation is necessarily expensive and requires numerical methods, in the standard RAM model of computation, its cost is $O(1)$.

Once we can compute the local optima for each combinatorial type in constant time, our simplex algorithm is enumerative. We test each quadruple of P (vertices, edges, and faces) as a possible contact type, check if it supports a simplex with the centroid property, and keep track of the smallest volume simplex found. The worst-case running time of this algorithm is $\Theta(n^4)$.

Next, we combine our slow but exact algorithm with a simple, fast, but crude approximation scheme to develop an ε-approximation algorithm. The algorithm computes in $O(n+\frac{1}{\varepsilon^6})$ time a simplex whose volume is within $(1+\varepsilon)$ factor of the optimal for any $\varepsilon > 0$.

2. Properties of centroids and locally optimal simplices. We assume throughout that P is a convex polytope of n vertices, and we want to compute a minimum volume simplex enclosing P. Let T denote a simplex, with vertices q_i, and facets t_i, for $1 \le i \le 4$. The facet t_i is *opposite* q_i, and let point p_i denote the *centroid* of t_i. Figure 1 illustrates these basic definitions. We recall from elementary geometry that if $\triangle abc$ is a triangle in $\mathcal{R}^3$ and d is the centroid of $\triangle abc$, then $d = (a+b+c)/3$.

[1] Recall that the centroid of a triangle (a, b, c) is the point $\frac{1}{3}(a+b+c)$.

[2] Vegter and Yap [22] give an example for which the type $EEEE$ simplex has local minimum.

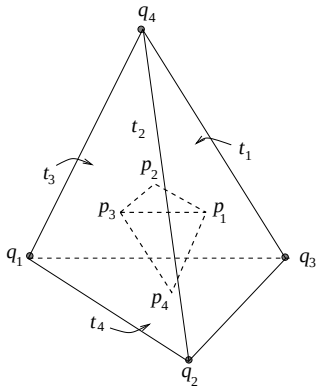


FIG. 1. *A simplex T with its vertices, faces, and centroids.*

Geometrically, d is the intersection of three median lines of $\triangle abc$. The distance from any vertex of the triangle to its opposite side is three times the distance from d to the side. For instance, the distance from a to bc is three times the distance from d to bc. The following lemma establishes two simple but useful facts relating the distances in a simplex.

LEMMA 2.1. *Let T be a simplex with vertices q_i, facets t_i, and centroids p_i, as above.*

1. *Given any pair i, j, where $1 \leq i < j \leq 4$, edges q_iq_j and p_ip_j are parallel; $q_iq_j = 3p_ip_j$; and vectors $\overrightarrow{q_iq_j}$ and $\overrightarrow{p_ip_j}$ are oppositely directed.*
2. *The triangle defined by three centroids is parallel to the fourth facet of T—for instance, $\triangle p_1p_2p_3$ is parallel to t_4. The distance between $\triangle p_1p_2p_3$ and t_4 is one third the distance between q_4 and t_4. The triangle $\triangle p_1p_2p_3$ is homothet to t_4, with area ratio $1/9$. By symmetry, the same holds for the other three facets.*

Proof.

1. Without loss of generality, consider $i = 1$ and $j = 2$. By the definition of centroid, we have $p_1 = (q_2 + q_3 + q_4)/3$ and $p_2 = (q_1 + q_3 + q_4)/3$. Therefore, $p_1 - p_2 = (q_2 - q_1)/3$. In vector notation, it is equivalent to $\overrightarrow{q_1q_2} = -3\overrightarrow{p_1p_2}$, which proves the claim.
2. In the triangle $\triangle q_2q_3q_4$, the distance from the centroid p_1 to edge q_2q_3 is $1/3$ the distance from q_4 and q_2q_3. Taking the vertical projection of these distances into the plane containing t_4, the distance from p_1 to t_4 is $1/3$ the distance from q_4 to t_4. A similar argument holds for p_2 and p_3. Thus, p_1, p_2, p_3 are equidistant from t_4, and therefore $\triangle p_1p_2p_3$ is parallel to t_4. Next, since $p_1p_2 \parallel q_1q_2$, $p_2p_3 \parallel q_2q_3$, and $p_3p_1 \parallel q_3q_1$, we conclude that triangles $\triangle p_1p_2p_3$ and $\triangle q_1q_2q_3$ are homothets. Furthermore, because their length ratio is $1/3$, the area ratio is $1/9$. ∎

If T is the smallest simplex enclosing P, then each facet of T must touch P. We will call T a *centroidal simplex* if all facet centroids of T lie on P. That is, T is centroidal if $p_i \in P$ for $i = 1, 2, 3, 4$. We will use the following theorem of Klee [11], which shows that a locally minimal enclosing simplex is always centroidal. (Klee proves this theorem for any dimension [11]. For the sake of completeness, we include

a proof of the centroid property in three dimensions in Appendix A.1.)

LEMMA 2.2 (centroid lemma [11]). *If T is an enclosing simplex of polytope P with locally minimal volume, then T is centroidal.*

Consider a locally minimal simplex T enclosing P. Each facet of T touches P, and this contact can be of one of three types: V (vertex), E (edge), or F (face). (That is, $T \cap P$ is a vertex, edge, or face of P.) We classify enclosing simplices according to the contact types of their facets. Since each facet of T has three possible contact types, altogether there are 15 such classes, which we call *combinatorial types.* These types are $FFFF$, $FFFE$, $FFFV$, $FFEE$, $FFEV$, $FFVV$, $FEEE$, $FEEV$, $FEVV$, $FVVV$, $EEEE$, $EEEV$, $EEVV$, $EVVV$, $VVVV$. For instance, type $FFEE$ means that simplex touches P in two faces and two edges.

A facet t of T has 0, 1, or 2 (rotational) *degrees of freedom* if the contact type of t is F, E, or V, respectively. A simplex's degree of freedom is the sum of its facets' degrees of freedom. Thus, a simplex of type $FFFF$ has zero degrees of freedom, while type $FEEV$ has four degrees of freedom. We will use the following result of Vegter and Yap [22], which says that a locally minimal simplex does not have more than four degrees of freedom.

LEMMA 2.3 (degrees of freedom [22]). *If T is an enclosing simplex of P with locally minimal volume, then T has at most four degrees of freedom.*

The "degrees of freedom" lemma allows us to exclude six of the 15 combinatorial types. We further eliminate two more types ($FFEV$ and $FFVV$) using a local perturbation argument, which leaves us seven classes of simplices to search.

LEMMA 2.4 (classification lemma). *Given a convex polytope P in $\mathcal{R}^3$, there always exists a minimum volume simplex enclosing P whose combinatorial type is one of the following seven: $FFFF$, $FFFE$, $FFFV$, $FFEE$, $FEEE$, $FEEV$, $EEEE$.*

Proof. Of the original 15 classes, we can eliminate the following six because they each have five or more degrees of freedom: $FEVV$, $FVVV$, $EEEV$, $EEVV$, $EVVV$, $VVVV$. Of the remaining nine, we show that classes $FFEV$ and $FFVV$ are nonessential, meaning that one can always find a minimum volume simplex in the remaining seven classes. Next, we show by a perturbation argument that the combinatorial types $FFEV$ and $FFVV$ are nonessential.

We perturb the vertices of P slightly to get a polytope $\hat{P}$; we use the notation $\hat{x}$ to denote the face x after the perturbation. The perturbed polytope satisfies the following condition: consider two facets $\hat{f}_1, \hat{f}_2$ and let ℓ be the line of intersection of the planes supporting these facets; then the line ℓ is *not parallel* to any plane determined by an edge $\hat{e}_3$ and a vertex $\hat{v}_4$ of $\hat{P}$, where $\hat{e}_3$ and $\hat{v}_4$ are disjoint from $\hat{f}_1, \hat{f}_2$.

We now observe that cases $FFEV$ and $FFVV$ cannot arise in this perturbed polytope. Indeed, if T is a simplex of type $FFVV$, whose facets t_1, t_2 are coplanar with facets $\hat{f}_1, \hat{f}_2$, and whose centroids p_3, p_4 are coincident with vertices $\hat{v}_3, \hat{v}_4$, then the line $\ell = t_1 \cap t_2$ is parallel to the line p_3p_4, which is parallel to $\hat{v}_3, \hat{v}_4$, and thus to the plane defined by $\hat{v}_4$ and any edge containing $\hat{v}_3$, a contradiction. Similarly, if T is a simplex of type $FFEV$, whose facets t_1, t_2 are coplanar with facets $\hat{f}_1, \hat{f}_2$, facet t_3 contains the edge $\hat{e}_3$, and the centroid p_4 is coincident with vertex $\hat{v}_4$, then the line $\ell = t_1 \cap t_2$ is parallel to the line p_3p_4. Because p_3 lies on $\hat{e}_3$, it follows that ℓ is parallel to the plane determined by edge $\hat{e}_3$ and vertex $\hat{v}_4$, again a contradiction.

If $T, \hat{T}$, respectively, are the minimum volume simplices enclosing P and $\hat{P}$, then $|\mathrm{Vol}(\hat{T}) - \mathrm{Vol}(T)| \to 0$, because $\hat{P}$ is an arbitrarily small perturbation of P. As we argued above, the minimum volume simplex for $\hat{P}$ cannot have combinatorial type $FFEV$ or $FFVV$. Let $\hat{T}'$ be the simplex enclosing P that is homothet to $\hat{T}$. Because

$\hat{P}$ is a small perturbation of P and $\hat{T}'$ is a homothetic copy of $\hat{T}$, $|\mathrm{Vol}(\hat{T}) - \mathrm{Vol}(\hat{T}')| \to 0$. Therefore, we have $|\mathrm{Vol}(\hat{T}') - \mathrm{Vol}(T)| \to 0$. In the limit of the perturbation, $\hat{T}'$ converges to a simplex T' enclosing P, with $\mathrm{Vol}(T') = \mathrm{Vol}(T)$. Furthermore, since the perturbation removes only degeneracies, the degrees of freedom can decrease only in going from $\hat{T}'$ to T'. Because T' is also the limit of $\hat{T}$ and $\hat{T}$ is centroidal, T' is centroidal with the same combinatorial type as $\hat{T}$. Thus, T' is a centroidal simplex enclosing P with the same volume as T, and its combinatorial type is neither $FFEV$ nor $FFVV$. □

3. Computing centroidal simplices. We now turn to the main contribution of our paper, which is to develop geometric procedures for computing a centroidal simplex of each combinatorial type. Specifically, given a set of four contact elements (vertices, edges, or faces of P), such that their contact types define a valid combinatorial type (Lemma 2.4), we show below how to find a centroidal simplex determined by these contacts. Observe that not every centroidal simplex necessarily encloses P, but checking for enclosure is relatively straightforward, and we discuss that in the next section. If the simplex T is defined by four vertices q_i, for $i = 1, 2, 3, 4$, where q_i has coordinates (x_i, y_i, z_i), then we have

$$A(T) = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{bmatrix} \qquad \text{and} \qquad \mathrm{Vol}(T) = \frac{1}{6}|\det(A)|.$$

The first type, $FFFF$, is quite trivial. Given four faces of P, we can easily compute the simplex defined by their supporting planes. (Intersecting three planes at a time gives the four vertices of the simplex.) We check in constant time if P lies in this simplex and, if so, compute its volume. The remaining six cases of Lemma 2.4 are less trivial.

3.1. Combinatorial type *FFFV*. Let f_1, f_2, f_3 be three faces of P and v a vertex of P. We want to compute the unique centroidal simplex determined by (f_1, f_2, f_3, v). (See Figure 1.) Let q_4 denote the intersection point of the three planes determined by f_1, f_2, f_3. Let $p_4 = v$, the centroid of facet t_4. The centroid p_1 of facet t_1 lies on the face f_1. Next, because p_1, p_4, respectively, are centroids of facets t_1, t_4, we have

$$p_1 - p_4 = \frac{(q_2 + q_3 + q_4)}{3} - \frac{(q_1 + q_2 + q_3)}{3} = \frac{(q_4 - q_1)}{3}.$$

Thus, $p_1p_4 \parallel q_1q_4$. We can now determine p_1 because it is the intersection of the plane determined by f_1 and the line passing through p_4 parallel to faces f_2 and f_3. Similarly, we can determine the centroids p_2 and p_3. Once we have all four centroids, we can compute the remaining three vertices of T from the following formula:

$$\begin{aligned} q_1 &= p_2 + p_3 + p_4 - 2p_1, \\ q_2 &= p_1 + p_3 + p_4 - 2p_2, \\ q_3 &= p_1 + p_2 + p_4 - 2p_3. \end{aligned}$$

With all four vertices of T known, we can compute its volume.

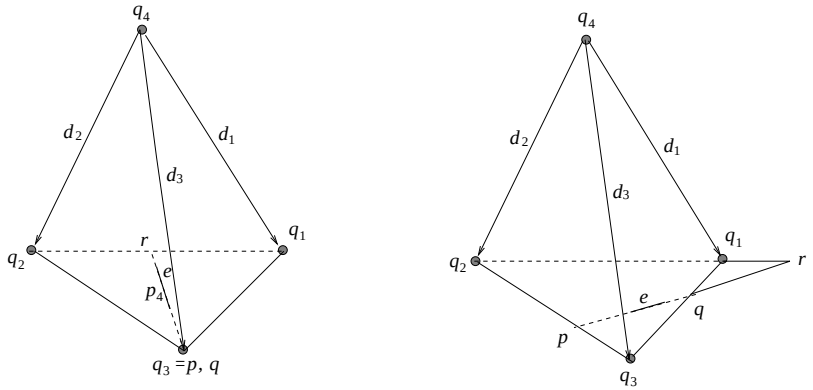


FIG. 2. *Type FFFE.* (a) *Case* 1. (b) *Case* 2.

3.2. Combinatorial type *FFFE*. Let f_1, f_2, f_3 be three faces of P and e an edge of P. We want to compute the unique centroidal simplex determined by (f_1, f_2, f_3, e). Let q_4 denote the intersection point of the three planes determined by f_1, f_2, f_3. Let p, q, r, respectively, denote the intersection points of line e with planes determined by faces f_1, f_2, f_3. (See Figure 2.) Let $\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3}$, respectively, be three unit vectors such that $\overrightarrow{d_i}$ is the direction of q_4q_i for $i = 1, 2, 3$. (The direction of q_4q_i can be obtained by calculating the intersection of its two adjacent faces, and the sign of $\overrightarrow{d_i}$ is chosen to ensure that P lies in the cone spanned by $\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3}$.)

We use a linear transform L to map q_4 to origin and $(\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3})$ into (x, y, z) basis. In the new coordinate system, $q_4 = (0, 0, 0)$, $p = (0, p_y, p_z)$, $q = (q_x, 0, q_z)$, $r = (r_x, r_y, 0)$. The edge e lies inside the cone spanned by $(\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3})$. We need to consider two cases.

1. **At least one of $p_y, p_z, q_x, q_z, r_x, r_y$ is zero.** Without loss of generality, assume that $p_y = 0$. Thus, p is collinear with q_4, q_3, and we have $p = q_3 = q$. Because the line containing e passes through q_3 and the centroid of $\triangle q_1q_2q_3$, this line is the median of $\triangle q_1q_2q_3$, and r is the middle point of q_1q_2. We can now determine $\triangle q_1q_2q_4$ because we know the vertex q_4, the directions $\overrightarrow{q_4q_1}, \overrightarrow{q_4q_2}$, and the middle point of q_1q_2. Thus, in this case, we know all the vertices of the simplex, and T is determined.
2. **None of $p_y, p_z, q_x, q_z, r_x, r_y$ is zero.** Of the three points p, q, r, two should lie on the boundary of the cone spanned by $(\overrightarrow{d_1}, \overrightarrow{d_2}, \overrightarrow{d_3})$, and one is outside. A point is outside the cone if it has one negative coordinate. Without loss of generality, we assume that p and q are on the boundary of the cone. Thus, $p_y, p_z, q_x, q_z > 0$ because they are nonzero. Suppose $q_1 = (l_x, 0, 0)$, $q_2 = (0, l_y, 0)$, and $q_3 = (0, 0, l_z)$. We get the following implications (the first because q_2, p, q_3 are collinear, and the second because q_3, q, q_1 are collinear):

$$\frac{p_y}{l_y} + \frac{p_z}{l_z} = 1 \implies l_y = \frac{p_y}{1 - \frac{p_z}{l_z}},$$
$$\frac{q_x}{l_x} + \frac{q_z}{l_z} = 1 \implies l_x = \frac{q_x}{1 - \frac{q_z}{l_z}}.$$

Thus,

$$l_x l_y l_z \;=\; \frac{p_y q_x l_z}{(1-\frac{p_z}{l_z})(1-\frac{q_z}{l_z})} \;=\; \frac{p_y q_x}{\lambda(1-p_z\lambda)(1-q_x\lambda)}, \text{ where } \lambda \;=\; \frac{1}{l_z}.$$

Because T has minimum volume, and the volume is proportional to $l_x l_y l_z$, the term $l_x l_y l_z$ should be locally minimal. We need to maximize the function $f(\lambda) = \lambda(1-p_z\lambda)(1-q_z\lambda)$. The derivative $f'(\lambda) = 3p_z q_z \lambda^2 - 2(p_z+q_z)\lambda + 1$ has the following two roots:

$$\lambda_1 = \frac{p_z + q_z - \sqrt{p_z^2 + q_z^2 - p_z q_z}}{3p_z q_z}, \quad \lambda_2 = \frac{p_z + q_z + \sqrt{p_z^2 + q_z^2 - p_z q_z}}{3p_z q_z}.$$

We claim that $0 < \lambda_1 < \min\{1/p_z, 1/q_z\} \le \lambda_2$. (This follows from straightforward algebra.) Thus, λ_2 is too big, and $\lambda = \lambda_1$ is the only choice. This yields

$$l_z \;=\; \frac{1}{\lambda} \;=\; p_z + q_z + \sqrt{p_z^2 + q_z^2 - p_z q_z}.$$

Once l_z is known, we can compute l_x and l_y, and the original coordinates of q_1, q_2, q_3 from the inverse transform L^{-1}.

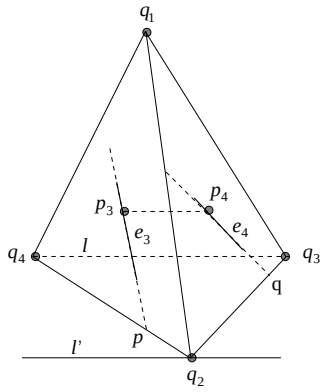


FIG. 3. *Combinatorial type FFEE.*

3.3. Combinatorial type *FFEE*. See Figure 3. Let f_1, f_2 be two faces of P and e_3, e_4 be two edges of P. We want to compute the unique centroidal simplex determined by (f_1, f_2, e_3, e_4). Assume that facets t_1, t_2 of T are determined by f_1, f_2, and facets t_3, t_4 are constrained by edges e_3, e_4. Let p_i be the centroid of facet t_i, and let line ℓ be the intersection of the planes determined by faces f_1 and f_2. Then, by Lemma 2.1, we first observe that $\ell \parallel p_3p_4$ and that $q_4q_3 = 3p_3p_4$. Let us use $c = q_4q_3$ to denote this quantity.

Let p, q, respectively, denote the intersection of plane through f_1 with lines through e_3 and e_4. Let $\overrightarrow{e_3}$, $\overrightarrow{e_4}$, and $\overrightarrow{\ell}$, respectively, denote the direction vectors of e_3, e_4, and ℓ. There exist scalars $\lambda_1, \lambda_2, \lambda_3$ such that $p_3 = p + \lambda_1 \overrightarrow{e_3}$, $p_4 = q + \lambda_2 \overrightarrow{e_4}$, and $p_3p_4 = p_4 - p_3 = \lambda_3 \overrightarrow{\ell}$. Thus,

$$p - q \;=\; -\lambda_1 \overrightarrow{e_3} + \lambda_2 \overrightarrow{e_4} - \lambda_3 \overrightarrow{\ell}. \tag{1}$$

The three equations in (1) have a unique solution for $(\lambda_1, \lambda_2, \lambda_3)$ if and only if $\det(\overrightarrow{e_3}, \overrightarrow{e_4}, \overrightarrow{\ell}) \neq 0$. The determinant $\det(\overrightarrow{e_3}, \overrightarrow{e_4}, \overrightarrow{\ell})$ is zero exactly when vectors $\overrightarrow{e_3}, \overrightarrow{e_4}, \overrightarrow{\ell}$ are dependent, which occurs exactly when e_3, e_4 are coplanar. We claim that the case when e_3, e_4 are coplanar is not needed—in Appendix A.2, we show that in this case there is an infinite family of centroidal simplices, whose limiting case is either $FFFF$ or $FFFE$. Thus, in the following, we assume that e_3, e_4 are not coplanar.

Since $\det(\overrightarrow{e_3}, \overrightarrow{e_4}, \overrightarrow{\ell}) \neq 0$, there is unique solution $(\lambda_1, \lambda_2, \lambda_3)$. With this solution, we know $p_3 = p + \lambda_1 \overrightarrow{e_3}$ and $p_4 = q + \lambda_2 \overrightarrow{e_4}$. By the centroid property, the distance from q_2 to plane t_2 is three times the distance from p_3 to t_2. We draw a line $\ell' \parallel \ell$ in the plane of t_1 whose distance to t_2 is three times the distance between p_3 and t_2. Now q_2 lies on ℓ' and $\ell' \parallel \ell$. We compute the distance between ℓ and ℓ', and let it be h. We want to fix the point q_2 on line ℓ' such that q_2p intersects with ℓ at point q_4, q_2q intersects with ℓ at point q_3, and $q_4q_3 = c$.

This can be done as follows. We choose ℓ' as the x-axis, and the line perpendicular to ℓ' as the y-axis such that line ℓ has y-coordinate h. Because q_4, p, q_2 are collinear, and q_2, q, q_3 are collinear, we have

$$(q_4q_2)_x = \frac{h * (pq_2)_x}{p_y} \qquad \text{and} \qquad (q_2q_3)_x = \frac{h * (q_2q)_x}{q_y}.$$

We plug these two equations into $q_4q_3 = c$, thus,

$$c = q_4q_3 = (q_4q_2)_x + (q_2q_3)_x = \frac{h * (pq_2)_x}{p_y} + \frac{h * (q_2q)_x}{q_y} = \frac{h * (q_{2x} - p_x)}{p_y} + \frac{h * (q_x - q_{2x})}{q_y},$$

and solve for q_{2x}:

$$q_{2x} = \frac{cp_yq_y/h + p_xq_y - q_xp_y}{q_y - p_y}.$$

Observe that we must have $p_y \neq q_y$, because otherwise $pq \parallel p_3p_4$, which would imply that e_3, e_4 are coplanar, a contradiction. From q_{2x}, we calculate q_2, and then get q_4, q_3, and finally $q_1 = 3p_3 - q_2 - q_4$. This completes the discussion of combinatorial type $FFEE$.

3.4. Combinatorial type *FEEV*. See Figure 4. Given a face f, two edges e_1, e_2, and a vertex v of P, we want to compute the unique centroidal simplex determined by (f, e_1, e_2, v). Let us suppose that $v = p_3$ is the centroid of facet t_3, while facet t_4 is coplanar with the face f. The remaining two facets of T are constrained by e_1 and e_2, with the centroid theorem ensuring that $p_1 \in e_1$ and $p_2 \in e_2$. Because the plane $p_1p_2p_3$ is parallel to t_4, if we draw a plane through p_3 and parallel to f, it will intersect e_1 at p_1 and e_2 at p_2.[3] Once p_1, p_2, p_3 are known, we can determine t_1, t_2, t_3 easily, as follows.

Since t_1 contains e_1 and $t_1 \parallel p_2p_3$, the plane containing t_1 is uniquely determined. Similarly, the plane containing t_2 is uniquely determined since t_2 contains e_2 and is parallel to p_3p_1. Once we know t_1, t_2, and t_4, this case reduces to combinatorial type $FFFV$, which has already been discussed.

[3]Observe that this construction fails if edges e_1 or e_2 are parallel to the plane $p_1p_2p_3$. This is a case of input degeneracy, since edge e_1 must be parallel to the plane determined by edge e_2 and vertex v_3. The symbolic perturbation argument of Lemma 2.4 can be used to show that this degenerate case need not be considered.

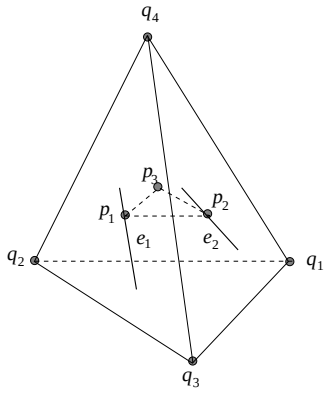


FIG. 4. *Combinatorial type FEEV.*

3.5. Combinatorial type *FEEE*. Given a face f, and three edges e_1, e_2, e_3 of P, we want to compute the unique centroidal simplex determined by (f, e_1, e_2, e_3). Let us suppose that f is coplanar with the facet t_4 of T. We first calculate the intersection points p, q, r, respectively, where the lines containing the edges e_1, e_2, e_3 intersect the plane containing f. (See Figure 5.) Next, choose any linear transform L that maps the plane containing f into the xy plane. Under this transform, the points p, q, r have new coordinates, and denote them as $p = (p_x, p_y, 0), q = (q_x, q_y, 0), r = (r_x, r_y, 0)$. Let $\overrightarrow{u}$ denote the direction of e_1 and scale it so that $\overrightarrow{u} = (u_x, u_y, 1)$. Similarly, let $\overrightarrow{v} = (v_x, v_y, 1)$ and $\overrightarrow{w} = (w_x, w_y, 1)$ denote the scaled direction of e_2 and e_3.

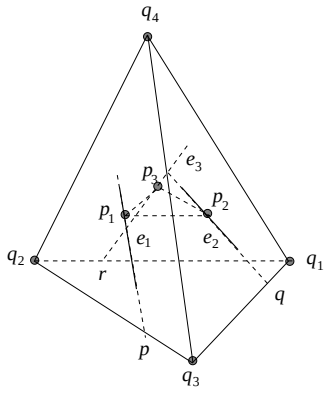


FIG. 5. *Combinatorial type FEEE.*

Since the centroid triangle $\triangle p_1, p_2, p_3$ is parallel to t_4, let h denote the common height of the three centroids p_1, p_2, p_3. The centroid p_i is also contained in edge e_i for $i = 1, 2, 3$. If h is fixed, the coordinates of the three centroids can be written as $p + hu, q + hv, r + hw$, respectively, and we know that the triangle formed by these centroids is a homothet of the base triangle t_4, with the area ratio $1/9$. Furthermore,

the height of the simplex is $3h$. Thus, the volume of the simplex can be expressed as

$$\mathrm{Vol}(T) = \frac{1}{3} * (3h) * 9 * \frac{1}{2} \begin{vmatrix} p_x + hu_x & p_y + hu_y & 1 \\ q_x + hv_x & q_y + hv_y & 1 \\ r_x + hw_x & r_y + hw_y & 1 \end{vmatrix} = \frac{9}{2} * (ah^3 + bh^2 + ch), \text{ where}$$

$$a = \begin{vmatrix} u_x & u_y & 1 \\ v_x & v_y & 1 \\ w_x & w_y & 1 \end{vmatrix}, \qquad b = \begin{vmatrix} p_x & u_y & 1 \\ q_x & v_y & 1 \\ r_x & w_y & 1 \end{vmatrix} + \begin{vmatrix} u_x & p_y & 1 \\ v_x & q_y & 1 \\ w_x & r_y & 1 \end{vmatrix}, \qquad c = \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}.$$

For the volume to be minimized, we need $3ah^2 + 2bh + c = 0$, which solves to

$$h_{1,2} = \frac{-b \pm \sqrt{b^2 - 3ac}}{3a}.$$

(If $a = 0$, then the minimization occurs for $h = -c/2b$.) For each of the two roots of this equation, we calculate the corresponding simplex and its volume. Once we know h, we get $p_1 = (p_x + hu_x,\ p_y + hu_y,\ h)$, $p_2 = (q_x + hv_x,\ q_y + hv_y,\ h)$, $p_3 = (r_x + hw_x,\ r_y + hw_y,\ h)$.

Because $q_2q_3 \parallel p_2p_3$ and q_2, p, q_3 are collinear, we can draw the line ℓ_1 passing through p and parallel to q_2q_3; similarly, we draw a line ℓ_2 passing through q and parallel to p_3p_1 and line ℓ_3 passing through r and parallel to p_1p_2. Now, $q_1 = \ell_2 \cap \ell_3$, $q_2 = \ell_3 \cap \ell_1$, $q_3 = \ell_1 \cap \ell_2$. The final vertex is determined as $q_4 = p_1 + p_2 + p_3 - 2/3(q_1 + q_2 + q_3)$. Finally, the transform L^{-1} maps the coordinates of q_1, q_2, q_3, q_4 back to the original space.

3.6. Combinatorial type *EEEE*. See Figure 6. Our final case is also the most complicated. Given four edges e_1, e_2, e_3, e_4 of P, we want to compute the unique centroidal simplex determined by them. We first use a linear transform L that maps the line containing e_4 to the x-axis. In the new coordinate system, e_i can be written as $o_i + \lambda v_i$ for $i = 1, 2, 3$. Suppose that t_4 makes an angle of θ with the xy-coordinate plane. Then, there is a rotation $R(\theta)$ around the x-axis that maps the plane containing t_4 into the xy plane. The orthogonal transform $R(\theta)$ can be written in matrix notation as

$$R(\theta) = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{vmatrix}.$$

Each edge e_i, for $i = 1, 2, 3$, has new coordinates $R(\theta)(o_i + \lambda v_i) = o_i(\theta) + \lambda v_i(\theta)$. The centroid p_i lies on e_i, and the three centroids p_1, p_2, p_3 have the same height (z-coordinate). Letting this height be h, we have

$$(o_i(\theta) + \lambda_i v_i(\theta))_z = o_{iz}(\theta) + \lambda_i v_{iz}(\theta) = h \Rightarrow \lambda_i = \frac{h - o_{iz}(\theta)}{v_{iz}(\theta)} \text{ for } i = 1, 2, 3.$$

The volume of the simplex T can be written as

$$\mathrm{Vol}(T) = \frac{1}{3} * (3h) * 9 * \frac{1}{2} \begin{vmatrix} o_{1x}(\theta) + \lambda_1 v_{1x}(\theta) & o_{1y}(\theta) + \lambda_1 v_{1y}(\theta) & 1 \\ o_{2x}(\theta) + \lambda_2 v_{2x}(\theta) & o_{2y}(\theta) + \lambda_2 v_{2y}(\theta) & 1 \\ o_{3x}(\theta) + \lambda_3 v_{3x}(\theta) & o_{3y}(\theta) + \lambda_3 v_{3y}(\theta) & 1 \end{vmatrix}$$

$$
\begin{aligned}
&= \frac{9}{2} * h * \begin{vmatrix} o_{1x}(\theta) + \frac{h-o_{1z}(\theta)}{v_{1z}(\theta)} v_{1x}(\theta) & o_{1y}(\theta) + \frac{h-o_{1z}(\theta)}{v_{1z}(\theta)} v_{1y}(\theta) & 1 \\ o_{2x}(\theta) + \frac{h-o_{2z}(\theta)}{v_{2z}(\theta)} v_{2x}(\theta) & o_{2y}(\theta) + \frac{h-o_{2z}(\theta)}{v_{2z}(\theta)} v_{2y}(\theta) & 1 \\ o_{3x}(\theta) + \frac{h-o_{3z}(\theta)}{v_{3z}(\theta)} v_{3x}(\theta) & o_{3y}(\theta) + \frac{h-o_{3z}(\theta)}{v_{3z}(\theta)} v_{3y}(\theta) & 1 \end{vmatrix} \\
&= \frac{9}{2} \frac{h}{v_{1z}(\theta) v_{2z}(\theta) v_{3z}(\theta)} \\
&\quad \times \begin{vmatrix} o_{1x}(\theta)v_{1z}(\theta) + (h - o_{1z}(\theta))v_{1x}(\theta) & o_{1y}(\theta)v_{1z}(\theta) + (h - o_{1z}(\theta))v_{1y}(\theta) & v_{1z}(\theta) \\ o_{2x}(\theta)v_{2z}(\theta) + (h - o_{2z}(\theta))v_{2x}(\theta) & o_{2y}(\theta)v_{2z}(\theta) + (h - o_{2z}(\theta))v_{2y}(\theta) & v_{2z}(\theta) \\ o_{3x}(\theta)v_{3z}(\theta) + (h - o_{3z}(\theta))v_{3x}(\theta) & o_{3y}(\theta)v_{3z}(\theta) + (h - o_{3z}(\theta))v_{3y}(\theta) & v_{3z}(\theta) \end{vmatrix} \\
&= \frac{9}{2} \frac{h}{v_{1z}(\theta) v_{2z}(\theta) v_{3z}(\theta)} [a(\theta)h^2 + b(\theta)h + c(\theta)],
\end{aligned}
$$

where

$$
\begin{aligned}
a(\theta) &= \begin{vmatrix} v_{1x}(\theta) & v_{1y}(\theta) & v_{1z}(\theta) \\ v_{2x}(\theta) & v_{2y}(\theta) & v_{2z}(\theta) \\ v_{3x}(\theta) & v_{3y}(\theta) & v_{3z}(\theta) \end{vmatrix} = \begin{vmatrix} v_{1x} & v_{1y} & v_{1z} \\ v_{2x} & v_{2y} & v_{2z} \\ v_{3x} & v_{3y} & v_{3z} \end{vmatrix} \\
b(\theta) &= \begin{vmatrix} v_{1x}(\theta) & o_{1y}(\theta)v_{1z}(\theta) - o_{1z}(\theta)v_{1y}(\theta) & v_{1z}(\theta) \\ v_{2x}(\theta) & o_{2y}(\theta)v_{2z}(\theta) - o_{2z}(\theta)v_{2y}(\theta) & v_{2z}(\theta) \\ v_{3x}(\theta) & o_{3y}(\theta)v_{3z}(\theta) - o_{3z}(\theta)v_{3y}(\theta) & v_{3z}(\theta) \end{vmatrix} \\
&\quad + \begin{vmatrix} o_{1x}(\theta)v_{1z}(\theta) - o_{1z}(\theta)v_{1x}(\theta) & v_{1y}(\theta) & v_{1z}(\theta) \\ o_{2x}(\theta)v_{2z}(\theta) - o_{2z}(\theta)v_{2x}(\theta) & v_{2y}(\theta) & v_{2z}(\theta) \\ o_{3x}(\theta)v_{3z}(\theta) - o_{3z}(\theta)v_{3x}(\theta) & v_{3y}(\theta) & v_{3z}(\theta) \end{vmatrix} \\
&= \begin{vmatrix} v_{1x} & o_{1y}v_{1z} - o_{1z}v_{1y} & v_{1z}(\theta) \\ v_{2x} & o_{2y}v_{2z} - o_{2z}v_{2y} & v_{2z}(\theta) \\ v_{3x} & o_{3y}v_{3z} - o_{3z}v_{3y} & v_{3z}(\theta) \end{vmatrix} + \begin{vmatrix} o_{1x}v_{1z}(\theta) - o_{1z}(\theta)v_{1x} & v_{1y} & v_{1z} \\ o_{2x}v_{2z}(\theta) - o_{2z}(\theta)v_{2x} & v_{2y} & v_{2z} \\ o_{3x}v_{3z}(\theta) - o_{3z}(\theta)v_{3x} & v_{3y} & v_{3z} \end{vmatrix} \\
c(\theta) &= \begin{vmatrix} o_{1x}(\theta)v_{1z}(\theta) - o_{1z}(\theta)v_{1x}(\theta) & o_{1y}(\theta)v_{1z}(\theta) - o_{1z}(\theta)v_{1y}(\theta) & v_{1z}(\theta) \\ o_{2x}(\theta)v_{2z}(\theta) - o_{2z}(\theta)v_{2x}(\theta) & o_{2y}(\theta)v_{2z}(\theta) - o_{2z}(\theta)v_{2y}(\theta) & v_{2z}(\theta) \\ o_{3x}(\theta)v_{3z}(\theta) - o_{3z}(\theta)v_{3x}(\theta) & o_{3y}(\theta)v_{3z}(\theta) - o_{3z}(\theta)v_{3y}(\theta) & v_{3z}(\theta) \end{vmatrix} \\
&= \begin{vmatrix} o_{1x}v_{1z}(\theta) - o_{1z}(\theta)v_{1x} & o_{1y}v_{1z} - o_{1z}v_{1y} & v_{1z}(\theta) \\ o_{2x}v_{2z}(\theta) - o_{2z}(\theta)v_{2x} & o_{2y}v_{2z} - o_{2z}v_{2y} & v_{2z}(\theta) \\ o_{3x}v_{3z}(\theta) - o_{3z}(\theta)v_{3x} & o_{3y}v_{3z} - o_{3z}v_{3y} & v_{3z}(\theta) \end{vmatrix}.
\end{aligned}
$$

The preceding calculations have used two key properties of the rotation $R(\theta)$. One is the x-coordinate of any point is invariant under $R(\theta)$ and the other that $R(\theta)$ restricted to the yz-plane is orthogonal. We can write the volume function as

$$
\mathrm{Vol}(T) = \frac{9}{2} \left[A(\theta)h^3 + B(\theta)h^2 + C(\theta)h \right], \text{ where}
$$

$$
A(\theta) = \frac{a(\theta)}{s(\theta)}, \quad B(\theta) = \frac{b(\theta)}{s(\theta)}, \quad C(\theta) = \frac{c(\theta)}{s(\theta)}, \quad s(\theta) = v_{1z}(\theta)v_{2z}(\theta)v_{3z}(\theta).
$$

The volume function $\mathrm{Vol}(T) = V(\theta, h)$ has two parameters, θ and h. For the volume to be locally minimal, we need that

$$
\frac{\partial V(\theta, h)}{\partial \theta} = 0, \qquad \frac{\partial V(\theta, h)}{\partial h} = 0.
$$

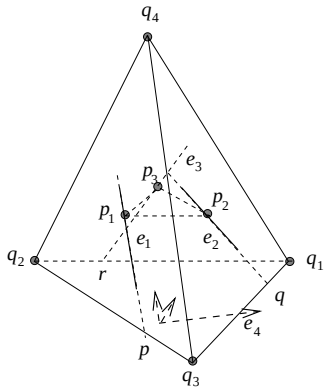


FIG. 6. *Type EEEE.*

The above equations solve to

$$3A(\theta)h^2 + 2B(\theta)h + C(\theta) = 0 \qquad \text{and} \qquad A'(\theta)h^2 + B'(\theta)h + C'(\theta) = 0.$$

We plug $a(\theta), b(\theta), c(\theta), s(\theta)$ into $A(\theta), B(\theta), C(\theta)$ and use the fact that $a(\theta) = a$ is a constant. Then

$$a_1h^2 + b_1h + c_1 = 0, \qquad a_2h^2 + b_2h + c_2 = 0,$$

where $a_1 = 3a, b_1 = 2b(\theta), c_1 = c(\theta), a_2 = as'(\theta), b_2 = [b(\theta)s'(\theta) - b'(\theta)s(\theta)], c_2 = [c(\theta)s'(\theta) - c'(\theta)s(\theta)]$.

We now need to discuss two cases depending on whether or not $a_1b_2 - a_2b_1 = 0$.

1. **Case 1. $a_1b_2 - a_2b_1 \neq 0$.** In this case, h is uniquely determined by the two equations above. We get $h = -(a_1c_2 - a_2c_1)/(a_1b_2 - a_2b_1)$. If we plug the expression of h into the first equation, and simplify a little, we get the following:

$$(a_1c_2 - a_2c_1)^2 + a_1c_1b_2^2 + a_2c_2b_1^2 = b_1b_2(a_1c_2 + a_2c_1).$$

Replacing $a_i, b_i, c_i, i = 1, 2$ by original $a, b(\theta), c(\theta), s(\theta)$ gives

$$\begin{aligned} a\left[3c'(\theta)s(\theta) - 2c(\theta)s'(\theta)\right]^2 &+ 3c(\theta)\left[b(\theta)s'(\theta) - b'(\theta)s(\theta)\right]^2 \\ &+ 4b(\theta)^2s'(\theta)\left[c(\theta)s'(\theta) - c'(\theta)s(\theta)\right] \\ = 2b(\theta)\left[b(\theta)s'(\theta) - b'(\theta)s(\theta)\right]&\left[4c(\theta)s'(\theta) - 3c'(\theta)s(\theta)\right]. \end{aligned}$$

In this expression, $b(\theta), c(\theta), s(\theta)$ are homogeneous polynomials of variables $\cos\theta$ and $\sin\theta$. The function $b(\theta)$ has degree 1, $c(\theta)$ has degree 2, and $s(\theta)$ has degree 3. Because these functions are trigonometric, their derivatives have the same degree as the functions themselves. Thus, the preceding equation is a homogeneous polynomial of degree 10 with variables $\cos\theta$ and $\sin\theta$, which can be converted into a tenth degree polynomial with single variable $\tan\theta$. While there is no analytic formula for the roots of a degree 10 polynomial, we can use numerical methods to solve for its roots. All the real roots need

to be checked separately. Once we have a root θ, we can compute

$$h = -\frac{2c(\theta)s'(\theta) - 3c'(\theta)s(\theta)}{b(\theta)s'(\theta) - 3b'(\theta)s(\theta)}.$$

Once we have h and θ, we calculate $\lambda_i = (h - o_{iz}(\theta))/v_{iz}(\theta)$, and t_i's centroid $p_i = o_i + \lambda_i v_i$, for $i = 1, 2, 3$. Now, the facet t_1 is determined by the plane that contains e_1 and is parallel to p_2p_3, t_2's plane contains e_2 and is parallel to p_3p_1, t_3's plane contains e_3 and is parallel to p_1p_2, and t_4's plane contains e_4 and is parallel to the plane containing the triangle $\triangle p_1p_2p_3$. The simplex T is now determined.

2. **Case 2. $a_1b_2 - a_2b_1 = 0$.** By plugging in the expressions for a_i, b_i, we get the equivalent equation

$$3b'(\theta)s(\theta) - b(\theta)s'(\theta) = 0.$$

This is actually a degree 4 polynomial in $\tan\theta$. We may compute the roots of this polynomial analytically or using numerical methods. Given a root θ, we can calculate h by solving

$$3ah^2 + 2b(\theta)h + c(\theta) = 0.$$

Once we have h and θ, we calculate the simplex T as in Case 1. This completes the discussion of type $EEEE$.

4. An exact algorithm. We can now describe our algorithm for computing the minimum volume simplex T enclosing P. Let L be a list containing all the vertices, edges, and faces of P. We consider all 4-tuples of L. If the combinatorial type of a tuple is not among the seven types listed in Lemma 2.4, we ignore the tuple and move on to the next tuple. Otherwise, we use the appropriate geometric procedure described in the preceding section to compute the centroidal simplex determined by this tuple. Once the centroidal simplex T is found, we also need to check if $P \subset T$.

The enclosure check is easily done in $O(1)$ time for all cases except $FFFV$ and $FEEV$. Observe that, in the remaining cases, all the contacts are of type F or E. Given a contact type F or E, we can locally decide if the facet of T determined by that contact has P on the correct side. For each face of P, store the outward normal—the one pointing away from P. If a face $f \in P$ supports the facet t of T, then it suffices to check that the outward normals of f and t point in the same direction. Given an edge contact e supporting a facet t, it suffices to check that the normal to t is a convex combination of the normals to the two faces incident to e.

On the other hand, a vertex v of P can have arbitrarily high degree (number of faces incident to it). Thus, checking if a facet $t \in T$ touching v lies entirely to one side of P can be expensive. However, since the only combinatorial types involving vertex contacts are $FFFV$ and $FEEV$, we can afford to spend even linear time per vertex contact. Given a vertex v, there are $O(n^3)$ triples of face or edge contacts to consider in conjunction with v. For each such combinatorial type, we spend $O(deg(v))$ time checking simplex enclosure, where $deg(v)$ is the number of faces or edges incident to v. The sum over all vertices v is

$$O(n^3) \sum_{v \in P} deg(v) \;=\; O(n^4),$$

since, by Euler's theorem, the sum of vertex degrees is $O(n)$. Thus, we can summarize our main result in the following theorem.

THEOREM 4.1. *Given a convex polytope P of n vertices in $\mathcal{R}^3$, we can determine the minimum volume simplex enclosing P in $\Theta(n^4)$ worst-case time.*

Next, we develop a faster approximation algorithm for the minimum enclosing simplex.

5. An approximation algorithm. One can easily compute in $O(n)$ time a simplex whose volume is within a constant factor of the optimal, using the following result of Barequet and Har-Peled [2]. Given a geometric object Q, and a positive number c, we use cQ to denote the uniform scaling of Q by factor c.

LEMMA 5.1 (see [2]). *Given a three-dimensional convex polytope P of n vertices, one can compute in $O(n)$ time an arbitrarily oriented rectangular box B_{apr} enclosing P such that $\text{Vol}(B_{\text{apr}}) \leq c_1 \text{Vol}(P)$ for $c_1 \geq 6\sqrt{6}$. Furthermore, P contains a translated copy of the box $\frac{1}{c_2}B_{\text{apr}}$, where $c_2 \geq 107$ is a positive number.*

A minimum volume simplex containing B_{apr} is a constant factor approximation of the optimal simplex containing P. While this is a simple approximation algorithm, the approximation factor is quite poor. We now turn this scheme into an ε-approximation scheme, following the approach of [2]. The main idea is to replace the polytope P with a *low-complexity* convex polytope P' enclosing P, and to compute an exact minimum volume simplex of P', using the algorithm of section 4. The closeness of P' to P will ensure that the optimal simplex enclosing P' is within ε factor of the optimal simplex of P. We begin with some definitions.

We denote a rectangular box B in $\mathcal{R}^3$ as a triple of orthogonal vectors (b_1, b_2, b_3), where b_1, b_2, b_3 are vectors having the directions and sizes of the edges of B. The *grid* generated by B is the set of points $\text{Grid}(B) = \{i_1b_1 + i_2b_2 + i_3b_3 | i_1, i_2, i_3 \in Z\}$. Given a grid G, the (i_1, i_2, i_3)th cell of G is $B^G_{(i_1,i_2,i_3)} = \{x_1b_1 + x_2b_2 + x_3b_3 | i_j \leq x_j < i_j + 1, j = 1, 2, 3\}$. We will approximate the polytope P by a set of grid points. Let P_G be the corners of all those cells in G that contain a vertex of P, and let P' be the convex hull of P_G. We can describe our ε-approximation algorithm now.

Algorithm ε-APPROXIMATION ENCLOSING TETRAHEDRON (P).
1. Compute a box B_{apr} enclosing P as in Lemma 5.1.
2. Compute the convex polytope $P' = CH(P_G)$, where $G = \text{Grid}(\frac{1}{2}B_\varepsilon)$, and B_ε is a translated copy of $\frac{\varepsilon}{428}B_{apr}$ centered at the origin.
3. Compute $T'_{\text{apr}} = T_{\text{opt}}(P')$ by our exact algorithm given in section 4.
4. Compute T_{apr} homothet to T'_{apr} such that every face of T_{apr} touches P.

end Algorithm

Let us first analyze the time complexity of this algorithm. Since $P' \subseteq P \oplus B_\varepsilon \subseteq B_{\text{apr}} \oplus B_\varepsilon$, and since the latter box contains at most $k = 856/\varepsilon + 3$ grid points in each orthogonal direction of the box, P' contains at most k^3 grid points. By a result of Andrews [1], the convex polytope P' has $O(k^{3/2}) = O(1/\varepsilon^{\frac{3}{2}})$ vertices. The dominant step of the algorithm is step 3, which takes $O((1/\varepsilon^{\frac{3}{2}})^4) = O(1/\varepsilon^6)$ time. The total time is therefore $O(n + 1/\varepsilon^6)$.

Finally, we show that the $\text{Vol}(T_{\text{apr}})$ is at most $(1 + \varepsilon)$ times the volume of the optimal simplex enclosing P. By Lemma 5.1, the approximate box B_{apr} is such that a translated copy of $\frac{1}{107}B_{\text{apr}}$ is contained in P. Thus, a uniform scaling by factor $\varepsilon/4$ gives that a translated copy of $\frac{\varepsilon}{428}B_{\text{apr}} = B_\varepsilon$ lies inside $\frac{\varepsilon}{4}P$. Since $P' \subseteq P \oplus B_\varepsilon$, it follows that $P' \subseteq P \oplus \frac{\varepsilon}{4}P$—that is, P' lies inside the Minkowski sum of P and

some translation of $\frac{\varepsilon}{4}P$. Thus, if $T_{\text{opt}}(P)$ is an optimal enclosing simplex of P, then $(1+\frac{\varepsilon}{4})T_{\text{opt}}(P)$ is an enclosing tetrahedron of P'. Because $T'_{\text{apr}} = T_{\text{opt}}(P')$ is an enclosing simplex of P' with minimal volume, we get

$$\text{Vol}(T_{\text{apr}}) \le \text{Vol}(T'_{\text{apr}}) = \text{Vol}(T_{\text{opt}}(P')) \le \left(1+\frac{\varepsilon}{4}\right)^3 \text{Vol}(T_{\text{opt}}) \le (1+\varepsilon)\text{Vol}(T_{\text{opt}}).$$

This completes the analysis of our approximation algorithm, and we summarize the main result in the following theorem.

THEOREM 5.2. *Given a convex polytope of v vertices in $\mathcal{R}^3$, and a real parameter $0 < \varepsilon < 1$, we can compute in $O(n + 1/\varepsilon^6)$ time a simplex enclosing P with volume at most $(1+\varepsilon)$ times the volume of an optimal enclosing simplex.*

6. Concluding remarks. We have described an exact algorithm for determining a *minimum volume simplex* enclosing a set of points in $\mathcal{R}^3$. If the *convex hull* of the points has n vertices, then our algorithm takes $\Theta(n^4)$ time. Combining our exact but slow algorithm with a simple but crude approximation technique, we also develop an ε-approximation algorithm. The algorithm computes in $O(n + 1/\varepsilon^6)$ time a simplex whose volume is within $(1+\varepsilon)$ factor of the optimal for any $\varepsilon > 0$. We are currently implementing the exact algorithm. While its overall complexity, both the running time as well as implementation effort, may be too large for any practical use, we hope to use it as a benchmark for developing fast and simple approximation algorithms.

A. The appendix. In this appendix, we include the proofs of various technical lemmas. We first begin with a proof of the centroid property. This property was established by Klee [11] for any dimension; we include a proof in three dimensions for completeness.

A.1. Proof of centroid property. Without loss of generality, assume that facet t_4 of T violates the centroid property. Let $Q = t_4 \cap P$ be the intersection of t_4 with P, where Q is a point, segment, or a convex polygon. Clearly, $p_4 \notin Q$. Since Q is convex, there exists a line ℓ in the plane of t_4 that separates p_4 and Q. Consider the point $p_4(\lambda) = \frac{1}{1+\lambda}(p_4 + \lambda q_4)$ on the segment p_4q_4 for $\lambda \in (0,1)$. Let $t_4(\lambda)$ denote the plane containing the point $p_4(\lambda)$ and the line ℓ. The line through q_4 and q_i intersects the plane $t_4(\lambda)$ at the point $q_i(\lambda)$ for $i = 1, 2, 3$.

Because volumetric ratios are invariant under nonsingular affine transformations, we may assume without loss of generality that ℓ is X-axis, $p_4 = (0,1,0)$, and $q_4 = (0,0,1)$. Then $p_4(\lambda) = (0, \frac{1}{1+\lambda}, \frac{\lambda}{1+\lambda})$ and plane $t_4(\lambda)$ is given by the equation $z = \lambda y$. Suppose that $q_i(\lambda) = (1-\tau_i)q_4 + \tau_i q_i = (\tau_i q_{ix}, \tau_i q_{iy}, 1-\tau_i)$; then $1 - \tau_i = \lambda\tau_i q_{iy}$, or $\tau_i = (1+\lambda q_{iy})^{-1}$, for $i = 1, 2, 3$. When λ is small enough, each τ_i is positive. The volume of tetrahedron $T(\lambda)$ is $1/6$ times the absolute value of the determinant

$$D(\lambda) = \begin{vmatrix} \tau_1 q_{1x} & \tau_1 q_{1y} & 1-\tau_1 & 1 \\ \tau_2 q_{2x} & \tau_2 q_{2y} & 1-\tau_2 & 1 \\ \tau_3 q_{3x} & \tau_3 q_{3y} & 1-\tau_3 & 1 \\ 0 & 0 & 1 & 1 \end{vmatrix} = \begin{vmatrix} \tau_1 q_{1x} & \tau_1 q_{1y} & -\tau_1 & 0 \\ \tau_2 q_{2x} & \tau_2 q_{2y} & -\tau_2 & 0 \\ \tau_3 q_{3x} & \tau_3 q_{3y} & -\tau_3 & 0 \\ 0 & 0 & 1 & 1 \end{vmatrix}$$

$$= (\tau_1\tau_2\tau_3) \begin{vmatrix} q_{1x} & q_{1y} & -1 \\ q_{2x} & q_{2y} & -1 \\ q_{3x} & q_{3y} & -1 \end{vmatrix}.$$

Since τ_i's are all 1 when $\lambda = 0$, the third determinant is equal to $D(0)$. Thus,

$$\frac{\text{Vol}(T(\lambda))}{\text{Vol}(T)} = \tau_1\tau_2\tau_3 = \prod_{i=1}^{3}(1+\lambda q_{iy})^{-1} = \left[\prod_{i=1}^{3}(1+\lambda q_{iy})\right]^{-1}$$
$$= \left[1 + \left(\sum_{i=1}^{3} q_{iy}\right)\lambda + o(\lambda)\right]^{-1} = 1 - \left(\sum_{i=1}^{3} q_{iy}\right)\lambda + o(\lambda).$$

Since p_4 is the centroid of $\triangle q_1q_2q_3$, we have $\sum_{i=1}^{3} q_{iy} = 3p_{4y} = 3$. Thus,

$$\frac{\text{Vol}(T(\lambda))}{\text{Vol}(T)} = 1 - 3\lambda + o(\lambda).$$

For small enough λ, $T(\lambda)$ is also an enclosing simplex of P, but the volume of $T(\lambda)$ is strictly less than the volume of T, which conflicts the assumed minimality of T. Thus, the simplex T must be centroidal.

Next, we give a proof for the case of coplanar e_3, e_4 in combinatorial type $FFEE$, which we claimed in section 3.3 that we did not need.

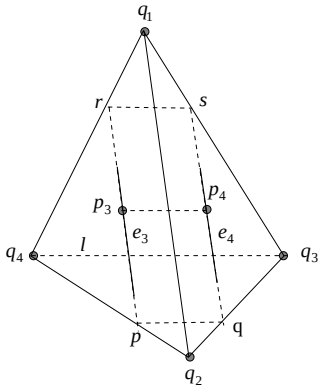


FIG. 7. *Combinatorial type $FFEE$ with coplanar e_3, e_4.*

A.2. Type $FFEE$ when e_3, e_4 are coplanar. Let π be the plane spanned by (coplanar) edges e_3, e_4. (See Figure 7.) Then, $q_4q_3 \parallel p_3p_4$ implies that $q_4q_3 \parallel \pi$, which in turn implies that $q_4q_3 \parallel pq$. Similarly, if r, s denote the intersections of plane through f_2 with lines through e_3 and e_4, then $q_4q_3 \parallel rs$ (Figure 3).

Let $x = pq/q_4q_3$, and $y = rs/q_4q_3$. Then, $\text{dist}(q_2, \ell) = \frac{1}{1-x}\text{dist}(p, \ell)$, and $\text{dist}(q_1, \ell)\frac{1}{1-y}\text{dist}(r, \ell)$, where $\text{dist}(x, \ell)$ denotes the distance from point x to line ℓ. Since the volume of a simplex is 1/3 the base area times height, we have

$$\text{Vol}(T) = \frac{1}{3} * \left[\frac{1}{2}q_4q_3 * \text{dist}(q_2, \ell)\right] * [\text{dist}(q_1, \ell) * \cos\theta]$$
$$= \frac{c}{6} * \cos\theta * \text{dist}(q_1, \ell) * \text{dist}(q_2, \ell)$$
$$= \left[\frac{c}{6} * \cos\theta * \text{dist}(p, \ell) * \text{dist}(r, \ell)\right] * \frac{1}{1-x} * \frac{1}{1-y},$$

where θ is the angle between planes defined by faces f_1 and f_2, and recall that $c = q_4q_3$.

In order for T to be centroidal, p, p_3, r should be collinear, as should the triple q, p_4, s. In the plane of t_3, if we choose $\{\overrightarrow{q_4q_2}, \overrightarrow{q_4q_1}\}$ as the basis, then $p = (1-x, 0)$, $r = (0, 1-y)$, and $p_3 = (1/3, 1/3)$. It is easy to verify that p, p_3, r are collinear if and only if $\frac{1}{1-x} + \frac{1}{1-y} = 3$, and q, p_4, s are collinear if and only if $\frac{1}{1-x} + \frac{1}{1-y} = 3$. Thus, if e_3, e_4 are coplanar and T is centroidal, then necessarily $\frac{1}{1-x} + \frac{1}{1-y} = 3$. However, if this condition is satisfied, then the position of T is not fixed! We need only q_4, q_3 to satisfy that $q_4q_3 = c$, and both of them belong to the line ℓ. Then the intersection of q_4p and q_3q is q_2, the intersection of q_4r and q_3s is q_1, and it guarantees that e_3, e_4 contain the centroids of t_3, t_4, respectively.

In summary, if there exists one centroidal simplex in this instance, then there exists a continuous family of them. We can take the limit of this continuous family in one direction, and the limiting simplex will have fewer degrees of freedom and the same volume as any of these intermediate centroidal simplices. Thus, the volume of a centroidal simplex of type $FFEE$ in this case is no smaller than the best simplex in class $FFFF$ or $FFFE$. Thus, we need not consider this case.

Acknowledgments. We thank Ken Clarkson, Joseph O'Rourke, and Chee Yap for several helpful discussions.

REFERENCES

[1] G. E. Andrews, *A lower bound for the volume of strictly convex bodies with many boundary lattice points,* Trans. Amer. Math. Soc., 106 (1963), pp. 270–279.

[2] G. Barequet and S. Har-Peled, *Efficiently approximating the minimum-volume bounding box of a point set in three dimensions,* in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, ACM, New York, pp. 82–91.

[3] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi, *I-COLLIDE: An interactive and exact collision detection system for large-scale environments,* in Proceedings of the ACM Interactive 3D Graphics Conference, 1995, pp. 189–196.

[4] M. Craig, *Minimum volume transforms for remotely sensed data,* IEEE Trans. Geosci. Remote Sensing, 32 (1994), pp. 542–552.

[5] R. Erlich and W. Full, *Sorting out geology—unmixing mixtures,* in Use and Abuse of Statistical Methods in the Earth Sciences, Oxford University Press, Oxford, UK, 1987, pp. 33–46.

[6] D. R. Fuhrmann, *A Simplex Shrink-Wrap Algorithm,* in Proceedings of SPIE AeroSense, 1999.

[7] B. Gärtner and W. Welzl, *Linear programming—randomization and abstract frameworks,* in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1064, Springer-Verlag, Berlin, 1996, pp. 669–687.

[8] P. Gritzmann and V. Klee, *On the complexity of some basic problems in computational convexity:* I. *Containment problems,* Discrete Math., 136 (1994), pp. 129–174.

[9] M. Held, J. T. Klosowski, and J. S. B. Mitchell, *Collision Detection for Fly-Throughs in Virtual Environments,* video presentation at ACM Symposium on Computation Geometry, 1996.

[10] P. M. Hubbard, *Collision detection for interactive graphics applications,* IEEE Trans. Visualization and Computer Graphics, 1 (1995), pp. 218–230.

[11] V. Klee, *Facet-centroids and volume minimization,* Studia Sci. Math. Hungar., 21 (1986), pp. 143–147.

[12] V. Klee and M. C. Laskowski, *Finding the smallest triangle containing a given convex polygon,* J. Algorithms, 6 (1985), pp. 359–366.

[13] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, *Efficient collision detection using bounding volume hierarchies of k-DOPs,* IEEE Trans. Visualization and Computer Graphics, 4 (1998), pp. 21–36.

[14] J.-C. Latombe, *Robot Motion Planning,* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[15] N. Megiddo, *Linear-time algorithms for linear programming in R^3 and related problems,* SIAM J. Comput., 12 (1983), pp. 759–776.

[16] M. P. Moore and J. Wilhelms, *Collision detection and response for computer animation,* Comput. Graph., 22 (1998), pp. 289–298.
[17] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, *An optimal algorithm for finding minimal enclosing triangle,* J. Algorithms, 7 (1986), pp. 258–269.
[18] J. O'Rourke, *Finding Minimal Enclosing Tetrahedra,* Technical Report JHU 84/05, Johns Hopkins University, Baltimore, MD, 1984.
[19] J. O'Rourke, *Finding minimal enclosing boxes,* Internat. J. Comput. Inform. Sci., 14 (1985), pp. 183–199.
[20] R. Renner, *The resolution of a compositional data into mixtures of fixed source compositions,* Appl. Stat., 42 (1993), pp. 615–631.
[21] S. Suri, P. M. Hubbard, and J. Hughes, *Analyzing bounding boxes for object intersection,* ACM Trans. Graphics, 18 (1999), pp. 257–277.
[22] G. Vegter and C. Yap, *Minimal circumscribing simplices,* in Proceedings of the 3rd Canadian Conference on Computational Geometry, Vancouver, Canada, 1991, pp. 58–61.
[23] Y. Zhou and S. Suri, *Analysis of a bounding box heuristic for object intersection,* J. ACM, 46 (1999), pp. 833–857.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1358–1369

PERCEPTRON, WINNOW, AND PAC LEARNING*

ROCCO A. SERVEDIO†

Abstract. We analyze the performance of the widely studied Perceptron and Winnow algorithms for learning linear threshold functions under Valiant's probably approximately correct (PAC) model of concept learning. We show that under the uniform distribution on boolean examples, the Perceptron algorithm can efficiently PAC learn nested functions (a class of linear threshold functions known to be hard for Perceptron under arbitrary distributions) but cannot efficiently PAC learn arbitrary linear threshold functions. We also prove that Littlestone's Winnow algorithm is not an efficient PAC learning algorithm for the class of positive linear threshold functions, thus answering an open question posed by Schmitt [*Neural Comput.*, 10 (1998), pp. 235–250]. Based on our results we conjecture that no "local" algorithm can learn linear threshold functions efficiently.

Key words. linear threshold function, PAC learning, Perceptron, Winnow

AMS subject classification. 68Q32

PII. S0097539798340928

1. Introduction.

The classical Perceptron algorithm and Littlestone's Winnow algorithm are two algorithms for learning linear threshold functions which have been studied extensively in the online mistake bound model [4, 8, 13, 16, 17, 19, 28]. In this model the learning algorithm sequentially makes predictions on examples as they are received, using a hypothesis which it can update after each example, and the performance of an algorithm is measured by the worst-case number of prediction mistakes it makes on any example sequence. The well-known Perceptron convergence theorem [6, 20, 24] establishes conditions under which the Perceptron algorithm will make a bounded number of mistakes on a (possibly infinite) sequence of examples, and Littlestone [14, 15, 16] has obtained similar results for the Winnow algorithm.

Despite widespread interest in the Perceptron and Winnow algorithms, relatively little is known about their performance in Valiant's commonly used probably approximately correct (PAC) model of concept learning [26]. In this paper we establish some positive and negative results which help to clarify the learning abilities of Perceptron and Winnow in the PAC model.

In the PAC learning model there is a fixed distribution $\mathcal{D}$ from which labelled examples are drawn; the goal of the learner is to find a hypothesis which closely approximates the target function under distribution $\mathcal{D}$. It has long been known that if the space of possible examples is the n-dimensional unit sphere S^n, then the Perceptron algorithm is not an efficient PAC learning algorithm for the class of linear threshold functions because of "hard" distributions which concentrate their weight on examples close to the separating hyperplane. Baum [5] has shown that if $\mathcal{D}$ is restricted to be the uniform distribution on S^n, though, then Perceptron is an efficient PAC learning algorithm for the class of linear threshold functions. Schmitt [25] has shown that Perceptron is not an efficient PAC learning algorithm for the class of linear threshold functions over the example space $\{0,1\}^n$; his proof works by exhibiting a

*Received by the editors June 19, 1998; accepted for publication (in revised form) January 23, 2002; published electronically May 29, 2002. This work was supported by an NSF Graduate Fellowship and by grants NSF-CCR-95-04436 and ONR-N00014-96-1-0550.

http://www.siam.org/journals/sicomp/31-5/34092.html

†Maxwell Dworkin 147, Division of Engineering and Applied Sciences, Harvard University, Cambridge MA 02138 (rocco@deas.harvard.edu).

nested boolean function (a special type of linear threshold function) and a distribution which is concentrated on "hard" examples for that function. No results analogous to these have appeared for the Winnow algorithm; as Schmitt noted, it was not known "whether Littlestone's rules can PAC identify in polynomial time" [25].

This paper makes the following contributions: In section 3 we show that under the uniform distribution on $\{0,1\}^n$ the Perceptron algorithm is an efficient PAC learning algorithm for the class of nested boolean functions. This demonstrates that Schmitt's negative result for nested functions depends on choosing a "hard" distribution. However, in section 4 we give a simple proof that the Perceptron algorithm is not an efficient PAC learning algorithm for the class of linear threshold functions under the uniform distribution on $\{0,1\}^n$. This provides an interesting contrast to Baum's result and answers an open question in Schmitt [25]. Finally, we prove in section 5 that the Winnow algorithm is not an efficient PAC learning algorithm for the class of positive linear threshold functions over $\{0,1\}^n$. To the best of our knowledge this is the first negative result for Winnow in the PAC model. We conclude in section 6 with suggestions for future research.

2. Preliminaries. A concept class C on an example space X is a collection of boolean functions on X. The sets C and X are striated; i.e., $C = \cup_{n\geq 1} C_n$ and $X = \cup_{n\geq 1} X_n$, where each $c \in C_n$ has domain X_n. Throughout this paper X_n will be the boolean cube $\{0,1\}^n$. We will mainly deal with the class H_n of n-dimensional linear threshold functions. A boolean function $f : \{0,1\}^n \to \{-1,1\}$ is a linear threshold function if there is a weight vector $w \in \mathbb{R}^n$ and a threshold $\theta \in \mathbb{R}$ such that $f(x) = 1$ iff $w \cdot x \geq \theta$. Such a pair (w, θ) is said to represent f. See [7, 21] for extensive treatments of linear threshold functions on $\{0,1\}^n$.

2.1. Perceptron. Throughout its execution the Perceptron algorithm maintains a weight vector w and a threshold θ as its current hypothesis. The algorithm proceeds in a series of steps; in each step it receives an example x, uses its hypothesis to make a prediction on x, and adjusts its hypothesis if the prediction is incorrect. Initially, the algorithm starts with weight vector $w = (0, \ldots, 0)$ and threshold $\theta = 0$. Upon receiving an example x, the algorithm predicts according to the threshold function $w \cdot x \geq \theta$. If the prediction is incorrect, the hypothesis is updated according to the following rule:

- On a false positive prediction, set $w \leftarrow w - x$ and set $\theta \leftarrow \theta + 1$.
- On a false negative prediction, set $w \leftarrow w + x$ and set $\theta \leftarrow \theta - 1$.

No change is made if the hypothesis was correct on x. If each example x belongs to $\{0,1\}^n$, then each w_i and θ will always be integers; this fact will prove useful later.

The well-known Perceptron convergence theorem bounds the number of mistakes which the Perceptron algorithm can make.

THEOREM 2.1 (see [6, 20, 24]). *Let $\langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples with $\|x^i\| \leq R$ and $y_i \in \{-1,1\}$ for all i. Let u be a vector and θ, ξ be such that $y_i(u \cdot x^i - \theta) \geq \xi$ for all i, where $\xi > 0$. Then the total number of mistakes made by the Perceptron algorithm on this example sequence is at most*

$$\frac{(R^2+1)(\|u\|^2+\theta^2)}{\xi^2}.$$

2.2. Winnow. The Winnow algorithm takes as input an initial vector $w^I \in \mathbb{R}^n$, a promotion factor $\alpha \in \mathbb{R}$, and a threshold $\theta \in \mathbb{R}$. The algorithm requires that w^I is positive (i.e., each coordinate of w^I is positive), that $\alpha > 1$, and that $\theta > 0$. Like

the Perceptron algorithm, Winnow proceeds in a series of trials and predicts in each trial according to the threshold function $w \cdot x \geq \theta$. If the prediction is correct, then no update is performed; otherwise the weights are updated as follows:

- On a false positive prediction for all i set $w_i \leftarrow \alpha^{-x_i} w_i$;
- On a false negative prediction for all i set $w_i \leftarrow \alpha^{x_i} w_i$.

It should be noted that, in this form, Winnow is capable only of expressing positive threshold functions as its hypotheses. This limitation can be easily overcome, however, by using various simple transformations on the input (see [14, 15]). Littlestone has proved the following result, analogous to the Perceptron convergence theorem, bounding the number of mistakes which Winnow makes.

THEOREM 2.2 (see [16]). *Let $\langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ be a sequence of labeled examples with $x^i \in [0,1]^n$ and $y_i \in \{-1, 1\}$ for all i. Let u be a positive vector and $\delta > 0$ be such that whenever $y_i = 1$ we have $u \cdot x \geq 1$ and whenever $y_i = -1$ we have $u \cdot x \leq 1 - \delta$. If Winnow is run on this example sequence with initial parameters w^I, α, θ where $w^I = (1, \ldots, 1)$, $\alpha = 1 + \frac{\delta}{2}$ and $\theta > 0$, then the total number of mistakes made by Winnow on this example sequence is at most*

$$\frac{8n}{\delta^2 \theta} + \max\left\{0, \frac{14}{\delta^2} \sum_{i=1}^{n} u_i \ln(u_i \theta)\right\}.$$

2.3. PAC learning with online learning algorithms. In Valiant's PAC learning model [12, 26] the learning algorithm has access to an *example oracle* $EX(c, \mathcal{D})$ which, in one time step, provides a labelled example $\langle x, c(x) \rangle$, where x is drawn from the distribution $\mathcal{D}$ on the example space X. The function $c \in C$ is called the *target concept*; the goal of the learning algorithm is to construct a hypothesis h which, with high probability, has low error with respect to c. We thus have the following.

DEFINITION 2.3. *We say that an online learning algorithm A (such as Perceptron or Winnow) is an efficient PAC learning algorithm for concept class C over X if there is a polynomial $p(\cdot, \cdot, \cdot)$ such that the following conditions hold for any $n \geq 1$, any distribution $\mathcal{D}$ over X_n, any $c \in C_n$, and any $0 < \epsilon, \delta < 1$:*

- *Given any example $x \in X_n$ algorithm A always evaluates its hypothesis on x and (once provided with $c(x)$) updates its hypothesis in poly(n) time.*
- *If algorithm A is run on a sequence of examples generated by successive calls to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A will generate a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] < \epsilon$ after at most $p(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ calls to $EX(c, \mathcal{D})$.*

An algorithm A is said to be an efficient PAC learning algorithm under a fixed distribution $\mathcal{D}$ if it satisfies the above definition for the fixed distribution $\mathcal{D}$. The most natural distribution over a finite set is of course the uniform distribution; we write $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$.

2.4. PAC learning versus online mistake bound learning. Several generic techniques are known [1, 11, 14] for converting any online mistake bound learning algorithm to a PAC algorithm. These conversion procedures yield PAC learning algorithms whose running time is polynomially related to the running time of the original online algorithm. Using these conversion procedures Theorems 2.1 and 2.2 imply that Perceptron and Winnow are efficient PAC learning algorithms for certain restricted linear threshold learning problems. For example, one straightforwardly obtains the following.

COROLLARY 2.4. *Let C_W be the class of linear threshold functions $w \cdot x \geq \theta$ over $\{0,1\}^n$ such that each w_i is an integer and $\sum_{i=1}^n |w_i| < W$. Then both Perceptron and Winnow yield PAC learning algorithms for C_W which run in time poly$(n, W, \frac{1}{\epsilon}, \frac{1}{\delta})$.*

It is well known, though, that there are linear threshold functions over $\{0,1\}^n$ which require integer coefficients of magnitude $2^{\Omega(n)}$. (See, e.g., [10]; we will see an example of such a linear threshold function in section 5.1.) For functions such as these the time bound of Corollary 2.4 is exponentially large in n and hence does not shed light on whether or not Perceptron and Winnow are efficient PAC learning algorithms.

3. Perceptron can learn nested functions under $\mathcal{U}_n$. In this section we establish a sufficient condition for a family of threshold functions to be efficiently learnable by Perceptron under $\mathcal{U}_n$. We prove that nested functions satisfy this condition and thereby immediately obtain the main result of this section. This complements Schmitt's result [25] that the Perceptron algorithm cannot efficiently PAC learn nested functions under arbitrary distributions.

The class of nested functions, denoted NF_n, was introduced by Anthony, Brightwell, and Shawe-Taylor in [3].

DEFINITION 3.1. *The class of nested functions over $x_1, \ldots, x_n$ is defined as follows:*

1. *For $n = 1$, the functions x_1 and $\overline{x}_1$ are nested.*
2. *For $n > 1$, $f(x_1, \ldots, x_n)$ is nested if $f = g * l_n$, where g is a nested function on $x_1, \ldots, x_{n-1}$, $*$ is either $\vee$ or $\wedge$, and l_n is either x_n or $\overline{x}_n$.*

It is easy to verify that the class of nested functions is equivalent to the class of 1-decision lists of length n in which the variables appear in the reverse order $x_n, \ldots, x_1$. The following lemma establishes a canonical representation of nested functions as threshold functions.

LEMMA 3.2. *Any $f \in NF_n$ can be represented by a linear threshold function*

$$w_1 x_1 + \cdots + w_n x_n \geq \theta_n$$

with $\theta_n = k + \frac{1}{2}$ for some integer k, $w_i = \pm 2^{i-1}$, and $\sum_{w_i<0} w_i < \theta_n < \sum_{w_i>0} w_i$.

Proof. The proof is by induction on n. For $n = 1$ the appropriate threshold function is $x_1 \geq \frac{1}{2}$ or $-x_1 \geq -\frac{1}{2}$. For $n > 1$, f must be of the form $g * l_n$, where g is a nested function on $x_1, \ldots, x_{n-1}$. By the induction hypothesis, g can be expressed as a threshold function $w_1 x_1 + \cdots + w_{n-1} x_{n-1} \geq \theta_{n-1}$, with $w_1, \ldots, w_{n-1}, \theta_{n-1}$ satisfying the conditions of the lemma. There are four possibilities:

1. $f = g \wedge x_n$: then f is $w_1 x_1 + \cdots + w_{n-1} x_{n-1} + 2^{n-1} x_n \geq \theta_n = \theta_{n-1} + 2^{n-1}$.
2. $f = g \wedge \overline{x}_n$: then f is $w_1 x_1 + \cdots + w_{n-1} x_{n-1} - 2^{n-1} x_n \geq \theta_n = \theta_{n-1}$.
3. $f = g \vee x_n$: then f is $w_1 x_1 + \cdots + w_{n-1} x_{n-1} + 2^{n-1} x_n \geq \theta_n = \theta_{n-1}$.
4. $f = g \vee \overline{x}_n$: then f is $w_1 x_1 + \cdots + w_{n-1} x_{n-1} - 2^{n-1} x_n \geq \theta_n = \theta_{n-1} - 2^{n-1}$.

In each case it can be verified that the stated threshold function is equivalent to f and that $w_1, \ldots, w_n, \theta_n$ satisfy the conditions of the lemma. ∎

DEFINITION 3.3. *Let G_n be a collection of hyperplanes in $\mathbb{R}^n$. A family $G = \cup_{n \geq 1} G_n$ of hyperplanes is said to be* gradual *if there is some constant $c > 0$ such that the following condition holds: for every $\tau \geq 0$, every $n \geq 1$, and every hyperplane in G_n, at most $c\tau 2^n$ boolean examples $x \in \{0,1\}^n$ lie within Euclidean distance τ of the hyperplane. A class of linear threshold functions F is said to be gradual if there is a mapping $\varphi : F \to G$, where G is a gradual family of hyperplanes, such that for all $f \in F$, if $\varphi(f)$ is the hyperplane $w \cdot x = \theta$, then (w, θ) represents the linear threshold function f.*

LEMMA 3.4. *The class of nested functions is gradual.*

Proof. We use the representation established in Lemma 3.2; so let $f \in NF_n$ and let $w \cdot x \geq \theta$ be a linear threshold function which represents f with $w_i = \pm 2^{i-1}$ and $\theta = k + \frac{1}{2}$ for some integer k. For $x \in \{0,1\}^n$, if $w \cdot x = t$, then t must be an integer, but since every integer has a unique binary representation, at most one $x \in \{0,1\}^n$ can satisfy $w \cdot x = t$ for any given value of t. Consequently, no example $x \in \{0,1\}^n$ can have $|w \cdot x - \theta| < \frac{1}{2}$, and

$$|\{x \in \{0,1\}^n : |w \cdot x - \theta| \leq m\}| \leq 2m + 1$$

for $m \geq \frac{1}{2}$. The lemma follows by noting that $\|w\| = (\frac{4^n - 1}{3})^{1/2} = \Theta(2^n)$ and that the distance from a point x' to the hyperplane $w \cdot x = \theta$ is $\|w\|^{-1} \cdot |w \cdot x' - \theta|$. □

This lemma ensures that relatively few points can lie close to the separating hyperplane for a nested function; consequently, as we run Perceptron most of the updates will cause the algorithm to make significant progress, and it will achieve ϵ-accuracy in polynomial time. The following theorem formalizes this intuition.

THEOREM 3.5. *If C is a gradual class of linear threshold functions, then Perceptron is an efficient PAC learning algorithm for C under $\mathcal{U}_n$.*

Proof. The proof is similar to the proof of Theorem 1 in [5]. Let $w \cdot x \geq \theta$ be a gradual linear threshold function which represents c. We assume without loss of generality that w, θ have been normalized; i.e., $\|w\| = 1$, so $|w \cdot x - \theta|$ is the distance from x to the hyperplane. By Definition 3.3, there is some constant $k > 0$ such that for all $\tau > 0$ the probability that a random example drawn from $EX(c, \mathcal{U}_n)$ is within distance τ of the hyperplane $w \cdot x = \theta$ is at most $\tau/2k$. Letting $\tau = k\epsilon$, we have that with probability at most $\epsilon/2$, a random example drawn from $EX(c, \mathcal{U}_n)$ is within distance $k\epsilon$ of the hyperplane. Let $B \subseteq \{0,1\}^n$ be the set of examples x which lie within distance $k\epsilon$ of the hyperplane; so $\Pr_{x \in \mathcal{U}_n}[x \in B] \leq \epsilon/2$.

Let (w_t, θ_t) represent the Perceptron algorithm's hypothesis after t updates have been made. If (w_t, θ_t) is not yet ϵ-accurate, then with probability at most $1/2$ the next example which causes an update will be in B. Consider the following potential function:

$$N_t(\alpha) = \|\alpha w - w_t\|^2 + (\alpha\theta - \theta_t)^2.$$

Recalling our Perceptron update rules, we see that $N_{t+1}(\alpha) - N_t(\alpha)$ is

$$\begin{aligned}
\Delta N(\alpha) &= \|\alpha w - w_{t+1}\|^2 + (\alpha\theta - \theta_{t+1})^2 - \|\alpha w - w_t\|^2 - (\alpha\theta - \theta_t)^2 \\
&= \mp 2\alpha w \cdot x \pm 2\alpha\theta \pm 2w_t \cdot x \mp 2\theta_t + \|x\|^2 + 1 \\
&\leq 2\alpha A \pm 2(w_t \cdot x - \theta_t) + n + 1
\end{aligned}$$

with $A = \mp(w \cdot x - \theta)$. Since x was misclassified, we know that $\pm(w_t \cdot x - \theta_t) \leq 0$, and hence $\Delta N(\alpha) \leq 2\alpha A + n + 1$. If $x \in B$, then $A \leq 0$; if $x \notin B$, then $A \leq -k\epsilon\alpha$. As a result, $\Delta N(\alpha) < n + 1$ for $x \in B$, and $\Delta N(\alpha) < n + 1 - 2k\epsilon\alpha$ for $x \notin B$.

Suppose that during the course of its execution the Perceptron algorithm has made r updates on examples in B and s updates on examples outside B. Since (w, θ) have been normalized we have that $|\theta| \leq \sqrt{n}$, and hence $N_0(\alpha) \leq \alpha^2(n+1)$. Since $N_t(\alpha)$ must always be nonnegative, it follows that

$$0 \leq r(n+1) + s(n + 1 - 2k\epsilon\alpha) + \alpha^2(n+1).$$

If we set $\alpha = \frac{12(n+1)}{5k\epsilon}$, then simplifying the above inequality we obtain

$$0 \leq r - \frac{19}{5}s + \frac{144(n+1)^2}{25(k\epsilon)^2},$$

from which it follows that if $m_1 = \frac{144(n+1)^2}{25(k\epsilon)^2}$ updates have been made, at least 7/12 of the updates must have been on examples in B.

Let $m = \max\{144 \ln \frac{\delta}{2}, m_1\}$ and consider running the Perceptron algorithm for $2m/\epsilon$ examples. Let $h_1, h_2, \ldots$ denote the hypotheses which are generated by the Perceptron algorithm during the course of its execution on these $2m/\epsilon$ examples. We have that

$$\begin{aligned}
\Pr[\text{each } h_i \text{ has error } > \epsilon] &= \Pr[(\text{each } h_i \text{ has error } > \epsilon)\ \& \\
&\qquad (\text{fewer than } m \text{ updates take place})] \\
&\quad + \Pr[(\text{each } h_i \text{ has error } > \epsilon)\ \& \\
&\qquad (\text{at least } m \text{ updates take place})] \\
&\le \Pr[(\text{fewer than } m \text{ updates take place}) \mid \\
&\qquad (\text{each } h_i \text{ has error} > \epsilon)] \\
&\quad + \Pr[(\text{at least } m \text{ updates take place}) \mid \\
&\qquad (\text{each } h_i \text{ has error} > \epsilon)].
\end{aligned}$$

To bound the first of these conditional probabilities, we note that conditioned on the event that each h_i has error at least ϵ, the expected number of updates is at least $2m$. A straightforward Chernoff bound [2] shows that this first conditional probability is at most $\delta/2$.

To bound the second conditional probability, we recall that if at least m updates take place, then at least 7/12 of the updates must be on examples in B. However, as noted earlier, if each h_i has error at least ϵ, then for each update the probability that the update is in B is at most 1/2. Another application of Chernoff bounds shows that the second conditional probability is at most $\delta/2$, and the theorem is proved. □

As an immediate corollary of Theorem 2.1 we have that the Perceptron is a PAC learning algorithm for the class of nested functions under the uniform distribution on $\{0,1\}^n$.

4. Perceptron cannot learn H_n under U_n. A very simple argument suffices to establish this result. We require the following definition.

DEFINITION 4.1 (see [25]). *The* weight complexity *of a linear threshold function f is the smallest positive integer t such that f can be represented as $w \cdot x \ge \theta$, with each w_i and θ an integer and $\max\{|w_1|, \ldots, |w_n|, |\theta|\} \le t$.*

THEOREM 4.2. *The Perceptron algorithm is not an efficient PAC learning algorithm for the class of linear threshold functions under the uniform distribution on $\{0,1\}^n$.*

Proof. We take $\epsilon = \frac{1}{2^n+1}$; so any ϵ-accurate hypothesis must agree exactly with the target concept, since misclassification of a single example would imply an error rate under the uniform distribution of at least $\frac{1}{2^n} > \epsilon$. Håstad [9] has constructed a linear threshold function which has weight complexity $2^{\Omega(n \log n)}$. If we take this as our target concept, then it follows that at least $2^{\Omega(n \log n)}$ update steps must be performed by the Perceptron algorithm in order to achieve exact identification (since Perceptron hypothesis weights are always integral and each weight is increased by at most 1 during each Perceptron update step). However, the amount of computation time which a PAC learning algorithm is allowed is only $\text{poly}(n, \frac{1}{\epsilon}) = \text{poly}(n, 2^n) = 2^{O(n)}$. □

5. Winnow cannot learn H_n. Although the Winnow algorithm has been extensively studied in the online mistake bound learning model, little is known about its performance in other learning models. In this section we show that Winnow is not an efficient PAC learning algorithm for the class of positive linear threshold functions. More precisely, we prove the following theorem.

THEOREM 5.1. *Given any positive start vector w^I, any promotion factor $\alpha > 1$ and any threshold $\theta > 0$, there is a positive threshold function c, a distribution $\mathcal{D}$ on $\{0,1\}^n$, and a value $\epsilon > 0$ for which Winnow(w^I, α, θ) will not generate a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$ in poly$(n, \frac{1}{\epsilon})$ steps.*

As a consequence of the proof of this theorem, we will obtain an explicit family of "hard" threshold functions and corresponding example sequences which cause Winnow to make exponentially many mistakes. Maass and Turan [18] have used a counting argument to show that, given any (w^I, α, θ), there exists a target threshold function and example sequence which together cause Winnow(w^I, α, θ) to make exponentially many mistakes, but no explicit construction was known.

5.1. A threshold function with large coefficients and a small specifying set. The proof of Theorem 3.5 makes use of several lemmas. In the first lemma we show that a nested boolean function with alternating connectives requires exponentially large coefficients. (Similar results can be found in [21, 22].)

LEMMA 5.2. *Let n be odd and let $u \cdot x \geq \theta$ be a positive threshold function which represents the nested function*

$$f_n = (\dots(x_1 \vee x_2) \wedge x_3) \vee x_4) \dots) \vee x_{n-1}) \wedge x_n.$$

For all $i \geq 3$ we have that $u_i \geq F_{i-3} u_3$, where F_i is the ith Fibonacci number: $F_0 = 1, F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, \dots$.

Proof. The proof is by induction on k, where $n = 2k + 1$. For clarity we use two base cases. The case $k = 1$ is trivial. If $k = 2$, then since $f_5(0,0,0,1,1) = 1$ and $f_5(0,0,1,0,1) = 0$, we find that $u_4 \geq u_3$. Similarly, since $f_5(0,0,1,1,0) = 0$, we find that $u_5 \geq u_3$.

We now suppose that the lemma is true for $k = 1, 2, \dots, j-1$ and let $n = 2j+1$. By assumption, $(u_1, \dots, u_n)$ and θ are such that $u \cdot x \geq \theta$ represents f_n. If we fix $x_n = 1$ and $x_{n-1} = 0$, then it follows that $u_1 x_1 + \cdots + u_{n-2} x_{n-2} \geq \theta - u_n$ is a threshold function which represents f_{n-2}; so by the induction hypothesis the lemma holds for $u_3, \dots, u_{n-2}$, and we need only show that it holds for u_{n-1} and u_n.

Since $f_n(1,1,\dots,1,0,0,1) = 0$, we have that $u_1 + u_2 + \cdots + u_{n-3} + u_n < \theta$. On the other hand, since $f_n(0,0,\dots,0,1,1) = 1$, we have that $u_{n-1} + u_n \geq \theta$. From these two inequalities it follows that

$$u_{n-1} \geq u_1 + u_2 + \cdots + u_{n-3}.$$

Since u is positive, this inequality implies that

$$u_{n-1} > u_3 + u_4 + \cdots + u_{n-3}.$$

Using the induction hypothesis we obtain the inequality

$$u_{n-1} \geq (1 + F_1 + \cdots + F_{n-6}) u_3 = F_{n-4} u_3.$$

Similarly, since $f_n(1,1,\dots,1,0) = 0$, we have that $u_1 + u_2 + \cdots + u_{n-1} < \theta$; so

$$u_n \geq u_1 + u_2 + \cdots + u_{n-2} > u_3 + u_4 + \cdots + u_{n-2}.$$

By the induction hypothesis, we find that

$$u_n \geq (1 + F_1 + \cdots + F_{n-5})u_3 = F_{n-3}u_3,$$

and the lemma is proved. □

Since $F_n = \Omega(\phi^n)$, where $\phi = \frac{1+\sqrt{5}}{2}$, we have shown that f_n must have coefficients whose ratio is exponentially large.

We require a definition from [3] before stating the next lemma.

DEFINITION 5.3. *Let $c \in H_n$. We say that c is* consistent *with a set $S = \{\langle x^1, b_1\rangle, \langle x^2, b_2\rangle, \ldots, \langle x^m, b_m\rangle\}$ of labelled examples if $c(x^i) = b_i$ for all i. The set S is said to* specify *c in H_n if c is the only function in H_n which is consistent with S; we say that such a set is a* specifying set *for c in H_n. The* specification number *of c, denoted $\sigma_n(c)$, is the smallest size of any specifying set for c in H_n.*

The following results are proved in [3].

LEMMA 5.4.

1. *Every $c \in H_n$ has a unique specifying set of size $\sigma_n(c)$.*
2. *If $c \in NF_n$, then $\sigma_n(c) = n + 1$.*
3. *Given any $c \in H_n$, let $c\uparrow(x_1, \ldots, x_{n-1}) = c(x_1, \ldots, x_{n-1}, 1)$ and let $c\downarrow(x_1, \ldots, x_{n-1}) = c(x_1, \ldots, x_{n-1}, 0)$. Then $\sigma_n(c) \leq \sigma_{n-1}(c\uparrow) + \sigma_{n-1}(c\downarrow)$.*
4. *If $c \in H_n$ and c depends on only coordinates $1, 2, \ldots, k$, then the specification number of c in H_n is $\sigma_n(c) = 2^{n-k}\sigma_k(c)$.*

We use the results of Lemma 5.4 to show that the function g_n defined below has a small specifying set.

LEMMA 5.5. *Let n be even and let g_n be the linear threshold function represented by*

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} + (2^{n-2} + 1)x_n \geq 4 + 16 + \cdots + 2^{n-4} + 2^{n-2} + \frac{1}{2}.$$

Then $\sigma_n(g_n) \leq 5n - 8$.

Proof. The function $g_n\downarrow$ is represented by

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} \geq 4 + 16 + \cdots + 2^{n-2} + \frac{1}{2}.$$

It is straightforward to verify that this is precisely the nested function

$$(\ldots(x_1 \vee x_2) \wedge x_3) \vee x_4)\ldots) \wedge x_{n-1}$$

on $n - 1$ variables. By part 2 of Lemma 5.4, we have $\sigma_{n-1}(g_n\downarrow) = n$.

The function $g_n\uparrow$ is represented by

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} \geq 4 + 16 + \cdots + 2^{n-4} - \frac{1}{2}.$$

Again, one can easily verify that this is precisely the nested function

$$(\ldots(x_3 \vee x_4) \wedge x_5) \vee x_6)\ldots) \wedge x_{n-3}) \vee x_{n-2}) \vee x_{n-1}$$

on the $n - 3$ variables $x_3, \ldots, x_{n-1}$; in this nested function the boolean connectives alternate between $\vee$ and $\wedge$ until the very end, where two consecutive $\vee$'s occur. Since $g_n\uparrow$ does not depend on the two variables x_1, x_2, parts 4 and 2 of Lemma 5.4 imply that $\sigma_{n-1}(g_n\uparrow) = 4\sigma_{n-3}(g_n\uparrow) = 4(n - 2)$. It follows from part 3 of Lemma 5.4 that $\sigma_n(g_n) \leq 5n - 8$. □

5.2. Proof of Theorem 3.5. One more lemma is required. We prove that the coefficients of x_{n-1} and x_n in any representation of g_n must be almost, but not exactly, equal.

LEMMA 5.6. *Let n be even and let g_n be as defined in Lemma* 5.5. *Let $v \cdot x \geq \theta$ represent g_n. Then $v_{n-1} < v_n < v_{n-1} + v_3$.*

Proof. Let e_j denote the boolean vector whose jth coordinate is 1 and all of whose other coordinates are 0. Let $a = e_3 + e_5 + e_7 + \cdots + e_{n-1}$. Since $g_n(a) = 0$, it follows that $v_3 + v_5 + \cdots + v_{n-1} < \theta$. On the other hand, let $b = e_3 + e_5 + \cdots + e_{n-3} + e_n$. Since $g_n(b) = 1$, it follows that $v_3 + v_5 + \cdots + v_{n-3} + v_n \geq \theta$. Combining these two inequalities, we find that $v_n > v_{n-1}$.

To see that v_n cannot be much greater than v_{n-1}, let $c = e_1 + e_5 + e_7 + e_9 + \cdots + e_{n-3} + e_n$. Since $g_n(c) = 0$, we have $v_1 + v_5 + \cdots + v_{n-3} + v_n < \theta$. On the other hand, let $d = e_1 + e_3 + \cdots + e_{n-1}$. Since $g_n(d) = 1$, we have that $v_1 + v_3 + \cdots + v_{n-1} \geq \theta$. Combining these two inequalities, we find that $v_n < v_{n-1} + v_3$, and the lemma is proved. □

Proof of Theorem 3.5. We first prove the theorem for the restricted case in which we assume that $w^I = (1, \ldots, 1)$. After we have done this we will show how this assumption can be eliminated.

Fix $\alpha > 1$ and $\theta > 0$. Let S denote the specifying set for the threshold function g_n; we know from Lemma 5.5 that $|S| \leq 5n - 8$. Let $\mathcal{D}$ be the distribution on $\{0,1\}^n$ which is uniform on S and gives zero weight to vectors outside of S. We will show that with g_n as the target concept, $\mathcal{D}$ as the distribution over examples, and $\epsilon = \frac{1}{5n-7}$ as the error parameter, Winnow$((1, \ldots, 1), \alpha, \theta)$ cannot achieve a hypothesis $h(x)$ which satisfies $\Pr_{\mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$ in $\text{poly}(n, \frac{1}{\epsilon}) = \text{poly}(n)$ steps. To see this, first note that by our choice of ϵ and $\mathcal{D}$, any threshold function $w \cdot x \geq \theta$ which is ϵ-accurate with respect to g_n must be consistent with S. (This technique was first used by Pitt and Valiant in [23].) Since S is a specifying set for g_n, though, if $w \cdot x \geq \theta$ is consistent with S, then it must in fact agree exactly with g_n. We will show that there is no value of α which could enable Winnow to generate a vector w such that $w \cdot x \geq \theta$ represents $g_n(x)$ in $\text{poly}(n)$ steps.

Let (w, θ) be such that Winnow generates w and $w \cdot x \geq \theta$ represents $g_n(x)$. Since $g_n\downarrow$ is precisely the nested function f_{n-1} of Lemma 5.2 and w is positive, by Lemma 5.2 we have that $w_{n-1} \geq F_{n-4} w_3$. Combining this with Lemma 5.6, we obtain

$$1 < \frac{w_n}{w_{n-1}} < 1 + \frac{1}{F_{n-4}} = 1 + \frac{1}{\Omega(\phi^n)}.$$

Since we assumed that $w^I = (1, \ldots, 1)$, and every example for Winnow lies in $\{0,1\}^n$, it follows that $\frac{w_n}{w_{n-1}} = \alpha^j$ for some positive integer j, and hence that $\alpha = 1 + \frac{1}{\Omega(\phi^n)}$. However, then at least $\Omega(n\phi^n)$ update steps are required to achieve $w_{n-1} \geq F_{n-4} w_3$; consequently, no hypothesis consistent with g_n can be achieved in $\text{poly}(n)$ steps.

Now we consider the case of an arbitrary positive start vector w^I; so fix some positive w^I, $\alpha > 1$, and $\theta > 0$. We assume without loss of generality that $w_1^I \leq w_2^I \leq \cdots \leq w_n^I$, since if this is not already the case renaming variables will make it so. Since all examples for Winnow are in $\{0,1\}^n$, at every point in the execution of Winnow the ratio of weights w_i and w_j must be $\frac{w_i^I}{w_j^I} \cdot \alpha^c$ for some integer c. If there is no integer i_1 such that

$$1 < \frac{w_n^I}{w_{n-1}^I} \cdot \alpha^{i_1} < 1 + \frac{1}{F_{n-4}},$$

then Winnow can never achieve a hypothesis which represents the threshold function $g_n(x)$, and can hence never achieve ϵ-accuracy; so we assume that such an i_1 exists. Similarly, if there is no integer i_2 such that

$$1 < \frac{w^I_{n-1}}{w^I_n} \cdot \alpha^{i_2} < 1 + \frac{1}{F_{n-4}},$$

then there is a threshold function which Winnow can never achieve (the function g_n with a permutation on the variables is such a function); so such an i_2 must exist as well. Taking the product of these inequalities, we find that

$$1 < \alpha^{i_1+i_2} < \left(1 + \frac{1}{F_{n-4}}\right)^2.$$

Since $i_1 + i_2$ must be a positive integer, this implies that $\alpha < (1 + \frac{1}{F_{n-4}})^2$. Now consider a threshold function which requires that $w_1 \geq F_{n-4}w_n$. (Again, g_n with a permutation on the variables is such a function.) Since $w^I_1 \leq w^I_n$ and $\alpha < (1+\frac{1}{F_{n-4}})^2$, it follows that $\Omega(n\phi^n)$ update steps will be required. Hence no hypothesis consistent with g_n can be achieved in polynomial time, and the theorem is proved. □

As an immediate consequence of this proof, we note that the example sequence which simply cycles through S will force Winnow to make exponentially many mistakes on g_n.

6. Conclusion. Many questions remain to be answered about the PAC learning ability of simple online algorithms such as Perceptron and Winnow. Perceptron is now known to be a PAC learning algorithm for linear threshold functions under the uniform distribution on the n-dimensional unit sphere and is known not to be a PAC learning algorithm for linear threshold functions under the uniform distribution on $\{0,1\}^n$. Analogous results have yet to be obtained for Winnow under uniform distributions. More generally, it would be interesting to identify the class of threshold functions which Perceptron (Winnow) can PAC learn under the uniform distribution and under arbitrary distributions.

The linear threshold function g_n used in our Winnow proof is very similar to a nested function; in particular, both $g_n\uparrow$ and $g_n\downarrow$ are nested functions. In light of this fact, and of Schmitt's proof that Perceptron is not a PAC learning algorithm for the class of nested functions, it would be interesting to determine whether Winnow is capable of PAC learning the class of nested functions (or, equivalently, the class of decision lists).

Both the Perceptron and Winnow algorithms are *local* in the sense of [27]: each update to a weight w_i depends only on the value of $w \cdot x$, the value of w_i, the value of x_i, and the correct classification of the example. Such algorithms are of particular interest because they require very limited communication between the processors that perform the updates for each weight and are hence well suited for implementation on a distributed system such as a neural network. Known algorithms for learning threshold functions efficiently, such as the algorithm of [18] which is based on linear programming, are nonlocal. We conjecture that local learning algorithms cannot efficiently learn the unrestricted class of threshold functions.

A final issue is attribute efficiency. A learning algorithm is said to be *attribute efficient* if, when the target concept has k relevant variables out of n, the number of calls which the algorithm makes to the example oracle is polynomial in k and poly-

logarithmic in n. Littlestone's results for Winnow show that it is an attribute efficient algorithm for certain simple subclasses of threshold functions such as disjunctions and r-out-of-k threshold functions. Our results imply that Winnow is not an attribute efficient PAC learning algorithm for the unrestricted class of linear threshold functions. It would be interesting to gain a better understanding of the abilities and limitations of Winnow as an attribute efficient learning algorithm.

Acknowledgment. We thank Les Valiant for helpful comments and suggestions.

REFERENCES

[1] D. Angluin, *Queries and concept learning,* Machine Learning, 2 (1988), pp. 319–342.
[2] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings,* J. Comput. System Sci., 18 (1979), pp. 155–193.
[3] M. Anthony, G. Brightwell, and J. Shawe-Taylor, *On specifying boolean functions using labelled examples,* Discrete Appl. Math., 61 (1993), pp. 1–25.
[4] P. Auer and M. Warmuth, *Tracking the best disjunction,* in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 312–321.
[5] E. B. Baum, *The perceptron algorithm is fast for nonmalicious distributions,* Neural Comput., 2 (1990), pp. 248–260.
[6] H. Block, *The perceptron: A model for brain functioning,* Rev. Modern Phys., 34 (1962), pp. 123–135.
[7] M. L. Dertouzos, *Threshold Logic: A Synthesis Approach,* MIT Press, Cambridge, MA, 1965.
[8] Y. Freund and R. Schapire, *Large margin classification using the Perceptron algorithm,* in Proceedings of the 11th Annual Conference on Computational Learning Theory, ACM, New York, 1998, pp. 209–217.
[9] J. Håstad, *On the size of weights for threshold gates,* SIAM J. Discrete Math., 7 (1994), pp. 484–492.
[10] S. Hampson and D. Volper, *Linear function neurons: Structure and training,* Biol. Cybernet., 53 (1986), pp. 203–217.
[11] D. Haussler, *Space Efficient Learning Algorithms,* Technical Report UCSC-CRL-88-2, University of California, Santa Cruz, CA, 1988.
[12] M. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory,* MIT Press, Cambridge, MA, 1994.
[13] J. Kivinen, M. Warmuth, and P. Auer, *The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant,* in Proceedings of the 8th Annual Conference on Computational Learning Theory, ACM, New York, 1995, pp. 289–296.
[14] N. Littlestone, *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm,* Machine Learning, 2 (1988), pp. 285–318.
[15] N. Littlestone, *Mistake Bounds and Logarithmic Linear-Threshold Learning Algorithms,* Ph.D. thesis, Technical Report UCSC-CRL-89-11, University of California, Santa Cruz, CA, 1989.
[16] N. Littlestone, *Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow,* in Proceedings of the 4th Annual Conference on Computational Learning Theory, ACM, New York, 1991, pp. 147–156.
[17] W. Maass and M. Warmuth, *Efficient Learning with Virtual Threshold Gates,* Technical Report UCSC-CRL-95-37, University of California, Santa Cruz, CA, 1995.
[18] W. Maass and G. Turan, *How fast can a threshold gate learn?*, in Computational Learning Theory and Natural Learning Systems. Volume I: Constraints and Prospects, S. J. Hanson, G. Drastal, and R. Rivest, eds., MIT Press, Cambridge, MA, 1994, pp. 381–414.
[19] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry,* expanded ed., MIT Press, Cambridge, MA, 1988.
[20] A. Novikoff, *On convergence proofs on perceptrons,* in Proceedings of the Symposium on Mathematical Theory of Automata, Vol. 12, 1962, pp. 615–622.
[21] S. Muroga, *Threshold Logic and its Applications,* Wiley-Interscience, New York, 1971.
[22] I. Parberry, *Circuit Complexity and Neural Networks,* MIT Press, Cambridge, MA, 1994.
[23] L. Pitt and L. G. Valiant, *Computational limitations on learning from examples,* J. ACM, 35 (1988), pp. 965–984.

[24] F. Rosenblatt, *Principles of Neurodynamics,* Springer-Verlag, New York, 1962.
[25] M. Schmitt, *Identification criteria and lower bounds for Perceptron-like learning rules,* Neural Comput., 10 (1998), pp. 235–250.
[26] L. G. Valiant, *A theory of the learnable,* Comm. ACM, 27 (1984), pp. 1134–1142.
[27] L. G. Valiant, *Circuits of the Mind,* Oxford University Press, New York, 1994.
[28] L. G. Valiant, *Projection learning,* in Proceedings of the 11th Annual Conference on Computational Learning Theory, ACM, New York, 1998, pp. 287–293.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1370–1382

MINIMIZING THE FLOW TIME WITHOUT MIGRATION*

BARUCH AWERBUCH†, YOSSI AZAR‡, STEFANO LEONARDI§, AND ODED REGEV‡

Abstract. We consider the classical problem of scheduling jobs in a multiprocessor setting in order to minimize the flow time (total time in the system). The performance of the algorithm, both in offline and online settings, can be significantly improved if we allow preemption, i.e., interrupt a job and later continue its execution, perhaps migrating it to a different machine. Preemption is inherent to make a scheduling algorithm efficient. While in the case of a single processor most operating systems can easily handle preemptions, migrating a job to a different machine results in a huge overhead. Thus, it is not commonly used in most multiprocessor operating systems. The natural question is whether migration is an inherent component for an efficient scheduling algorithm in either the online or offline setting.

Leonardi and Raz [*Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 110–119] showed that the well-known algorithm, *shortest remaining processing time* (SRPT), performs within a logarithmic factor of the optimal offline algorithm. Note that SRPT must use *both* preemption and migration to schedule the jobs. It is not known if better approximation factors can be reached and thus SRPT, although it is an online algorithm, becomes the best known algorithm in the offline setting. In fact, in the online setting, Leonardi and Raz showed that no algorithm can achieve a better bound.

Without migration, no (offline or online) approximations are known. This paper introduces a new algorithm that does not use migration, works online, and is just as effective (in terms of approximation ratio) as the best known offline algorithm that uses migration.

Key words. scheduling, flow time, migration, approximation, online

AMS subject classifications. 68Q25, 68W25, 68M20, 90B35

PII. S009753970037446X

1. Introduction.

Objectives. One of the most basic performance measures in multiprocessor scheduling problems is the overall time the jobs are spending in the system. This includes the delay of waiting for service as well as the actual service time. This measure captures the overall quality of service of the system. Multiprocessor scheduling problems arise, for example, in the context of server farms accommodating requests for retrieving Web contents over the Internet.

We consider the classical problem of minimizing the flow time in a multiprocessor setting with jobs which are released over time. The performance of the algorithm, both

*Received by the editors June 26, 2000; accepted for publication (in revised form) January 16, 2002; published electronically May 29, 2002. A preliminary version of this paper appears in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, Atlanta, GA, 1999, pp. 198–205.

http://www.siam.org/journals/sicomp/31-5/37446.html

†Johns Hopkins University, Baltimore, MD 21218 (baruch@blaze.cs.jhu.edu). This author's research was supported by Air Force Contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

‡Department of Computer Science, Tel Aviv University, Tel-Aviv, 69978, Israel (azar@tau.ac.il, odedr@math.tau.ac.il). The research of the second author was supported in part by the Israel Science Foundation, by the US-Israel Binational Science Foundation (BSF), and by the IST Program of the EU.

§Dipartimento di Informatica Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198-Roma, Italia (leon@dis.uniroma1.it). This author's research was partially supported by the IST Programme of the EU under contract IST-1999-14186 (ALCOM-FT) and IST-2000-14084 (APPOL), and by the MURST Projects "Algorithms for Large Data Sets: Science and Engineering" and "Resource Allocation in Computer Networks."

in offline and online settings, can be significantly improved if we allow preemption, i.e., interrupt a job and later continue its execution, perhaps migrating it to a different machine. As shown below, preemption is inherent to make a scheduling algorithm efficient.

While in the case of a single processor, most operating systems can easily handle preemptions, migrating a job to a different machine results in a huge overhead. Thus, it is not commonly used in most multiprocessor operating systems. The natural question is whether migration is an inherent component for an efficient scheduling algorithm in either the online or offline setting.

Existing work. Surveys on approximation algorithms for scheduling can be found in [4, 7]. In the nonpreemptive case it is impossible to achieve a "reasonable" approximation. Specifically, even for one machine one cannot achieve an approximation factor of $O(n^{\frac{1}{2}-\epsilon})$ unless $NP = P$, where n is the number of jobs [6]. For $m > 1$ it is impossible to achieve an $O(n^{\frac{1}{3}-\epsilon})$ approximation factor unless $NP = P$ [8]. Thus, preemptions really seem to be essential.

Existing work: Single processor. Minimizing the flow time on one machine with preemption can be done optimally in polynomial time using the natural algorithm shortest remaining processing time (SRPT) [1].

Existing work: Multiple processors with migration. For more than one machine the preemptive problem becomes NP-hard [2]. Only very recently, Leonardi and Raz [8] showed that SRPT achieves logarithmic approximation for the multiprocessor case, showing a tight bound of $O(\log(\min\{n/m, P\}))$ on $m > 1$ machines with n jobs, where P denotes the ratio between the processing time of the longest and the shortest jobs.

Note that SRPT must use both preemption and migration to schedule the jobs. In the offline setting, it is not known if better approximation factors can be reached. In fact, in the online setting SRPT is optimal; i.e., no algorithm can achieve a better bound up to a constant factor [8]. For the easier problem of minimizing the total completion time a constant approximation can be achieved [3].

Existing work: Multiple processors without migration. In a recent paper by Kalyanasundaram and Pruhs [5] an algorithm is shown that converts any multiprocessor preemptive schedule to a nonmigratory one. However, it is assumed that the number of processors available is a constant factor more than in the original schedule. An important property of their algorithm is that the jobs are scheduled to begin later than their original start time and are completed before their original completion time. Thus, the total flow time is not increased by their transformation. By combining their methods with a migratory algorithm such as SRPT, one can achieve a logarithmic factor approximation with the assumption that a constant factor more machines are available.

Our result: Multiple processors without migration. Without migration, no (offline or online) approximations are known (without increasing the number of processors). We present in section 2 a new algorithm for minimizing flow time that uses local preemption but does not migrate jobs between machines. We show in section 3 that our algorithm performs as well as the best known offline algorithm (SRPT) for the preemptive problem that uses migration. More specifically, our algorithm guarantees, on all input instances, a small performance gap in comparison to the optimal offline schedule that allows both preemption and migration. Denote by P the ratio between the processing time of the longest and the shortest jobs. Then our algorithm performs by at most an $O(\min\{\log P, \log n\})$ factor of the optimal flow time of any (possibly

migratory) schedule. The algorithm can be easily implemented in polynomial time in the size of the input instance.

Our algorithm is also online. We note that in the proof of the $\Omega(\log P)$, $\Omega(\log(n/m))$ lower bounds of [8] for online algorithms, the optimal algorithm does not use migration. Hence, the (randomized) lower bound holds also for a nonmigratory algorithm. This implies that our algorithm is optimal with respect to the parameter P; i.e., no online algorithm can achieve a better bound both when the offline algorithm is or is not allowed to migrate jobs (the first claim is obviously stronger), while there is still a small gap between the $O(\log n)$ upper bound and the $\Omega(\log(n/m))$ lower bound for large m. An $\Omega(\log n)$ competitive lower bound for our algorithm is also proved in section 4 for any m.

Our algorithm and its analysis draw on some ideas from the SRPT algorithm [8]. Moreover, we eliminated several difficulties in the analysis by basing the analysis on classes of jobs. Basically, our algorithm prefers short jobs over long jobs. However, unlike SRPT, our algorithm may continue to run a job on a machine even when a shorter job is waiting to be processed. This seems essential in the nonmigratory setting since being too eager to run a shorter job may result in an unbalanced commitment to machines. A nonmigratory algorithm has to trade off between the commitment of a job to a machine and the decrease in the flow time yielded by running a shorter job. Our algorithm runs the job with the SRPT among all the jobs that were already assigned to that machine, and a new job is assigned to a machine if its processing time is considerably shorter than the job that is currently running.

The model. We are given a set J of n jobs and a set of m identical machines. Each job j is assigned a pair (r_j, p_j), where r_j is the release time of the job and p_j is its processing time. In the preemptive model a job that is running can be preempted and continued later on any machine. Our model allows preemption but does not allow migration; i.e., a job that is running can be preempted but must later continue its execution on the same machine on which its execution began. The scheduling algorithm decides which of the jobs should be executed at each time. Clearly, a machine can process at most one job in any given time and a job cannot be processed before its release time. For a given schedule define c_j to be the completion time of job j in this schedule. The flow time of job j for this schedule is $F_j = c_j - r_j$. The total flow time is $\sum_{j \in J} F_j$. The goal of the scheduling algorithm is to minimize the total flow time for each given instance of the problem. In the offline version of the problem all the jobs are known in advance. In the online version of the problem each job is introduced at its release time, and the algorithm bases its decision only upon the jobs that were already released.

2. The algorithm. A job is called alive at time t for a given schedule if it has already been released but has not been completed yet. Our algorithm classifies the jobs that are alive into classes according to their remaining processing times. A job j whose remaining processing time is in $[2^k, 2^{k+1})$ is in class k for $-\infty < k < \infty$. Notice that a given job changes its class during its execution. The algorithm holds a pool of jobs that are alive and have not been processed at all. In addition, the algorithm holds a stack of jobs for each of the machines. The stack of machine i holds jobs that are alive and have already been processed by machine i. The algorithm works as follows:

- Each machine processes the job at the top of its stack.

- When a new job arrives the algorithm looks for a machine that is idle or currently processing a job of a higher class than the new job. In case it finds one, the new job is pushed into the machine's stack and its processing begins. Otherwise, the job is inserted into the pool.
- When a job is completed on some machine it is popped from its stack. The algorithm compares the class of the job at the top of the stack with the minimum class of a job in the pool. If the minimum is in the pool, then a job that achieves the minimum is pushed into the stack (and removed from the pool).

Clearly, when a job is assigned to a machine it will be processed only on that machine and thus the algorithm does not use migration. In fact, the algorithm bases its decisions only on the jobs that were released up to the current time and hence is an online algorithm. Note that it may seem that the algorithm has to keep track of all the infinite number of classes through which a job evolves. However, the algorithm recalculates the classes of jobs only at arrival or completion of a job.

3. Analysis. We denote by A our scheduling algorithm and by OPT the optimal offline algorithm that minimizes the flow time for any given instance. For our analysis we can even assume that OPT may migrate jobs between machines. Whenever we talk about time t we mean the moment after the events of time t happened. For a given scheduling algorithm S we define $V^S(t)$ to be the volume of a schedule at a certain time t. This volume is the sum of all the remaining processing times of jobs that are alive. In addition, we define $\delta^S(t)$ to be the number of jobs that are alive. $\Delta V(t)$ is defined to be the volume difference between our algorithm and the optimal algorithm, i.e., $V^A(t) - V^{OPT}(t)$. We also define by $\Delta\delta(t) = \delta^A(t) - \delta^{OPT}(t)$ the alive jobs difference at time t between A and OPT. For a generic function f (V, ΔV, δ or $\Delta\delta$) we use $f_{\geq h,\leq k}(t)$ to denote the value of f at time t when restricted to jobs of classes between h and k. Similarly, the notation $f_{=k}(t)$ will represent the value of function f at time t when restricted to jobs of class precisely k.

Let $\gamma^S(t)$ be the number of nonidle machines at time t. Notice that because our algorithm does not migrate jobs, there are situations in which $\gamma^A(t) < m$ and $\delta^A(t) \geq m$. We denote by $\mathcal{T}$ the set of times in which $\gamma^A(t) = m$, that is, the set of times in which none of the machines is idle. Denote by $P_{\min}$ the processing time of the shortest job and by $P_{\max}$ the processing time of the longest job. Note that $P = P_{\max}/P_{\min}$. Denote by $k_{\min} = \lfloor \log P_{\min} \rfloor$ and $k_{\max} = \lfloor \log P_{\max} \rfloor$ the classes of the shortest and longest jobs upon their arrival.

We start by observing the simple fact that the flow time is the integral over time of the number of jobs that are alive. (For example, see [8].)

FACT 3.1. *For any scheduler S,*

$$F^S = \int_t \delta^S(t)dt.$$

First we note that the algorithm preserves the following property of the stacks.

LEMMA 3.2. *In each stack the jobs are ordered in a strictly increasing class order and there is at most one job whose class is at most $k_{\min}$.*

Proof. At time $t = 0$ the lemma is true since all the stacks are empty. The lemma is proved by induction on time. The classes of jobs in the stacks change in one of three cases. The first is when the class of the currently processed job decreases. Since the currently processed job is the job with the lowest class, the lemma remains true.

The second case is when a new job arrives. In case it enters the pool there is no change in any stack. Otherwise, it is pushed into a stack whose top is of a higher class which preserves the first part of the lemma. Since the class of the new job is at least $k_{\min}$, the second part of the lemma remains true as well. The third case is when a job is completed on some machine. If no job is pushed into the stack of that machine the lemma remains easily true. If a new job is pushed to the stack, then the lemma remains true in much the same way as in the case of the arrival of a new job. □

COROLLARY 3.3. *There are at most* $2 + \log P$ *jobs in each stack.*

Proof. The number of classes of jobs in each stack is at most $k_{\max} - k_{\min} + 1 \leq 2 + \log P$. □

We look at the state of the schedule at a certain time t. First let us look at $t \notin \mathcal{T}$.

LEMMA 3.4. *For* $t \notin \mathcal{T}$, $\delta^A(t) \leq \gamma^A(t)(2 + \log P)$.

Proof. By definition of $\mathcal{T}$, at time t at least one machine is idle. This implies that the pool is empty. Moreover, all the stacks of the idle machines are obviously empty. So, all the jobs that are alive are in the stacks of the nonidle machines. The number of nonidle machines is $\gamma^A(t)$, and the number of jobs in each stack is at most $2 + \log P$ according to Corollary 3.3. □

Now, assume that $t \in \mathcal{T}$ and let $\hat{t} < t$ be the earliest time for which $[\hat{t}, t) \subset \mathcal{T}$. We denote the last time in which a job of class more than k was processed by t_k. In case such jobs were not processed at all in the time interval $[\hat{t}, t)$ we set $t_k = \hat{t}$. So, $\hat{t} \leq t_{k_{\max}} \leq t_{k_{\max}-1} \leq \cdots \leq t_{k_{\min}} \leq t$.

LEMMA 3.5. *For* $t \in \mathcal{T}$, $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(t_k)$.

Proof. Notice that in the time interval $[t_k, t)$, algorithm A is constantly processing on all the machines jobs whose class is at most k. The offline algorithm may process jobs of higher classes. Moreover, that can cause jobs of class more than k to actually lower their classes to k and below therefore adding even more to $V^{OPT}_{\leq k}(t)$. Finally, the release of jobs of class $\leq k$ in the interval $[t_k, t)$ is not affecting $\tilde{\Delta} V_{\leq k}(t)$. Therefore, the difference in volume between the two algorithms cannot increase. □

LEMMA 3.6. *For* $t \in \mathcal{T}$, $\Delta V_{\leq k}(t_k) \leq m2^{k+2}$.

Proof. First we claim that at any moment $t_k - \epsilon$, for any $\epsilon > 0$ small enough, the pool does not contain jobs whose class is at most k. In case $t_k = \hat{t}$, at any moment just before t_k there is at least one idle machine which means the pool is empty. Otherwise, $t_k > \hat{t}$ and by definition we know that a job of class more than k is processed just before t_k. Therefore, the pool does not contain any job whose class is at most k.

At time t_k jobs of class at most k might arrive and fill the pool. However, these jobs increase both $V^{OPT}_{\leq k}(t_k)$ and $V^{A}_{\leq k}(t_k)$ by the same amount, so jobs that arrive exactly at t_k do not change $\Delta V_{\leq k}(t_k)$ and can be ignored.

Since the jobs in the pool at time t_k can be ignored, we are left with the jobs in the stacks. Using Lemma 3.2, $\Delta V_{\leq k}(t_k) \leq m(2^{k+1} + 2^k + 2^{k-1} + \cdots) \leq m2^{k+2}$. □

LEMMA 3.7. *For* $t \in \mathcal{T}$, $\Delta V_{\leq k}(t) \leq m2^{k+2}$.

Proof. Combining Lemma 3.5 and 3.6, we obtain $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(t_k) \leq m2^{k+2}$. □

The claim of the following lemma states a property that will be used in the proof of both the $O(\log P)$ and the $O(\log n)$ approximation results.

LEMMA 3.8. *For* $t \in \mathcal{T}$, *for* $k_{min} \leq k_1 \leq k_2 \leq k_{max}$, $\delta^A_{\geq k_1, \leq k_2}(t) \leq 2m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\leq k_2}(t)$.

Proof. We note that $\delta^A_{\geq k_1,\leq k_2}(t)$ can be expressed as

$$\begin{aligned}
\sum_{i=k_1}^{k_2} \delta^A_{=i}(t) &\leq \sum_{i=k_1}^{k_2} \frac{\Delta V_{=i}(t) + V^{OPT}_{=i}(t)}{2^i} \\
&\leq \sum_{i=k_1}^{k_2} \frac{\Delta V_{\leq i}(t) - \Delta V_{\leq i-1}(t)}{2^i} + 2\delta^{OPT}_{\geq k_1,\leq k_2}(t) \\
&\leq \frac{\Delta V_{\leq k_2}(t)}{2^{k_2}} + \sum_{i=k_1}^{k_2-1} \frac{\Delta V_{\leq i}(t)}{2^{i+1}} - \frac{\Delta V_{\leq k_1-1}(t)}{2^{k_1}} + 2\delta^{OPT}_{\geq k_1,\leq k_2}(t) \\
&\leq 4m + \sum_{i=k_1}^{k_2-1} 2m + \delta^{OPT}_{\leq k_1-1}(t) + 2\delta^{OPT}_{\geq k_1,\leq k_2}(t) \\
&\leq 2m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\leq k_2}(t).
\end{aligned}$$

The first inequality follows since 2^i is the minimum processing time of a job of class i. The second inequality follows since the processing time of a job of class i is less than 2^{i+1}. The fourth inequality is derived by applying Lemma 3.7, observing that $\Delta V_{\leq k_1-1}(t) \geq -V^{OPT}_{\leq k_1-1}(t)$ and that 2^{k_1} is the maximum processing time of a job of class at most $k_1 - 1$. The claim of the lemma then follows. ∎

The following corollary of Lemma 3.8 is used in the proof of the $O(\log P)$ approximation ratio of Theorem 3.10.

COROLLARY 3.9. *For $t \in \mathcal{T}$, $\delta^A(t) \leq 2m(4 + \log P) + 2\delta^{OPT}(t)$.*

Proof. We express

$$\begin{aligned}
\delta^A(t) &= \delta^A_{\leq k_{\max},\geq k_{\min}}(t) + \delta^A_{<k_{\min}}(t) \\
&\leq 2m(k_{\max} - k_{\min} + 2) + 2\delta^{OPT}(t) + m \\
&\leq 2m(4 + \log P) + 2\delta^{OPT}(t).
\end{aligned}$$

The second inequality follows from the claim of Lemma 3.8, when $k_2 = k_{\max}$ and $k_1 = k_{\min}$, and from the claim of Lemma 3.2 stating that the stack of each machine contains at most one job of class less than $k_{\min}$. The third inequality is obtained since $k_{\max} - k_{\min} + 5/2 \leq \log P + 4$. ∎

THEOREM 3.10. *$F^A \leq 2(5 + \log P) \cdot F^{OPT}$; that is, algorithm A has a $2(5 + \log P)$ approximation factor even compared to the flow time of the (possibly migratory) schedule of the optimal offline algorithm.*

Proof.

$$\begin{aligned}
F^A &= \int_t \delta^A(t)dt \\
&= \int_{t\notin\mathcal{T}} \delta^A(t)dt + \int_{t\in\mathcal{T}} \delta^A(t)dt \\
&\leq \int_{t\notin\mathcal{T}} \gamma^A(t)(2 + \log P)dt + \int_{t\in\mathcal{T}} (2m(4 + \log P) + 2\delta^{OPT}(t))dt \\
&\leq (2 + \log P) \int_{t\notin\mathcal{T}} \gamma^A(t)dt + 2(4 + \log P) \int_{t\in\mathcal{T}} m dt + 2\int_{t\in\mathcal{T}} \delta^{OPT}(t)dt \\
&\leq (8 + 2\log P) \int_t \gamma^A(t)dt + 2\int_t \delta^{OPT}(t)dt \\
&\leq 2(5 + \log P) \cdot F^{OPT}.
\end{aligned}$$

The first equality is from the definition of F^A. The second is obtained by looking at the time in which none of the machines is idle and the time in which at least one machine is idle separately. The third inequality uses Lemma 3.4 and Corollary 3.9. The fifth inequality is true since $\gamma^A(t) = m$ when $t \in \mathcal{T}$. Finally, $\int_t \gamma^A(t)dt$ is the total time spent processing jobs by the machines which is exactly $\sum_{j \in J} p_j$. That sum is upper bounded by the flow time of OPT since each job's flow time must be at least its processing time. □

We now turn to prove the $O(\log n)$ approximation ratio of the algorithm. A different argument is required to prove this second bound. The main idea behind the proof of the $O(\log P)$ approximation ratio was to bound for any time $t \in \mathcal{T}$ the alive jobs difference between A and OPT by $O(m \log P)$. A similar approach does not allow us to prove the $O(\log n)$ approximation ratio: It is possible to construct instances where the alive jobs difference is $\Omega(n)$. The proof that follows uses ideas drawn from the proof given by Leonardi and Raz [8] of the $O(\log(n/m))$ approximation ratio for SRPT when migration is allowed. The idea behind the proof is that the worst case ratio between the algorithm's flow time and the optimal flow time can be raised only if a "big" alive jobs difference is kept for a "long" time period.

We need to introduce more notation. Recall that $\mathcal{T}$ is defined to be the set of times in which $\gamma^A(t) = m$. We denote by $T = \int_{t \in \mathcal{T}} dt$ the size of set $\mathcal{T}$. For any $t \in \mathcal{T}$, define by $\delta^{A,P}(t)$ the number of jobs in the pool of algorithm A, i.e., not assigned to a machine, at time t, and by $\Delta\delta^P(t) = \delta^{A,P}(t) - 2\delta^{OPT}(t)$ the difference between the number of jobs in the pool of algorithm A and twice the number of jobs not finished by the optimal algorithm. For any machine l, time t, define by $\delta^{A,l}(t)$ the number of jobs assigned to machine l at time t in the schedule of algorithm A. Moreover, define by $\mathcal{T}^l = \{t | \delta^{A,l}(t) > 0\}$ the set of times when machine l is assigned with at least one job and by $T^l = \int_{t \in \mathcal{T}^l} dt$ the size of set $\mathcal{T}^l$.

LEMMA 3.11. *For any time $t \in \mathcal{T}$, if $\Delta\delta^P(t) \geq 2mi$, for $i \geq 3$, then the pool of algorithm A contains at least $2m$ jobs of remaining processing time at most $\frac{V^A(t)}{m2^{i-3}}$.*

Proof. Let k_{high} be the maximum integer such that $\delta^{A,P}_{\geq k_{high}}(t) \geq 2m$ and let k_{low} be the maximum integer such that $\delta^{A,P}_{< k_{low}}(t) < 2m$. Note that both numbers are well defined and $k_{low} \leq k_{high}$ since there are at least $6m > 2m$ jobs in the pool. Then,

$$2m \leq \delta^{A,P}_{\geq k_{high}}(t) \leq \delta^{A}_{\geq k_{high}}(t) \leq \frac{V^A(t)}{2^{k_{high}}},$$

thus yielding $2^{k_{high}} \leq \frac{V^A(t)}{2m}$. In particular, the last inequality follows since $2^{k_{high}}$ is the minimum processing time of a job of class k_{high}.

By the definition of k_{high}, we have

$$\begin{aligned}
\Delta\delta^P_{\leq k_{high}}(t) &= \delta^{A,P}_{\leq k_{high}}(t) - 2\delta^{OPT}_{\leq k_{high}}(t) \\
&= \delta^{A,P}(t) - \delta^{A,P}_{> k_{high}}(t) - 2(\delta^{OPT}(t) - \delta^{OPT}_{> k_{high}}(t)) \\
&= \Delta\delta^P(t) - \delta^{A,P}_{> k_{high}}(t) + 2\delta^{OPT}_{> k_{high}}(t) \\
&\geq 2m(i-1),
\end{aligned}$$

where the last inequality follows since $\delta^{A,P}_{> k_{high}}(t) < 2m$.

From Lemma 3.8, we get

$$\begin{aligned}\Delta\delta^P_{\le k_{high}}(t) &= \delta^{A,P}_{\le k_{high},\ge k_{low}}(t) + \delta^{A,P}_{<k_{low}}(t) - 2\delta^{OPT}_{\le k_{high}}(t)\\ &\le \delta^{A}_{\le k_{high},\ge k_{low}}(t) + \delta^{A,P}_{<k_{low}}(t) - 2\delta^{OPT}_{\le k_{high}}(t)\\ &\le \delta^{A,P}_{<k_{low}}(t) + 2m(k_{high} - k_{low} + 2),\end{aligned}$$

thus yielding $\delta^{A,P}_{<k_{low}}(t) \ge 2m(i + k_{low} - k_{high} - 3)$. Therefore, we get that $2m > 2m\ (i + k_{low} - k_{high} - 3)$ and thus $k_{low} \le k_{high} - i + 3$. It follows that there exist at least $2m$ jobs of class at most $k_{low} \le k_{high} - i + 3$ in the pool of the algorithm. The remaining processing time of these $2m$ jobs is bounded by $2^{k_{high}-i+4} \le \frac{V^A(t)}{m2^{i-3}}$, thus proving the claim. ∎

LEMMA 3.12. *For any machine l, time $t \in \mathcal{T}^l$, if $\delta^{A,l}(t) \ge i$ for $i \ge 1$, then there exists a job with remaining processing time at most $\frac{T^l}{2^{i-2}}$ assigned to machine l at time t.*

Proof. For any time $t \in \mathcal{T}^l$, there is at most one job assigned to machine l for every specific class. Assume k to be the highest class of a job assigned to machine l, obviously satisfying $T^l \ge 2^k$. If $\delta^{A,l}(t) \ge i$, then there is a job of class at most $k - i + 1$ assigned to machine l, with processing time at most $2^{k-i+2} \le \frac{T^l}{2^{i-2}}$. ∎

At this stage of the exposition we give a brief overview of the proof. Roughly speaking, Lemmas 3.11 and 3.12 show that if the alive jobs difference between the algorithm and the optimum is order of mi, then the remaining processing time of the jobs on execution is proportional to $1/2^i$. This implies, as we will show in Lemmas 3.14 and 3.15, that one new job is released for every nonidle machine in every interval of size proportional to $1/2^i$. This fact is exploited in Theorem 3.17 to argue that the integral of the alive jobs difference between A and OPT (e.g., order of mi) is logarithmic in the number of jobs that are released (i.e., order of 2^i).

We partition the set of time instants $\mathcal{T}$ when no machine is idle into a collection of disjoint intervals $I_k = [t_k, r_k)$, $k = 1, \ldots, s$, and associate an integer i_k with each interval such that for any time $t \in I_k$, $2m(i_k - 1) < \Delta\delta^P(t) < 2m(i_k + 2)$.

Each maximal time interval $[t_b, t_e)$ contained in $\mathcal{T}$ is dealt with separately. Assume we already dealt with all times in $\mathcal{T}$ which are smaller than t_b, and we have created $k - 1$ intervals. We then define $t_k = t_b$. Given t_k we choose $i_k = \lfloor \frac{\Delta\delta^P(t_k)}{2m} \rfloor$. Given an interval's t_k and i_k, we define

$$r_k = \min\{t_e, t | t > t_k, \Delta\delta^P(t) \ge 2m(i_k + 2) \text{ or } \Delta\delta^P(t) \le 2m(i_k - 1)\};$$

that is, r_k is the first time $\Delta\delta^P(t)$ reaches the value $2m(i_k+2)$ or the value $2m(i_k-1)$. As long as $r_k < t_e$, we continue with the next interval beginning at $t_{k+1} = r_k$.

OBSERVATION 3.13. *When an interval k begins, $2mi_k \le \Delta\delta^P(t_k) < 2m(i_k + 1)$. When it ends, either $\Delta\delta^P(r_k) \le 2m(i_k - 1)$, $\Delta\delta^P(r_k) \ge 2m(i_k + 2)$, or $\Delta\delta^P(r_k) \le 0$.*

Proof. The first part is clear by the way we choose i_k. The second part is clear when r_k is not equal to some t_e, that is, $t_k \in \mathcal{T}$. Otherwise, $r_k \notin \mathcal{T}$ and therefore $\delta^{A,P}(r_k) = 0$ and $\Delta\delta^P(r_k) \le 0$. ∎

Denote by $x_k = r_k - t_k$ the size of interval I_k and define $\mathcal{T}_i = \{\cup I_k | i_k = i\}$, $i \ge 1$, as the union of the intervals I_k with $i_k = i$. We indicate by $T_i = \int_{t\in\mathcal{T}_i} dt$ the size of set $\mathcal{T}_i$. We also denote by $D = \max\{T, \max_{t\in\mathcal{T}}\{V^A(t)/m\}\}$. The following lemma relates the number of jobs, n, and the values of T_i.

LEMMA 3.14. *The following lower bound holds for the number of jobs:*

$$n \geq \frac{m}{8D} \sum_{i \geq 4} T_i \; 2^{i-5}.$$

Proof. We prove the lemma by associating with every interval a set of jobs that have been, during the interval itself, either released, or completed by OPT, or moved to execution or completed by algorithm A. Consider an interval I_k, with a corresponding $i_k \geq 4$. According to Observation 3.13, when the interval starts $\Delta\delta^P(t_k)$ is between $2mi_k$ and $2m(i_k+1)$. This interval ends when $\Delta\delta^P(r_k)$ goes above $2m(i_k+2)$ or below $2m(i_k-1)$. (It might also reach 0 but $0 < 2m(i_k-1)$.) In the first case we have the evidence of at least m jobs finished by OPT. (Recall that $\Delta\delta^P(t) = \delta^{A,P}(t) - 2\delta^{OPT}(t)$.) In the second case we have the evidence of at least $2m$ jobs that either leave the pool to be assigned to a machine by algorithm A or arrive to both A and OPT. In both cases we charge $n_k \geq m$ jobs to interval I_k. We can then conclude with a first lower bound $n_k \geq m$ on the number of jobs charged to any interval $I_k \in \mathcal{T}_i$, $i_k \geq 4$.

Next, we give a second lower bound, based on Lemma 3.11, stating that during an interval $I_k = [t_k, r_k)$ there exist in the pool $2m$ jobs with remaining processing time at most $\frac{D}{2^{i_k-4}}$, since $\Delta\delta^P(t) > 2m(i_k - 1)$ for any $t \in [t_k, r_k)$. This implies that all the m machines are processing jobs with remaining processing time at most $\frac{D}{2^{i_k-5}}$. We look at any subinterval of I_k of length $\frac{D}{2^{i_k-5}}$. For each machine, during this subinterval, a job is either finished by the algorithm or is preempted by a job of lower class. Therefore, we can charge at least m jobs that are either released or finished with any subinterval of size $\frac{D}{2^{i_k-5}}$ of I_k. A second lower bound on the number of jobs charged to any interval is then given by $n_k \geq m\lfloor \frac{x_k 2^{i_k-5}}{D} \rfloor$.

Now observe that each job is charged at most four times: when it is released, when it is assigned to a machine by A, when it is finished by A, and when it is finished by OPT. Then,

$$\frac{m}{2D} \sum_{i \geq 4} T_i \; 2^{i-5} \leq \sum_{k | i_k \geq 4} m \; \max\left\{1, \left\lfloor \frac{x_k 2^{i_k-5}}{D} \right\rfloor \right\} \leq 4n,$$

where the first inequality is obtained by summing over I_k's instead of over T_i's and the simple fact that $\alpha \leq \max\{1, \lfloor 2\alpha \rfloor\}$ and the second inequality follows from the lower bounds we have shown on the charged jobs. The lemma easily follows. ∎

We next bound the number of jobs that are assigned to a machine l during the time instants of $\mathcal{T}^l$. We partition the set of time instants $\mathcal{T}^l$ into a set of disjoint intervals $I_k^l = [t_k^l, r_k^l)$, $k = 1, \ldots, s^l$, and associate an integer i_k^l with each interval such that for any time $t \in I_k^l$, $\delta^{A,l}(t) = i_k^l$. Consider a maximal interval of times $[t_b^l, t_e^l)$ contained in $\mathcal{T}^l$. Assume $t_k^l = t_b^l$. Given t_k^l and i_k^l, define $r_k^l = \min\{t_e^l, t | t > t_k^l, \delta^{A,l}(t) \neq i_k^l\}$. In case $r_k^l < t_e^l$, we set $t_{k+1}^l = r_k^l$. Denote by $x_k^l = r_k^l - t_k^l$ the size of interval I_k^l. Define by $\mathcal{T}_i^l = \{\cup I_k^l | i_k^l = i\}$, $i \geq 1$, the union of the intervals I_k^l when the number of jobs assigned to machine l is exactly i and by $T_i^l = \int_{t \in \mathcal{T}_i^l} dt$ the size of set $\mathcal{T}_i^l$.

LEMMA 3.15. *The following lower bound holds for the number of jobs assigned to machine l:*

$$n^l \geq \frac{1}{4T^l} \sum_{i \geq 1} T_i^l \; 2^{i-2}.$$

Proof. For each interval I_k^l we charge at least one job. Every job will be charged at most twice: when it is assigned to a machine by A and when it is finished by A.

Consider the generic interval I_k^l, with the corresponding i_k^l. The interval starts when $\delta^{A,l}(t_k)$ reaches i_k^l, from above or from below. The interval ends when $\delta^{A,l}(r_k)$ reaches $i_k^l + 1$ or $i_k^l - 1$. In the first case we have the evidence of one job that is assigned by A to machine l and in the second case of one job that is finished by A. In both cases we charge one job to interval I_k^l.

Next we give a second lower bound, based on Lemma 3.12. Lemma 3.12 states that during an interval $I_k^l = [t_k^l, r_k^l)$, machine l is constantly assigned with a job of remaining processing time at most $T^l/2^{i_k^l-2}$. We look at any subinterval of I_k^l of length $T^l/2^{i_k^l-2}$. During this subinterval, a job is either finished by the algorithm or is preempted by a job of a lower class. In any case, a job that is assigned or finished during any subinterval of size $T^l/2^{i_k^l-2}$ is charged. A second lower bound on the number of jobs charged to any interval is then given by $n_k \geq \lfloor x_k^l 2^{i_k^l-2}/T^l \rfloor$.

Now observe that each job is considered at most twice: when it is assigned to machine l and when it is finished by A. Then, from the inequalities

$$\frac{1}{2T^l} \sum_{i \geq 1} T_i^l \; 2^{i-2} \leq \sum_{k | i_k^l \geq 1} \max \left\{ 1, \left\lfloor \frac{x_k^l 2^{i_k^l-2}}{T^l} \right\rfloor \right\} \leq 2n^l,$$

the claim follows. □

Before completing the proof, we still need a simple mathematical lemma.

LEMMA 3.16. *Given a sequence $a_1, a_2, \ldots$ of nonnegative numbers such that $\sum_{i \geq 1} a_i \leq A$ and $\sum_{i \geq 1} 2^i a_i \leq B$, then $\sum_{i \geq 1} i a_i \leq \log(4B/A)A$.*

Proof. Define a second sequence, $b_i = \sum_{j \geq i} a_j$ for $i \geq 1$. Then it is known that $A \geq b_1 \geq b_2 \geq \cdots$. Also, it is known that $\sum_{i \geq 1} 2^i a_i = \sum_{i \geq 1} 2^i (b_i - b_{i+1}) = \frac{1}{2} \sum_{i \geq 1} 2^i b_i + b_1$. This implies that $\sum_{i \geq 1} 2^i b_i \leq 2B$.

The sum we are trying to upper bound is $\sum_{i \geq 1} b_i$. This can be viewed as an optimization problem where we try to maximize $\sum_{i \geq 1} b_i$ subject to $\sum_{i \geq 1} 2^i b_i \leq 2B$ and $b_i \leq A$ for $i \geq 1$. This corresponds to the maximization of a continuous function in a compact domain and any feasible point where $b_i < A, b_{i+1} > 0$ is dominated by the point we get by replacing b_i, b_{i+1} with $b_i + 2\epsilon, b_{i+1} - \epsilon$. Therefore, it is upper bounded by assigning $b_i = A$ for $1 \leq i \leq k$ and $b_i = 0$ for $i > k$, where k is large enough such that $\sum_{i \geq 1} 2^i b_i \geq 2B$. A choice of $k = \lceil \log(2B/A) \rceil$ is adequate, and the sum is upper bounded by kA from which the result follows. □

THEOREM 3.17. *$F^A = O(\log n) F^{OPT}$; that is, algorithm A has an $O(\log n)$ approximation ratio even compared with the flow time of the (possibly migratory) schedule of the optimal offline algorithm.*

Proof.

$$\begin{aligned}
F^A &= \int_t \delta^A(t) dt \\
&= \sum_{l=1}^m \int_t \delta^{A,l}(t) dt + \int_{t \in \mathcal{T}} \delta^{A,P}(t) dt \\
&= \sum_{l=1}^m \int_{t \in \mathcal{T}^l} \delta^{A,l}(t) dt + \int_{t \in \mathcal{T}} (2\delta^{OPT}(t) + \Delta\delta^P(t)) dt \\
&\leq \sum_{l=1}^m \int_{t \in \mathcal{T}^l} \delta^{A,l}(t) dt + 2F^{OPT} + \sum_i \int_{t \in \mathcal{T}_i} 2m(i+2) dt
\end{aligned}$$

$$\leq \sum_{l=1}^{m} \sum_{i\geq 1} i\ T_i^l + 2F^{OPT} + 2m \sum_i (i+2)\ T_i$$
$$= \sum_{l=1}^{m} \sum_{i\geq 1} i\ T_i^l + 2F^{OPT} + 2m \sum_i (i-3)\ T_i + 2m \sum_i 5\ T_i$$
$$\leq \sum_{l=1}^{m} \sum_{i\geq 1} i\ T_i^l + 2F^{OPT} + 2m \sum_{i\geq 4} (i-3)\ T_i + 10F^{OPT}$$
$$\leq \sum_{l=1}^{m} \sum_{i\geq 1} i\ T_i^l + 12F^{OPT} + 2m \sum_{i\geq 1} i\ T_{i+3}.$$

The second equality is obtained by separately considering the contribution of every machine l and the contribution of the jobs in the pool. The fourth inequality is obtained by partitioning $\mathcal{T}$ into the $\mathcal{T}_i$'s such that at any time $t \in \mathcal{T}_i$, $\Delta\delta^P(t) < 2m(i+2)$. The seventh inequality is obtained by observing that $m\sum_i T_i \leq \sum_j p_j \leq F^{OPT}$, since all machines are busy processing jobs at any time $t \in \mathcal{T}$.

We upper bound $\sum_{i\geq 1} i\ T_{i+3}$ under the constraint $\sum_{i\geq 1} T_{i+3} \leq T \leq D$ and the constraint on n given by Lemma 3.14. By choosing $a_i = T_{i+3}$ we see that $\sum_{i\geq 1} a_i \leq D$ and $\sum_{i\geq 1} 2^i a_i \leq 32\frac{n}{m}D$. These two bounds together with Lemma 3.16 result in the upper bound $\sum_{i\geq 1} i a_i \leq (7 + \log\frac{n}{m})D$. A similar argument is used for the other sum. We choose $a_i = T_i^l$ instead. First note that $\sum_{i\geq 1} a_i \leq T^l$. Then, according to Lemma 3.15, $\sum_{i\geq 1} 2^i a_i \leq 16n^l T^l$. The upper bound by Lemma 3.16 is $\sum_{i\geq 1} i a_i \leq (6 + \log n^l)T^l$.

We finally express the total flow time of algorithm A as

$$\begin{aligned} F^A &\leq \sum_{l=1}^{m} \sum_{i\geq 1} iT_i^l + 12F^{OPT} + 2m \sum_{i\geq 1} i\ T_{i+3} \\ &\leq \sum_{l=1}^{m} O(T^l \log n^l) + 12F^{OPT} + O\left(mD \log \frac{n}{m}\right) \\ &= O(\log n) \sum_{l=1}^{m} T^l + 12F^{OPT} + O\left(\log \frac{n}{m}\right) F^{OPT} \\ &= O(\log n) F^{OPT}. \end{aligned}$$

The third equality follows since $mD \leq \max\{mT, \max_{t\in\mathcal{T}}\{V^A(t)\}\} \leq \sum_{j\in J} p_j \leq F^{OPT}$. The fourth equality follows since $\sum_{l=1}^{m} T^l = \sum_j p_j \leq F^{OPT}$. □

4. Tightness of the analysis. In this section we present an $\Omega(\log n)$ lower bound on the competitive ratio of the algorithm analyzed in the previous section. The number of machines is an arbitrary $m \geq 2$ but note that the lower bound uses only the first two machines. While presenting the lower bound we assume an order between jobs with the same release time. The algorithm deals with jobs released at the same time following the specified order.[1] Moreover, we assume that the algorithm arbitrarily decides whether to schedule a job on an idle machine or on a machine

[1] This can be forced by releasing some jobs slightly after previous ones in the specified order and decreasing their processing times appropriately.

processing a job of higher class. The latter assumption is crucial in proving the lower bound, and it agrees with the definition of the algorithm.

We choose P, the ratio between the maximum and the minimum processing time to be a power of 2. The maximum processing time is $P/2$ and the minimum is $1/2$. At each time $r_i = P(1 - \sum_{j=i}^{\frac{1}{2}\log P-1} \frac{1}{2^{2j+1}})$, for $i = 0, \ldots, \frac{1}{2}\log P - 1$, three jobs are released; one job of processing time $\frac{P}{2^{2i+1}}$ followed by two jobs of processing time $\frac{P}{2^{2i+2}}$. Finally, two jobs of size $\frac{1}{2}$ are released every $1/2$ time unit between time $r_{\frac{1}{2}\log P} = P$ and $2P - 1/2$.

Let us consider the behavior of the algorithm and of the optimal solution in this instance. At time r_0 the released job of processing time $P/2$ is assigned to machine 1, and a job of processing time $P/4$ is assigned to machine 2 while the second job of processing time $P/4$ preempts the job on execution on machine 1. The two jobs of processing time $P/4$ are finished at time $r_0 + P/4$, where the job of size $P/2$ is taken again on execution on machine 1. In general, at time r_i, $i = 1, \ldots, \frac{1}{2}\log P$, i jobs are in the queue of machine 1, the smallest of remaining processing time $\frac{P}{2^{2i}}$, while no job is on execution on machine 2. The job on execution on machine 1 is preempted by the job of size $\frac{P}{2^{2i+1}}$ that is released at time r_i. Immediately afterwards it is preempted by a job of size $\frac{P}{2^{2i+2}}$ that is released at time r_i while the other job of size $\frac{P}{2^{2i+2}}$ released at time r_i is scheduled on machine 2.

At time P, $\frac{1}{2}\log P$ jobs are in the queue of machine 1, the smallest of size 1, while no job is in the pool or assigned to the other machines. The two jobs of size $1/2$ released every $1/2$ time unit between time P and $2P - 1/2$ are scheduled on machines 1 and 2. Observe that $\frac{1}{2}\log P$ jobs are waiting on queue 1 between time P and $2P$, thus leading to a flow time of $\Omega(P \log P)$ for the algorithm on this instance of $n = \frac{3}{2}\log P + 4P$ jobs.

On the other hand, an optimal solution schedules the job of size $\frac{P}{2^{2i+1}}$ released at time r_i for $i = 0, \ldots, \frac{1}{2}\log P - 1$ on machine 1 and the two jobs of size $\frac{P}{2^{2i+2}}$ released at time r_i on machine 2. Observe that in this schedule machines 1 and 2 are idle at time r_i. Thus, the jobs of size $\frac{1}{2}$ are scheduled at their release times on machines 1 and 2 and are completed before the next two jobs of size $\frac{1}{2}$ are released. The flow time of the optimal solution is then $O(P)$.

It follows that the ratio between the flow time of the algorithm and the flow time of the optimal solution is $\Omega(\log P)$ that is also $\Omega(\log n)$.

5. Conclusions. In this paper we considered the problem of finding a preemptive schedule that optimizes the total flow time of a set of jobs released over time when job migration is not allowed. We presented a new online algorithm that is almost as effective as the best known algorithm that uses migration [8].

In our algorithm jobs are kept in a pool since the release time until they are assigned to a machine. An interesting open problem is to devise an efficient algorithm that assigns jobs to machines at the time of release. A challenging open problem is also to devise a constant offline approximation algorithm for optimizing total flow time even if both preemption and migration are allowed.

REFERENCES

[1] K.R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

[2] J. Du, J. Y. T. Leung, and G. H. Young, *Minimizing mean flow time with release time constraint,* Theoret. Comput. Sci., 75 (1990), pp. 347–355.

[3] L. Hall, D. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms,* Math. Oper. Res., 22 (1997), pp. 513–544.

[4] L.A. Hall, *Approximation algorithms for scheduling*, in Approximation Algorithms for NP-Hard Problems, D.S. Hochbaum, ed., PWS, Boston, 1997, pp. 1–45.

[5] B. Kalyanasundaram and K. Pruhs, *Eliminating migration in multi-processor scheduling*, J. Algorithms, 38 (2001), pp. 2–24.

[6] H. Kellerer, T. Tautenhahn, and G.J. Woeginger, *Approximability and nonapproximability results for minimizing total flow time on a single machine,* SIAM J. Comput., 28 (1999), pp. 1155–1166.

[7] E.L. Lawler, J.K Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, in Handbooks in Operations Research and Management Science, Vol. 4, North-Holland, Amsterdam, 1993, pp. 445–522.

[8] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, Texas, 1997, pp. 110–119.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1383–1404

PSEUDORANDOM FUNCTIONS AND FACTORING*

MONI NAOR†, OMER REINGOLD‡, AND ALON ROSEN†

Abstract. The computational hardness of factoring integers is the most established assumption on which cryptographic primitives are based. This work presents an efficient construction of *pseudorandom functions* whose security is based on the intractability of factoring. In particular, we are able to construct efficient length-preserving pseudorandom functions, where each evaluation requires only a (small) *constant* number of modular multiplications per output bit. This is substantially more efficient than any previous construction of pseudorandom functions based on factoring and matches (up to a constant factor) the efficiency of the best-known factoring-based *pseudorandom bit generators*.

Key words. cryptography, pseudorandomness, pseudorandom functions, computational number theory, integer factorization

AMS subject classifications. 94A60, 68P25, 11K45, 68Q99

PII. S0097539701389257

1. Introduction.

Almost any interesting cryptographic task must be based on the computational hardness of some problem. Proving such hardness assumptions exceeds by far the state of the art of complexity theory. It is therefore desirable to base the security of a cryptographic construction on as reasonable an assumption as possible. A natural approach is to rely on a well-studied problem where many algorithms have been tried and their complexity is well understood. The most established candidate in these respects, and certainly the one with the best pedigree, is the problem of factoring integers (see [22] for the state of the art of factoring).

The focus of this paper is an efficient construction of *pseudorandom functions* (see definition below) whose security is based on the intractability of factoring. In particular, we are able to construct efficient length-preserving pseudorandom functions whose evaluation requires only a constant number of modular multiplications per output bit. This is substantially more efficient than any previous construction of pseudorandom functions based on factoring and matches (up to a constant factor) the efficiency of the best-known factoring-based *pseudorandom bit generators*.

Pseudorandom functions,[1] originally defined by Goldreich, Goldwasser, and Micali (GGM) [13] are an important cryptographic primitive. A distribution of functions is pseudorandom if it satisfies the following requirements:

- *Easy to sample.* It is easy to sample a function according to the distribution.
- *Easy to compute.* Given such a function, it is easy to evaluate it at any given point.

*Received by the editors May 11, 2001; accepted for publication (in revised form) February 1, 2002; published electronically July 1, 2002. An extended abstract has appeared in the 32nd annual ACM Symposium on Theory of Computing, 2000, pp. 11–20.

http://www.siam.org/journals/sicomp/31-5/38925.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (naor@wisdom.weizmann.ac.il, alon@wisdom.weizmann.ac.il). The first author's research was supported by a grant from the Israel Science Foundation adminstered by the Israel Academy of Sciences.

‡AT&T Labs—Research, 180 Park Avenue, Bldg. 103, Florham Park, NJ 07932 (omer@research.att.com). This author's work was done while he was at the Weizmann Institute, Rehovot, Israel. Research was supported by an Eshkol Fellowship of the Israeli Ministry of Science.

[1]Note the difference between a pseudorandom function and a bit generator—the latter expands a random seed to some fixed length sequence that should be indistinguishable from a random sequence of similar length; there is no "probing" in the attack.

- *Pseudorandom.* It is hard to tell apart a function sampled according to the pseudorandom distribution from a uniformly distributed function when the distinguisher is given access to the function as a black-box.

Pseudorandom functions have a wide range of applications, most notably in cryptography, but also in computational complexity and computational learning theory. Coming up with efficient constructions for such functions is a challenge of great practical and theoretical interest.

The new construction improves the one by Naor and Reingold (NR) [20], who showed how to construct pseudorandom functions based on factoring, where the cost of evaluation is comparable to two modular exponentiations. The drawback of those functions is that the output is only a single bit. In order to apply them for achieving a length-preserving pseudorandom function, one would need to repeat the process n times, rendering it inefficient. The improvement we propose lies in a method to expand the one-bit output of the NR functions to polynomially many bits while paying only a small overhead in the complexity of the evaluation (i.e., one modular multiplication for each additional output bit). This improvement will be achieved through a surprising combination of the NR functions and the Blum–Blum–Shub (BBS) pseudorandom generator [5]. As will be demonstrated in what follows, in general, such a composition does *not* necessarily yield a pseudorandom function. This, in particular, implies a nonstraightforward proof of security.

The method we suggest enables us to construct efficient length-preserving pseudorandom functions which are at least as secure as factoring Blum-integers and can be evaluated at the cost of fewer than three modular exponentiations. This is comparable to another attractive construction by NR [20] of pseudorandom functions which are at least as secure as the *decisional Diffie–Hellman (DDH)* problem. While the DDH problem has received much attention recently (see [4]), it is not nearly as well established as factoring.

Organization. The next section contains the background material and our construction. The main result of our work, the efficient construction of a pseudorandom function which is at least as secure as factoring Blum-integers, is presented in section 4. The proof of security is given in section 5.

2. Old and new constructions.

2.1. Background. When GGM originally defined pseudorandom functions [13], they were at least partly motivated by the construction of the BBS pseudorandom *generator* [5] and, in particular, by an open question suggested there—the *easy-access* problem.[2] Nevertheless, the actual GGM construction of pseudorandom functions did not appear to have any direct connection to the BBS generator (apart from the fact that the BBS generator can be used as a building block for the GGM construction). The current work suggests a construction which is directly related to the BBS generator. We now turn to survey previous constructions of pseudorandom functions (as well as the BBS generator).

[2]The *easy-access* problem arises when one notices that it is easy to access exponentially far away bits in the BBS pseudorandom pad. The question is whether the BBS pad remains pseudorandom even when the distinguisher has access to these exponentially far away bits.

2.1.1. The BBS generator.

The Blum–Micali paradigm (cf. [7, 27]). Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation (i.e., one where it is easy to compute $f(x)$ but intractable to find $x = f^{-1}(y)$), and let $\mathcal{B}(\cdot)$ be a hard-core predicate for $f(\cdot)$ (i.e., given y, it is difficult to guess $\mathcal{B}(f^{-1}(y))$). Let $\ell : \mathbb{N} \to \mathbb{N}$ be a function so that $\ell(n) > n$ for all n. Blum and Micali proposed the following scheme to construct a generator stretching an n-bit seed, x, to an ℓ-bit pseudorandom string:

$$\mathcal{B}(x), \mathcal{B}(f(x)), \ldots, \mathcal{B}(f^{(k)}(x)), \ldots, \mathcal{B}(f^{(\ell(n)-1)}(x)). \tag{1}$$

As a candidate one-way permutation (based on factoring), BBS [5] considered the squaring function modulo an integer $N = P \cdot Q$ (i.e., the mapping $x \mapsto x^2 \bmod N$).[3] Following [5], the values of N are restricted to integers of the form $N = P \cdot Q$, where P and Q are two distinct primes both congruent to 3 mod 4. (Such integers are known as Blum-integers.)

Blum-integers. Restricting N to a Blum-integer enabled BBS to prove that the squaring function is indeed a permutation (when its domain is limited to the subgroup of *quadratic-residues* in $\mathbb{Z}_N^*$). Let $N = P \cdot Q$ be an integer; an element in $\mathbb{Z}_N^*$ is called a quadratic-residue if it has a square-root; namely, there is a $y \in \mathbb{Z}_N^*$ such that $y^2 = x \bmod N$. It is easy to verify that the set of quadratic-residues in $\mathbb{Z}_N^*$ forms a subgroup (which we denote by QR_N). We note that every $x^2 \in QR_N$ has exactly four distinct square-roots, $\pm x, \pm y \in \mathbb{Z}_N^*$, and, in the special case that N is a Blum-integer, it is possible to prove [5] that exactly one of these square-roots resides in QR_N (which implies that squaring is indeed a permutation over QR_N).

Constructing the BBS generator. The BBS pseudorandom generator is obtained by applying the Blum–Micali paradigm to the squaring permutation together with the $\mathcal{LSB}(\cdot)$ (least significant bit) hard-core predicate. This generator has been originally proven secure assuming intractability of the quadratic residuosity problem in [5], and subsequently under the assumption that factoring Blum-integers is hard (Assumption 4.1) in [26] (by adapting the techniques in [1]). Note also that it is the basis for the Blum–Goldwasser public-key encryption scheme [6]. For simplicity of exposition, we choose to replace the $\mathcal{LSB}(\cdot)$ hard-core predicate with the Goldreich–Levin $\mathcal{B}_r(\cdot)$ predicate [14] (where $\mathcal{B}_r(m)$ denotes the inner product, $\langle m, r \rangle \bmod 2$). We obtain a generator which stretches an n-bit seed, $x \in QR_N$, to an ℓ-bit pseudorandom string (and is completely analogous to the BBS generator):[4]

$$\mathcal{B}_r(x), \mathcal{B}_r(x^2), \ldots, \mathcal{B}_r(x^{2^k}), \ldots, \mathcal{B}_r(x^{2^{\ell(n)-1}}). \tag{2}$$

The BBS generator is considered efficient (relative to other generators based on factoring); each bit in its output can be obtained at the cost of one modular multiplication. In particular, by performing $2n$ modular multiplications, it is possible to stretch an n-bit seed to a $2n$-bit pseudorandom string.

[3] It was shown by Rabin [23] that the problem of factoring an integer $N = P \cdot Q$ can be reduced to the problem of extracting square-roots in $\mathbb{Z}_N^*$. Thus, if factoring $N = P \cdot Q$ is hard, then squaring is indeed one-way.

[4] We denote by n the size (in bits) of N and by x^{2^j} the value of $x^{2^j} \bmod N$.

The easy-access problem. In the BBS generator, a seed (x, N) defines an infinite (ultimately periodic) bit-sequence $b_0, b_1, \ldots$ (even though a pseudorandom string generated with an n-bit long seed consists of only polynomially many (in n) bits). An interesting feature of the BBS generator is that knowledge of the factorization of N allows *easy access* to each of the first 2^n bits; that is, if $\log i < n$, the ith bit, b_i, can be computed in poly(n) time (by first computing $\beta_i = 2^{i-1} \bmod \varphi(N)$ and then setting $b_i = \mathcal{B}_r(x^{\beta_i})$). However, as GGM noted [13], this easily accessible exponentially long bit-string *may not* appear "random." What BBS have proved is that any single polynomially long interval of *consecutive* bits in the string is pseudorandom (provided that factoring Blum-integers is hard). Indeed, it might be the case that, say, given $b_1, \ldots, b_n$ and $b_{2\sqrt{n}+1}, \ldots, b_{2\sqrt{n}+n}$, it is easy to compute any other bit in the string.

The *easy-access* problem is whether direct access to exponentially far away bits in the BBS bit-sequence is an operation which preserves pseudorandomness. This problem was discussed in [2, 8, 5, 13] and is still unresolved.

2.1.2. The GGM construction. Motivated in part by the easy-access problem, GGM [13] introduced the notion of pseudorandom functions and provided a generic construction based on any length-doubling pseudorandom generator. Note that a pseudorandom function may be viewed as an exponentially long bit-string which remains pseudorandom even after its bits are accessed in a direct manner. Thus, in some sense, GGM have bypassed the easy-access problem.[5]

When applied to an efficient pseudorandom generator based on factoring (e.g., the BBS generator), the GGM construction yields a length-preserving pseudorandom function which is as secure as factoring but requires as much as $O(n^2)$ modular multiplications per evaluation. On the other hand, a positive answer to the easy-access problem implies that the function

$$f^{BBS}_{N,g,r}(i) = G^{BBS}_{N,r,n}(g^{2^{i \cdot n}}) \tag{3}$$

is a length-preserving pseudorandom function which is at least as secure as factoring and requires only $O(n)$ modular multiplications per evaluation. Thus, in some sense, the question of whether it is possible to construct such efficient pseudorandom functions (which require as much modular multiplications as $f^{BBS}_{N,g,r}$) based on factoring remained open.

2.1.3. The NR constructions. About a decade after the GGM paper appeared, NR [19] suggested a parallel construction of pseudorandom functions. The NR construction was obtained by introducing a new cryptographic primitive, the *pseudorandom synthesizer*. By applying their method to specific constructions of pseudorandom synthesizers, they were later able to present efficient pseudorandom functions based on standard number-theoretic assumptions [20]. These constructions are considerably more efficient than the constructions which would have been obtained by applying the generic GGM construction to specific pseudorandom generators that are based on the same assumptions.

The DDH construction. The DDH construction is a construction of length-preserving pseudorandom functions as secure as the DDH problem requiring roughly

[5] What GGM have actually demonstrated is how to construct exponentially long, easily accessible, pseudorandom strings based on *any* one-way function (following [16]). However, this does not imply that the *specific* BBS bit-sequence remains pseudorandom given direct access to exponentially many of its bits.

$2n$ modular multiplications per evaluation [20]. This already matches the efficiency offered by $f^{BBS}_{N,g,r}$ and, to the best of our knowledge, is the most efficient construction of pseudorandom functions to date (based on standard intractability assumptions).

The factoring construction. The factoring construction is a construction of pseudorandom functions at least as secure as factoring requiring roughly $2n$ modular multiplications per evaluation [20]. (Their proof of security utilizes Biham, Boneh, and Reingold's [3] result that breaking the generalized Diffie–Hellman assumption over composites implies an efficient algorithm for factoring.)

For every $n \in \mathbb{N}$, a key of a function in the NR pseudorandom function ensemble, F_n, is a tuple $(N, \vec{a}, g, r)$, where N is an n-bit Blum-integer, r is an n-bit string, g is a quadratic-residue in $\mathbb{Z}^*_N$, and $\vec{a} = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$ is a sequence of $2n$ elements in $\{1, \ldots, N\}$. For any n-bit input $x = x_1 x_2 \ldots x_n$, the NR function (with a single bit of output) is defined as

$$f_{N,\vec{a},g,r}(x) = \mathcal{B}_r(g^{\Pi_{i=1}^n a_{i,x_i}}). \tag{4}$$

The NR construction gives a pseudorandom function which *seems* to be as efficient as $f^{BBS}_{N,g,r}$. Note however, that the NR function has only one bit of output, whereas $f^{BBS}_{N,g,r}$ has linear output length. While this may be sufficient for some applications, in most scenarios it is not. The goal of our work is to match the result one would have obtained by proving that $f^{BBS}_{N,g,r}$ are indeed pseudorandom functions. That is, we provide a new construction of pseudorandom functions that (1) are at least as secure as factoring Blum-integers, (2) have linear output length, and (3) require only $O(n)$ modular multiplications per evaluation.

2.2. Our construction. The NR pseudorandom function, $f_{N,\vec{a},g,r}$, is obtained by extracting the $\mathcal{B}_r(\cdot)$ predicate from the value of the function

$$h_{N,\vec{a},g}(x) = g^{\Pi_{i=1}^n a_{i,x_i}}. \tag{5}$$

It turns out that, even though it is not pseudorandom in itself, the function $h_{N,\vec{a},g}$ is unpredictable in some weak sense. Assuming the intractability of factoring Blum-integers, NR have shown [20, 21] that $h_{N,\vec{a},g}$ is unpredictable against an *adaptive* sample and a *random* challenge. That is, for a random $x \in \{0,1\}^n$, no polynomial-time adversary is able to predict the value of $h(x)$ after adaptively querying the value of $h(y)$ for polynomially many $y \neq x$ of his choice.

The main idea behind our construction is using the value of $h_{N,\vec{a},g}$ as a seed to the BBS pseudorandom generator. At first glance, it is not clear why this method should work *at all*. Indeed, applying a pseudorandom generator to an "unpredictable" value does *not* necessarily yield a pseudorandom function (see section 5.1 for a more detailed discussion on the subject). The reason for which our construction *does* work lies in the *specific* number theoretic features which the function $h_{N,\vec{a},g}$ and the BBS generator have in common.

This enables us to expand the output length of the NR function to polynomially many bits while paying a "reasonable" overhead in the complexity of the evaluation (i.e., one modular multiplication for each additional output bit). Specifically, let $N, \vec{a}, g$, and r be defined as in the NR function; the function we propose is defined as

$$f_{N,\vec{a},g,r}(x) = G^{BBS}_{N,r,\ell}(g^{\Pi_{i=1}^n a_{i,x_i}}). \tag{6}$$

Even though this does not solve the particular easy-access problem, it *does* match the efficiency one would have obtained by proving that $f^{BBS}_{N,g,r}$ are indeed pseudorandom functions (as well as the efficiency of the DDH-based pseudorandom functions by

NR [20]). By taking $\ell(n) = n$, we obtain a length-preserving pseudorandom function which is at least as secure as factoring, has linear output length, and requires only $3n$ modular multiplications per evaluation. This already matches (up to a constant factor) the efficiency of the best-known factoring-based pseudorandom *generators* (which also require $O(n)$ multiplications per evaluation) and certainly improves the efficiency of the GGM pseudorandom functions which use BBS as a building block.

3. Preliminaries. For the sake of completeness, we present the formal definition of pseudorandom functions. Our exposition follows the ones appearing in [11, 12, 19].

3.1. Pseudorandom functions—definition. Pseudorandom functions were defined by GGM [13]. Loosely speaking, these are efficient distributions of functions that cannot be efficiently distinguished from the uniform distribution. That is, an efficient algorithm that gets a function as a black-box cannot tell (with nonnegligible advantage) from which of the distributions it was sampled.[6] To formalize the notion of pseudorandom functions, we will need to consider ensembles of functions.

DEFINITION 3.1. *Let ℓ_d and ℓ_r be any two integer functions. An $I^{\ell_d} \to I^{\ell_r}$* function ensemble *is a sequence $F = \{F_n\}_{n\in\mathbb{N}}$ of random variables such that the random variable F_n assumes values in the set of $I^{\ell_d(n)} \to I^{\ell_r(n)}$ functions. The* uniform $I^{\ell_d} \to I^{\ell_r}$ function ensemble*, $R = \{R_n\}_{n\in\mathbb{N}}$, has R_n uniformly distributed over the set of $I^{\ell_d(n)} \to I^{\ell_r(n)}$ functions.*

An explicit description of a function $f : I^{\ell_d} \to I^{\ell_r}$ requires as many as $2^{\ell_r 2^{\ell_d}}$ bits. This suggests an alternative view of pseudorandom functions: These are distributions of exponentially long bit-sequences that cannot be distinguished from random by an efficient algorithm which has direct access to the sequence. To be of practical value, however, we require that pseudorandom functions can be efficiently sampled and computed. This property is not satisfied by every function ensemble (e.g., the uniform function ensemble: it contains $2^{\ell_r 2^{\ell_d}}$ functions whose mere representation requires as many as $\ell_r 2^{\ell_d}$ bits); we therefore restrict ourselves to efficiently computable function ensembles.

DEFINITION 3.2. *A function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is* efficiently computable *if there exist probabilistic polynomial-time algorithms, $\mathcal{I}$ and $\mathcal{V}$, and a mapping from strings to functions, ϕ such that $\phi(\mathcal{I}(1^n))$ and F_n are identically distributed and $\mathcal{V}(i, x) = (\phi(i))(x)$.*

We denote by f_i the function assigned to i (i.e., $f_i \stackrel{\text{def}}{=} \phi(i)$). We refer to i as the key of f_i and to $\mathcal{I}$ as the key-generating algorithm of F.

In particular, functions in efficiently computable function ensembles have relatively succinct representation (i.e., of polynomial rather than exponential length). As a consequence, these ensembles may have only exponentially many functions (out of double-exponentially many possible functions).

The distinguisher, in our setting, is defined to be an oracle machine that can make queries to a function (which is either sampled from the pseudorandom function ensemble[7] or from the uniform function ensemble[8]). We assume that on input 1^n

[6] For a detailed exposition on pseudorandom functions and their applications, we refer the reader to [24].

[7] We stress that, in the case that the function is sampled from the pseudorandom function ensemble, the distinguisher is *not* given the representation of the function f_i (i.e., the key i).

[8] As we have mentioned, it is not clear even how to efficiently represent a uniformly distributed function (as the representation it is too large to store). Still, one may simulate such a function by answering given queries with independently and uniformly chosen answers (while memorizing previous answers for the sake of consistency).

the oracle machine makes only n-bit queries. For any probabilistic oracle machine, $\mathcal{M}$, and any $I^n \to I^{\ell(n)}$ function, O, we denote by $\mathcal{M}^O(1^n)$ the distribution of $\mathcal{M}$'s output on input 1^n and with access to O.

DEFINITION 3.3. *An efficiently computable $I^n \to I^{\ell(n)}$ function ensemble, $F = \{F_n\}_{n\in\mathbb{N}}$, is* pseudorandom *if, for every probabilistic polynomial-time oracle machine $\mathcal{M}$, every polynomial $p(\cdot)$, and all sufficiently large n's,*

$$\left|\Pr\left[\mathcal{M}^{F_n}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{R_{n,\ell}}(1^n) = 1\right]\right| < \frac{1}{p(n)},$$

where $R = \{R_{n,\ell}\}_{n\in\mathbb{N}}$ is the uniform $I^n \to I^{\ell(n)}$ function ensemble.

The term "pseudorandom functions" is hereafter used as an abbreviation for "efficiently computable pseudorandom function ensemble."

3.2. Notation.

- $\mathbb{N}$ denotes the set of all natural numbers.
- For any integer $k \in \mathbb{N}$, denote by $[k]$ the set of integers $\{0, 1, \ldots, k-1\}$.
- For any integer $N \in \mathbb{N}$, the multiplicative group modulo N is denoted by $\mathbb{Z}_N^*$.
- The order of $\mathbb{Z}_N^*$ (i.e., the number of $x \in [N]$ such that $\gcd(x, N) = 1$) is denoted by $\varphi(N)$.
- I^n denotes the set of all n-bit strings, $\{0,1\}^n$.
- U_n denotes the random variable uniformly distributed over I^n.
- Let x and y be any two-bit strings; then x, y denote the string x concatenated with y.

4. The main result. We are now ready to present the main result of our work, an efficient construction of pseudorandom functions whose security is based on the intractability of factoring. Specifically, we are able to show how any procedure which is able to distinguish our functions from randomly chosen ones can be turned into an algorithm which factors a nonnegligible fraction of Blum-integers. We begin by formalizing the assumption that factoring Blum-integers is hard.

4.1. The factoring assumption. In order to keep our result general, we let N be generated by *some* polynomial-time algorithm FIG (where FIG stands for factoring-instance-generator).

DEFINITION 4.1. *A factoring-instance-generator, FIG, is a probabilistic polynomial-time algorithm such that, on input 1^n, its output, $N = P \cdot Q$, is distributed over n-bit integers, where P and Q are two primes that satisfy $P = Q = 3 \bmod 4$. (Such N is known as a Blum-integer.)*

A natural example for a factoring-instance-generator would be to let $FIG(1^n)$ be uniformly distributed over *n-bit* Blum-integers.[9] However, other choices were previously considered (e.g., letting P and Q obey some "safety" conditions).[10] We now formalize the assumption that factoring Blum-integers is hard.

DEFINITION 4.2 (ϵ-factoring). *Let $\mathcal{A}$ be a probabilistic Turing machine, and let $\epsilon = \epsilon(n)$ be a real-valued function. $\mathcal{A}$ ϵ-factors if, for infinitely many n's,*

$$\Pr[\mathcal{A}(P \cdot Q) \in \{P, Q\}] > \epsilon(n),$$

[9] We note that *n-bit* Blum-integers are a nonnegligible fraction of all *n-bit* integers and that it is easy to sample a uniformly distributed *n-bit* Blum-integer.

[10] For example, it is often required that P and Q be of equal size and that P, Q be of the form $P = 2P' + 1$ and $Q = 2Q' + 1$ for some primes P', Q'.

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$.

In spite of the extensive research directed toward the construction of efficient integer factoring algorithms, the best algorithms currently known for factoring an integer N have (heuristic) running-time $L(N) \stackrel{\text{def}}{=} e^{1.92(\log N)^{1/3}(\log\log N)^{2/3}}$ (cf. [22]). This (together with the fact that Blum-integers are a nonnegligible fraction of all n-bit integers) leads us to the following assumption.

ASSUMPTION 4.1 (factoring integers generated by *FIG*). *Let $\mathcal{A}$ be any probabilistic polynomial-time machine. There is no positive constant α such that $\mathcal{A}$ $\frac{1}{n^\alpha}$-factors.*

All exponentiations in the rest of this section are in $\mathbb{Z}_N^*$. To simplify the notation, we omit the expression "mod N" from now on.

4.2. The construction.

CONSTRUCTION 4.1. *We define a function in the ensemble $F = \{F_n\}_{n\in\mathbb{N}}$. For every $n \in \mathbb{N}$, a key of a function in F_n is a tuple $(N, \vec{a}, g, r)$, where N is an n-bit Blum-integer, g is a quadratic-residue in $\mathbb{Z}_N^*$, $\vec{a} = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$ is a sequence of $2n$ elements in $[N]$, and r is an n-bit string. For any n-bit input $x = x_1 x_2 \ldots x_n$ and for every integer-valued function $\ell = \ell(n)$, the function $f_{N,\vec{a},g,r} : I^n \to I^{\ell(n)}$ is defined by*

$$f_{N,\vec{a},g,r}(x) \stackrel{\text{def}}{=} \mathcal{B}_r(g^{\Pi_{i=1}^n a_{i,x_i}}), \mathcal{B}_r(g^{2\Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^{\ell-1}\Pi_{i=1}^n a_{i,x_i}}),$$

where $\mathcal{B}_r(m)$ denotes the inner product, $\langle m, r\rangle \bmod 2$. The distribution of functions in F_n is induced by the following distribution on their keys: $\vec{a}$, g, and r are uniform in their range, and the distribution of N is $FIG(1^n)$.

REMARK 4.1. *Construction* 4.1 *employs a Blum-integer, N. In this we follow* [5] *and many other works. As discussed in section* 2.1.1*, this restriction implies that squaring is a permutation on the subgroup of quadratic-residues in $\mathbb{Z}_N^*$. Nevertheless, as was pointed out to us by Shai Halevi and an anonymous referee, it is rather easy to extend our construction (as well as many previous results) in order to allow an arbitrary moduli $N = P \cdot Q$ (that is assumed to be hard to factor) instead of a Blum-integer. The main observation that is needed is that, for any such N, squaring is a permutation on the subgroup of 2^n-powers in $\mathbb{Z}_N^*$. See* [15, 25] *for additional details on avoiding the restriction to a Blum-integer in related contexts.*

An additional variant of the construction is discussed in section 6*. There, we discuss how to replace the Goldreich–Levin hard-core bit with the $\mathcal{LSB}$ predicate.*

4.3. Efficiency of the construction. Consider a function $f_{N,\vec{a},g,r} \in F_n$ as in Construction 4.1. Computing the value of this function at any given point, x, involves one multiple product $y = \prod_{i=1}^n a_{i,x_i}$ (which can be performed modulo $\varphi(N)$), one modular exponentiation, $z = g^y \bmod N$, and $\ell(n) - 2$ successive modular squarings $z^2, \ldots, z^{2^k}, \ldots, z^{2^{\ell-1}}$ (which require less than one modular multiplication each). The value of the function is finally obtained by computing $\mathcal{B}_r(z), \mathcal{B}_r(z^2), \ldots, \mathcal{B}_r(z^{2^{\ell-1}})$ (which is a cheap operation compared to modular multiplication). As discussed in [20], it is possible to use preprocessing in order to get improved efficiency.[11] This gives us a pseudorandom function which can be evaluated roughly at the cost of $2n+\ell(n)$ modular multiplications. (A modular exponentiation is counted as n modular multiplications.)

[11] The most obvious preprocessing would be to compute the values g^{2^i} (for every positive integer i up to the length of $(P-1)\cdot(Q-1)$). See [20] for additional preprocessing techniques which further improve the efficiency.

An attractive feature of our construction is that, for each input, we can have a *variable* length output; i.e., if for some x's one needs more bits in the output of $f(x)$, then the natural way of simply taking more bits of the form $\mathcal{B}_r(z^{2^j})$ works. While it is possible to get this feature generically by combining a pseudorandom function and a generator, here we get it "for free."

5. Proof of security. The following theorem establishes the security of Construction 4.1.

THEOREM 5.1. *If the factoring assumption holds (Assumption* 4.1*), then* $F = \{F_n\}_{n\in\mathbb{N}}$ *(as in Construction* 4.1*) is an efficiently computable pseudorandom function ensemble.*

REMARK 5.1. *The proof of Theorem* 5.1 *yields a more quantitative version as well: Assume that there exists a probabilistic polynomial-time oracle machine with running-time* $t(n)$ *that distinguishes* $f_{N,\vec{a},g,r}$ *from* $\rho_{n,\ell}$ *with advantage* $\epsilon(n)$ *(where* $\rho_{n,\ell}$ *is uniformly distributed in the set of functions with domain* $\{0,1\}^n$ *and range* $\{0,1\}^{\ell(n)}$*). Let* $q = q(n)$ *be a bound on the number of queries made this machine. Then there exists a probabilistic polynomial-time algorithm with running-time* $poly(\frac{1}{\epsilon(n)}, t(n), \ell(n))$ *that* ϵ''*-factors for* $\epsilon''(n)$ *which equals* $\Omega(\frac{\epsilon(n)^2}{q(n)^2\cdot\ell(n)^2})$.

5.1. On the methodology.

5.1.1. A simple approach does not work. In order to prove Theorem 5.1, one might be tempted to use the following approach: Recall the definition of the function $h_{N,\vec{a},g}(x) = g^{\Pi_{i=1}^n a_{i,x_i}}$ in (5). As mentioned above, it was shown in [20, 21] that $h_{N,\vec{a},g}$ is unpredictable against an *adaptive* sample and a *random* challenge. In light of this, Construction 4.1 can be viewed as based on the following methodology:

1. Take an "unpredictable" function $h_s : \{0,1\}^n \to \{0,1\}^n$.
2. Take a pseudorandom generator $G : \{0,1\}^n \to \{0,1\}^\ell$.
3. Obtain a pseudorandom function $f : \{0,1\}^n \to \{0,1\}^\ell$ by setting $f_s(x) = G(h_s(x))$.

Unfortunately, this method does not work in general. As will be demonstrated next, there exist an "unpredictable" function and a pseudorandom generator such that their composition is *not* a pseudorandom function.

5.1.2. The counterexample. Consider the following (unnatural) counterexample:

1. (a) Let $h'_s : \{0,1\}^n \to \{0,1\}^{\frac{n}{2}}$ be an unpredictable function.
   (b) For $y \in \{0,1\}^{\frac{n}{2}}$, define $h_{s,y} : \{0,1\}^n \to \{0,1\}^n$ as $h_{s,y}(x) = (h'_s(x), y)$.
   (c) Clearly $h_{s,y}$ is unpredictable.
2. (a) Let $G' : \{0,1\}^{\frac{n}{2}} \to \{0,1\}^\ell$ be a pseudorandom generator.
   (b) For $z, y \in \{0,1\}^{\frac{n}{2}}$, define $G : \{0,1\}^n \to \{0,1\}^\ell$ as $G(z,y) = G'(y)$.
   (c) Clearly G is pseudorandom.
3. However, the function $f : \{0,1\}^n \to \{0,1\}^\ell$ obtained by setting $f_{s,y}(x) = G(h_{s,y}(x))$ is always equal to $G'(y)$ (regardless of the value of x). Obviously, $f_{s,y}$ cannot be pseudorandom.

It seems that the reason our construction *does* work lies in the *specific* number theoretic features which the function $g^{\Pi_{i=1}^n a_{i,x_i}}$ and the BBS generator have in common. Since we do not know what precisely are the features of a function h_s and of a pseudorandom generator G that are needed in order to obtain a pseudorandom function (using the above methodology), we are forced to provide a direct proof for our construction. As most proofs of pseudorandomness do, we will use a *hybrid argument*,

i.e., mixing a truly random and pseudorandom distribution. The type of hybrid we apply is the *reverse hybrid*, where first the random part is used and only then the pseudorandom one. This is an instance of the *principle of deferred decision* (see [18]): Do not commit to any value in the pseudorandom part of the distribution until you have to.

5.2. Proof of Theorem 5.1.

Proof. Let $F = \{F_n\}_{n\in\mathbb{N}}$ be as in Construction 4.1. It is clear that F is efficiently computable. Assume that F is not pseudorandom; then there exist a probabilistic polynomial-time oracle machine $\mathcal{M}$ and a nonnegligible real-valued function $\epsilon = \epsilon(n)$ such that, for infinitely many n's,

$$\left|\Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n) = 1\right] - \Pr\left[\mathcal{M}^{\rho_{n,\ell}}(1^n) = 1\right]\right| > \epsilon(n), \tag{7}$$

where, in the first probability, $f_{N,\vec{a},g,r}$ is distributed according to F_n and, in the second probability, $\rho_{n,\ell}$ is distributed according to $R_{n,\ell}$ (the uniform ensemble of functions with domain $\{0,1\}^n$ and range $\{0,1\}^{\ell(n)}$).

A hybrid black-box. Inequality (7) tells us that there is a nonnegligible difference between the output behavior of $\mathcal{M}$ in the case it is given access to a black-box which answers according to $f_{N,\vec{a},g,r}$ and in the case it is given access to a black-box which answers according to $\rho_{n,\ell}$. However, $\mathcal{M}$'s response is still a well defined random variable even when its queries are answered according to some other distribution. This means that we are allowed to invoke $\mathcal{M}$ and answer its queries in whatever way we find suitable for our purposes. The way we choose to do it is by defining a *hybrid black-box*.[12] Informally, this is a black-box which starts by answering $\mathcal{M}$'s queries according to $\rho_{n,\ell}$ and then switches mode to continue and answer according to $f_{N,\vec{a},g,r}$.

Let $t = t(n)$ be a polynomial that bounds the running-time of $\mathcal{M}$; assume without loss of generality that $\mathcal{M}$ always makes exactly t queries. Since each single answer given to these queries is ℓ-bit long, we have that the total number of bits which $\mathcal{M}$ gets as answers during its execution is precisely $t \cdot \ell$. Each one of these bits corresponds to a location for which one of the hybrid black-box distributions will switch from answering according to $\rho_{n,\ell}$ to answering according to $f_{N,\vec{a},g,r}$. (Note that this may also happen in the middle of an answer.)

Definition 5.1 (hybrid black-box). *Let J be an element in $[t \cdot \ell + 1]$ written as $J = I \cdot \ell + k$ (where $0 \le I \le t$ and $0 \le k < \ell$). The Jth hybrid black-box, $H^J_{N,\vec{a},g,r}$, is defined by the answers it gives to $\mathcal{M}$'s queries. The first I queries are answered according to $\rho_{n,\ell}$ (i.e., at random), and the answer to the $(I+1)$st query up to the kth bit-location is according to $\rho_{n,\ell}$, and from then on, according to $f_{N,\vec{a},g,r}$. All remaining queries are answered according to $f_{N,\vec{a},g,r}$.*

Notice that $H^0_{N,\vec{a},g,r}$ is a black-box which always answers according to $f_{N,\vec{a},g,r}$, whereas $H^{t\cdot\ell}_{N,\vec{a},g,r}$ always answers according to $\rho_{n,\ell}$. By inequality (7), we have that $\mathcal{M}$ distinguishes between $f_{N,\vec{a},g,r}$ and $\rho_{n,\ell}$ with advantage $\epsilon(n)$. Therefore, if we pick J at random, the expected advantage that $\mathcal{M}$ has in distinguishing between $H^J_{N,\vec{a},g,r}$ and $H^{J+1}_{N,\vec{a},g,r}$ is at least $\epsilon'(n) = \frac{\epsilon(n)}{t(n)\cdot\ell(n)}$. An example of a hybrid black-box is depicted in Figure 1.

[12] This is just a methodological modification of the standard hybrid technique (see [11] for details on the hybrid technique).

	1	2	$\cdots$	I	$I+1$	$I+2$	$\ldots$	t
1	r	r	$\cdots$	r	r			
2	r	r		r	r			
.	.	.	$\cdots$	.	.			
k	r	r		r	r		F_n	
$k+1$	r	r		r				
.	.	.	$\cdots$	.				
ℓ	r	r	$\cdots$	r				

FIG. 1. *Illustrates the Jth hybrid black-box, $H^J_{N,\vec{a},g,r}$ (where $J = I \cdot \ell + k$). Columns correspond to queries given to the black-box, and rows correspond to individual bits in the relevant answers.*

5.2.1. Simplified proof. We start by giving a simplified version of the proof under the assumption that $\mathcal{M}$ decides membership in F_n with advantage $\epsilon(n)$ for *any* sequence $\vec{a}$ of $2n$ elements in $[N]$. (Recall that $\vec{a}$ equals $(a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$.) We then proceed and show how to modify the proof so it will work for a randomly chosen $\vec{a}$.

Roughly speaking, we show how, given a distinguisher for our pseudorandom functions, we can construct an algorithm that on input $(v^2 \bmod N, N, r)$ predicts the value of $\mathcal{B}_r(u)$, where u is the unique quadratic-residue in $\mathbb{Z}^*_N$ which satisfies $u^2 = v^2 \bmod N$. Using the Goldreich–Levin reconstruction algorithm, we are then able to retrieve u (see [11] for details). This means that we can extract square-roots in $\mathbb{Z}^*_N$ and consequently factor Blum-integers (as described in [23]).

THEOREM 5.2 (Goldreich–Levin [14]). *Let $z, r \in \{0,1\}^n$. Given an oracle that, on input r, predicts the value of $\mathcal{B}_r(z)$ with advantage $\epsilon(n)$ (over the choice of r) in time $t(n)$, there exists a probabilistic polynomial-time algorithm with running-time $O(\frac{n^2 \cdot t(n)}{\epsilon(n)^2})$ that retrieves z with probability at least $\Omega(\epsilon(n))$.*

The following lemma can be viewed as the heart of the simplified part of the proof. It describes how, given a distinguisher for our pseudorandom functions, it is possible to construct an algorithm, $\mathcal{D}$, which can be used by the reconstruction algorithm as an oracle for the value of $\mathcal{B}_r(u)$.

LEMMA 5.3. *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$ satisfying inequality* (7) *for* any *choice of $\vec{a}$. Then there exists a probabilistic polynomial-time algorithm $\mathcal{D}$ such that, for infinitely many n's,*

$$\left|\Pr\left[\mathcal{D}(v^2, N, r, \mathcal{B}_r(u)) = 1\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1\right]\right| > \epsilon'(n),$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}^*_N$, r is a random n-bit string, $b \in_R \{0,1\}$, and u is the unique quadratic-residue in $\mathbb{Z}^*_N$ which satisfies $u^2 = v^2 \bmod N$.

Using an averaging argument, it can be shown that, on at least an $\frac{\epsilon'(n)}{2}$ fraction of the choices of N and v^2, algorithm $\mathcal{D}$ distinguishes the value of $\mathcal{B}_r(u)$ from random with advantage $\frac{\epsilon'(n)}{2}$ (over the choice of r). In particular, $\mathcal{D}$ can be used in order to predict $\mathcal{B}_r(u)$ with probability $\frac{1}{2} + \frac{\epsilon'(n)}{4}$. By Theorem 5.2, we know that we can use $\mathcal{D}$ in order to construct a probabilistic polynomial-time algorithm that retrieves $u \bmod N$ with probability $\Omega(\epsilon'(n))$. We now have that $u^2 = v^2 \bmod N$ and $\Pr[u \neq \pm v] = 1/2$. This implies (cf. [23]) that $\Pr[gcd(u - v, N) \in \{P, Q\}] = 1/2$, which enables us to construct an algorithm that $\Omega\left(\epsilon'(n)^2\right)$-factors, in contradiction to Assumption 4.1.

Description of $\mathcal{D}$. Let the input of $\mathcal{D}$ be (v^2, N, r, α), where the distribution of N, v, and r is as in Lemma 5.3. Let u be the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $u^2 = v^2 \bmod N$. On this input, $\mathcal{D}$ first picks $J = I \cdot \ell + k$ at random in $[t \cdot \ell]$. Then $\mathcal{D}$ invokes $\mathcal{M}$ and answers its queries in a way that simulates $H^J_{N,\vec{a},g,r}$ (for some value of $\vec{a}$ and g) if α equals $\mathcal{B}_r(u)$ and simulates $H^{J+1}_{N,\vec{a},g,r}$ if α is a random bit. $\mathcal{D}$ will now be able to utilize the expected advantage that $\mathcal{M}$ has in distinguishing between $H^J_{N,\vec{a},g,r}$ and $H^{J+1}_{N,\vec{a},g,r}$ in order to guess the actual distribution of α. Specifically, $\mathcal{D}$ will answer $\mathcal{M}$'s queries in the following way:

1. Answer the first I queries according to $\rho_{n,\ell}$.
2. Answer the $(I+1)$st query with $b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(u^2), \ldots, \mathcal{B}_r(u^{2^{\ell-k-1}})$ (where $b_0 b_1 \ldots b_{k-1}$ denotes a random k-bit string).
3. Answer the remaining queries consistently according to $f_{N,\vec{a},g,r}$.

The challenge in constructing $\mathcal{D}$ is to assign values to $\vec{a}$ and g such that the above answers will be distributed according to the correct hybrid black-box distribution and will be efficiently computable by $\mathcal{D}$.

Defining $\vec{a}$ and g. The way we define the value of g depends on the choice of J, whereas the values assigned to $\vec{a}$ will depend on the $(I+1)$st query made by $\mathcal{M}$. We require that, if x is the $(I+1)$st query, then the value of the $(k+1)$st bit of $f_{N,\vec{a},g,r}(x)$ (which is answered with α by $\mathcal{D}$) will be equal to $\mathcal{B}_r(u)$. In addition, we require that $\mathcal{D}$ will be able to efficiently compute the answers to all the subsequent queries made by $\mathcal{M}$ (starting from the $(k+2)$nd bit-location in the answer to the $(I+1)$st query).

The definition of g. $\mathcal{D}$ computes $s = \ell \cdot n - k$ and sets $g = v^{2^s} \bmod N$.

CLAIM 5.1. *Let γ be the order of g in $\mathbb{Z}_N^*$; then γ is odd.*

Proof. It will be sufficient to show that the size of QR_N is odd. This implies that all quadratic-residues in $\mathbb{Z}_N^*$ (and, in particular, g) have odd order. Since N is a Blum-integer, for every quadratic-residue in $\mathbb{Z}_N^*$, exactly one of its four square-roots resides in QR_N. This means that $|QR_N| = \frac{|\mathbb{Z}_N^*|}{4} = \frac{(P-1)\cdot(Q-1)}{4}$. Since $P = Q = 3 \bmod 4$, we have that $\frac{P-1}{2}$ and $\frac{Q-1}{2}$ are odd, which implies that $\frac{(P-1)\cdot(Q-1)}{4}$ is also odd, and the claim follows. □

CLAIM 5.2. *For every $0 < i < s$, $g^{2^{-i} \bmod \gamma} = v^{2^{s-i}} \bmod N$.*[13]

Proof. By Claim 5.1, we have that γ is odd. This implies that $2 \in \mathbb{Z}_\gamma^*$ and, therefore, $2^{-1} \bmod \gamma$ exists (and is simply $\frac{\gamma+1}{2}$). For simplicity of exposition, let us denote by $g^{2^{-1}}$ the value $g^{2^{-1} \bmod \gamma} \bmod N$. We now have that, whenever 2^{-1} appears in the exponent, it denotes the value $2^{-1} \bmod \gamma = \frac{\gamma+1}{2}$. Similarly, 2^{-i} in the exponent denotes the value $2^{-i} \bmod \gamma = (\frac{\gamma+1}{2})^i \bmod \gamma$. Therefore, for every i, the value $g^{2^{-i}} \bmod N$ is a quadratic-residue (since g is a quadratic-residue). Take $i = 1$; we now have that both $g^{2^{-1}}$ and $v^{2^{s-1}}$ are square-roots of g and they are both quadratic-residues. Since squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$ (for any Blum-integer N), we must have that $g^{2^{-1}}$ and $v^{2^{s-1}}$ are equal. By induction on $0 < i < s$, we get that $g^{2^{-i}} = v^{2^{s-i}} \bmod N$. □

COROLLARY 5.1. *Let $u = g^{2^{-s}} \bmod N$; then $u^2 = v^2 \bmod N$.*

The definition of $\vec{a}$. Since the first I queries of $\mathcal{M}$ are answered randomly, we can defer the assignment to the values of $\vec{a}$ until $\mathcal{D}$ is given the $(I+1)$st query, $x = x_1 x_2 \ldots x_n$. It is then possible to define $\vec{a}$ so that the value of the $(k+1)$st

[13] Note that γ is not known to $\mathcal{D}$. However, as long as $s - i > 0$, it is possible for $\mathcal{D}$ to compute $v^{2^{s-i}} \bmod N$.

bit of $f_{N,\vec{a},g,r}(x)$ (namely, $\mathcal{B}_r(g^{2^k\Pi_{i=1}^n a_{i,x_i}})$) will be equal to $\mathcal{B}_r(u)$.[14] Let $\vec{a}$ be the vector $(a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$, where, for all i, $a_{i,x_i} = 2^{-\ell} \bmod \gamma$ and $a_{i,\overline{x_i}}$ is uniformly distributed in $[N]$.

CLAIM 5.3. *Let $\vec{a}, g$, and u be defined as above; then*

$$\mathcal{B}_r(g^{2^k\Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(u).$$

Proof. By the above notation,

$$g^{2^k\Pi_{i=1}^n a_{i,x_i}} = g^{2^{k-\ell n}} = g^{2^{-s}} = u \bmod N,$$

and the claim follows. □

The running-time of $\mathcal{D}$. We now show that $\mathcal{D}$ is indeed able to complete steps (2) and (3) (i.e., answer all the remaining queries of $\mathcal{M}$ starting from the $(k+2)$nd bit-location in the answer to the $(I+1)$st query in a way which is consistent with the definition of $\vec{a}$ and g). The key point is that $\mathcal{D}$ can achieve this task even though it does not actually know the values of a_{i,x_i}.

CLAIM 5.4. *Algorithm $\mathcal{D}$ is able to efficiently complete the answer to the $(I+1)$st query (step 2).*

Proof. Since $v^2 = u^2 \bmod N$ and since v^2 and r are given to $\mathcal{D}$ in the input, $\mathcal{D}$ is able to complete the answer to the $(I+1)$st query and answer $\mathcal{M}$ with

$$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(v^2), \ldots, \mathcal{B}_r(v^{2^{\ell-k-1}})$$

as required. □

CLAIM 5.5. *For* any *query $y \neq x$, $\mathcal{D}$ is able to efficiently compute the value of $f_{N,\vec{a},g,r}(y)$ (step 3).*

Proof. It will be sufficient to show that, for any query $y = y_1 y_2 \ldots y_n \neq x$, $\mathcal{D}$ is able to compute the value $g^{\Pi_{i=1}^n a_{i,y_i}}$, and thus it is always able to answer the query with the value of $f_{N,\vec{a},g,r}(y)$. Now

$$\begin{aligned} g^{\Pi_{i=1}^n a_{i,y_i}} &= g^{(\Pi_{\{y_i=x_i\}} a_{i,y_i})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \\ &= g^{(2^{-\ell j})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \\ &= v^{(2^{s-\ell j})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \bmod N, \end{aligned}$$

where j is the number of locations for which y_i equals x_i. Since $j < n$ (remember that $y \neq x$) and since $k < \ell$, we always have that $s - \ell j = (n-j)\ell - k$ is at least 1 over the integers. As we have already stated, $\mathcal{D}$ knows the value of v^2; therefore, by performing the appropriate exponentiations, it is always able to compute $g^{\Pi_{i=1}^n a_{i,y_i}} = v^{(2^{s-\ell j})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})}$ as required. (Remember that $\mathcal{D}$ knows the value of a_{i,y_i} for all $y_i \neq x_i$.) □

We are finally ready to establish Lemma 5.3. Recall that, given some J, the value of α determines which of the possible hybrid black-boxes is simulated by $\mathcal{D}$. If α equals $\mathcal{B}_r(u)$, then $\mathcal{D}$ simulates $H^J_{N,\vec{a},g,r}$, whereas if α is a random bit, $\mathcal{D}$ simulates $H^{J+1}_{N,\vec{a},g,r}$. Since the expected advantage $\mathcal{M}$ has in distinguishing the above neighboring hybrid black-boxes is $\epsilon'(n)$ and since $\mathcal{D}$ picks J at random, we expect that $\mathcal{D}$ will be able to decide with advantage $\epsilon'(n)$ whether α is indeed the value of $\mathcal{B}_r(u)$.

[14] It is worth noticing that this would not have been possible in the case that $\mathcal{D}$ would have answered $\mathcal{M}$'s queries in a reverse order (i.e., by first answering according to $f_{N,\vec{a},g,r}$ and then switching mode to $\rho_{n,\ell}$).

5.2.2. Completing the proof. To complete the proof, we follow the same lines, along with an additional "randomization" of the values in $\vec{a}$, achieved by taking $a_{i,x_i} = \xi_i + 2^{-\ell} \bmod \gamma$. This causes the value of the $(k+1)$st bit in the answer to the $(I+1)$st query to change into $\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(g^{2^k \Pi_{i=1}^n (\xi_i + 2^{-\ell})}) = \mathcal{B}_r(u \cdot w)$, where w is an element in $\mathbb{Z}_N^*$ which is completely determined by the ξ_i's and by the value of v^2 (and is efficiently computable given the above values). Note that now algorithm $\mathcal{D}$ becomes an oracle for the value of $\mathcal{B}_r(u \cdot w)$ and will therefore be used in order to retrieve $u \cdot w$ (rather than u).

Jumping ahead, we note that letting $\mathcal{D}$ pick the random ξ_i's by itself would have caused the value of w (and therefore $u \cdot w$) to change each time $\mathcal{D}$ is invoked. This would not allow the reconstruction algorithm to retrieve $u \cdot w$ for any specific value of w. The solution to this problem will be to fix the values which determine w in advance and then use $\mathcal{D}$ (which now takes only r as input) as an oracle for $\mathcal{B}_r(u \cdot w)$.[15] For the time being, we ignore the above issue and let $\mathcal{D}$ pick the random ξ_i's by itself. We now give the (full) analogue of Lemma 5.3.

LEMMA 5.4. *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$ satisfying inequality* (7). *Then there exists a probabilistic polynomial-time algorithm $\mathcal{D}$ such that, for infinitely many n's,*

$$\left|\Pr\left[\mathcal{D}(v^2, N, r, \mathcal{B}_r(u \cdot w)) = 1\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1\right]\right| > \epsilon'(n) - n \cdot 2^{-O(n)},$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b \in_R \{0,1\}$, $u \cdot w$ is the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $(u \cdot w)^2 = v^2 \cdot w^2 \bmod N$, and w is a randomly chosen quadratic-residue in $\mathbb{Z}_N^*$ (which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses and is efficiently computable by $\mathcal{D}$).

Proof. On input (v^2, N, r, α), $\mathcal{D}$ is defined as follows:

1. (a) Sample $J = I \cdot \ell + k$ uniformly at random in $[t \cdot \ell]$.
   (b) Sample J random bits (needed in order to simulate the hybrid black-box).
   (c) Sample $\vec{\xi} = (\xi_1, \xi_2, \ldots, \xi_n, \xi_1', \xi_2', \ldots, \xi_n')$ by uniformly picking $2n$ elements in $[N]$.
2. Compute $s = \ell \cdot n - k$, and set $g = v^{2^s} \bmod N$.
3. Invoke $\mathcal{M}$ on input 1^n:
   (a) Answer each of its first I queries with a random string in $\{0,1\}^\ell$.
   (b) Let $x_1 x_2 \ldots x_n$ be $\mathcal{M}$'s $(I+1)$st query.
   For $1 \le i \le n$, denote by a_{i,x_i} the value $\xi_i + 2^{-\ell} \bmod \gamma$ and by $a_{i,\overline{x_i}}$ the value ξ_i'. Denote by $\vec{a}$ the sequence $(a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$. ($a_{i,x_i}$ is not actually known to $\mathcal{D}$.)
   Let $b_0 b_1 \ldots b_{k-1}$ be a random k-bit string; answer the $(I+1)$st query with the ℓ-bit string

$$b_0, b_1, \ldots, b_{k-1}, \alpha, \mathcal{B}_r(g^{2^{k+1} \Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{B}_r(g^{2^{\ell-1} \Pi_{i=1}^n a_{i,x_i}}).$$

   (c) Answer each remaining query of $\mathcal{M}$, y with the value $f_{N,\vec{a},g,r}(y)$.

[15] Here we use a special property of the $\mathcal{B}_r$ predicate, which enables the reconstruction of $u \cdot w$ by asking only queries which refer to $u \cdot w$ (i.e., $u \cdot w$ is *fixed* throughout the process, and only r changes from one query to another). As discussed in section 6, a similar property is satisfied by the $\mathcal{LSB}$-based reconstruction techniques by Alexi et al. [1, 10]. See section 6 for an analogous construction that uses the $\mathcal{LSB}$ predicate.

4. If $\mathcal{M}$ outputs 1, then output 1.
   If $\mathcal{M}$ outputs 0, then output 0.

Why does $\mathcal{D}$ predict the value of $\mathcal{B}_r(u \cdot w)$? To define u, we set $u = g^{2^{-s}} \bmod N$. As before (see Corollary 5.1), we have that $u^2 = v^2 \bmod N$. To define w, we set $w = v^{(\Sigma_{i=0}^{n-1}\beta_i 2^{\ell(n-i)})} \bmod N$, where β_i are the coefficients (over $\mathbb{Z}$) of the polynomial $p(x) = \prod_{i=1}^{n}(\xi_i + x) = x^n + \sum_{i=0}^{n-1}\beta_i x^i$. Note that the β_i's can be efficiently computed given the ξ_i's (either recursively or by interpolation). Given the values of the β_i's and of v^2, we are able to compute w. (Note that the exponent of v in the definition of w is always even.) Therefore, w is an efficiently computable quadratic-residue in $\mathbb{Z}_N^*$ which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses (i.e., the ξ_i's sampled in step 1c).

CLAIM 5.6. *Let $\vec{a}$, g, u, and w be defined as above; then $\mathcal{B}_r(g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{B}_r(u \cdot w)$.*

Proof. Using the above notation (and Claim 5.2), we have

$$\begin{aligned} g^{2^k \Pi_{i=1}^n a_{i,x_i}} &= g^{2^k \Pi_{i=1}^n (\xi_i + 2^{-\ell})} \\ &= g^{2^k p(2^{-\ell})} \\ &= g^{2^{k-\ell n} + \Sigma_{i=1}^{n-1}\beta_i 2^{-(\ell i - k)}} \\ &= g^{2^{-s}} \cdot v^{(\Sigma_{i=1}^{n-1}\beta_i 2^{s-(\ell i-k)})} \\ &= u \cdot v^{(\Sigma_{i=1}^{n-1}\beta_i 2^{\ell(n-i)})} \\ &= u \cdot w \bmod N, \end{aligned}$$

and the claim follows. □

This implies that the $(k+1)$st bit in the answer that the $f_{N,\vec{a},g,r}$ black-box is supposed to give to the $(I+1)$st query (and is answered with α instead) is equal to $\mathcal{B}_r(u \cdot w)$. As we have already seen, this fact can be used by $\mathcal{D}$ in order to decide whether α equals $\mathcal{B}_r(u \cdot w)$.

The running-time of $\mathcal{D}$. It is clear that steps 1, 2, and 3a can be carried out in time $poly(n, \ell(n))$. In order to prove that steps 3b and 3c can be carried out in time $poly(n, \ell(n)) \cdot t(n)$, we observe that, if, for some j in $[\ell]$, $\mathcal{D}$ is able to compute $\lambda = g^{2^j \Pi_{i=1}^n a_{i,x_i}} \bmod N$, then, by squaring and taking the inner product of the results with r, it is also able to compute any bit-sequence of the form $\mathcal{B}_r(\lambda), \mathcal{B}_r(\lambda^2), \ldots, \mathcal{B}_r(\lambda^{2^{\ell-j-1}})$.

CLAIM 5.7. *Algorithm $\mathcal{D}$ is able to efficiently complete the answer to the $(I+1)$st query (step* 3b*).*

Proof. By the above observation, it will be sufficient to show that $\mathcal{D}$ is able to efficiently compute the value of $g^{2^{k+1}\Pi_{i=1}^n a_{i,x_i}}$. To see that, notice that $g^{2^k \Pi_{i=1}^n a_{i,x_i}}$ equals $u \cdot w$ (see Claim 5.6). This implies that $g^{2^{k+1}\Pi_{i=1}^n a_{i,x_i}} = u^2 \cdot w^2 = v^2 \cdot w^2 \bmod N$. Since both the values of v^2 and w^2 are known to $\mathcal{D}$, it is possible for it to efficiently answer the $(I+1)$st query (as described in step 3b). □

CLAIM 5.8. *For* any *query $y \neq x$, $\mathcal{D}$ is able to efficiently compute the value of $f_{N,\vec{a},g,r}(y)$ (step* 3c*).*

Proof. By the above observation, it will be sufficient to show that $\mathcal{D}$ is able to compute the value $g^{\Pi_{i=1}^n a_{i,y_i}}$ (and, consequently, it will be able to compute the value of $f_{N,\vec{a},g,r}(y)$). Given a query $y = y_1 y_2 \ldots y_n \neq x$, $\mathcal{D}$ starts by computing (in time $poly(n)$) the coefficients, $\delta_i \in \mathbb{Z}$, of the polynomial $q(x) = \prod_{\{y_i = x_i\}}(\xi_i + x) =$

$\sum_{i=0}^{j} \delta_i x^i$ (where j is the number of locations for which y_i equals x_i). Under this notation, $\prod_{\{y_i=x_i\}} a_{i,y_i}$ equals $q(2^{-\ell})$, we then have

$$\begin{aligned} g^{\Pi_{i=1}^{n} a_{i,y_i}} &= g^{(\Pi_{\{y_i=x_i\}} a_{i,y_i})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \\ &= g^{(\Sigma_{i=0}^{j} \delta_i 2^{-\ell i})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \\ &= \prod_{i=0}^{j} g^{(\delta_i 2^{-\ell i})(\Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \\ &= \prod_{i=0}^{j} v^{2^{s-\ell i}(\delta_i \Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \bmod N; \end{aligned}$$

since $i \leq j < n$ (remember that $y \neq x$) and since $k < \ell$, we always have that the value of $s - \ell i = (n-i)\ell - k$ is at least 1 over the integers. Therefore, $\mathcal{D}$ is always able to compute all the values $v^{2^{s-\ell i}(\delta_i \Pi_{\{y_i\neq x_i\}} a_{i,y_i})} \bmod N$ by performing the appropriate exponentiations of v^2. (Remember that $\mathcal{D}$ knows the value of all δ_i's and the value of a_{i,y_i} for all $y_i \neq x_i$.) Finally, by taking the product of the above values (reduced mod N), $\mathcal{D}$ is able to compute the value of $g^{\Pi_{i=1}^{n} a_{i,y_i}}$ as required. □

The success-probability of $\mathcal{D}$. To find the success-probability of $\mathcal{D}$, we notice that the distribution of the function $f_{N,\vec{a},g,r}$ (which is induced by the way $\mathcal{D}$ chooses $\vec{a}$ and g) is statistically close to the distribution of functions in F_n. To see this, we will need the following claims regarding the distributions of $\vec{a}$ and g.

CLAIM 5.9. *g is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$.*

Proof. Since v^2 is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$ and squaring is a permutation over the set of quadratic-residues in $\mathbb{Z}_N^*$, it immediately follows that $g = v^{2^s}$ is a uniformly distributed quadratic-residue in $\mathbb{Z}_N^*$. □

CLAIM 5.10. *Let ξ_i and a'_{i,x_i} be uniformly distributed elements in $[N]$, and denote by a_{i,x_i} the value $\xi_i + 2^{-\ell} \bmod \gamma$. Then the statistical distance of a_{i,x_i} and $a'_{i,x_i} \bmod \gamma$ is $2^{-O(n)}$.*

Proof. Note that γ divides $(P-1)(Q-1)$. Therefore, the distribution of a_{i,x_i} conditioned on the event that $\xi_i \in [(P-1)(Q-1)]$ is the same as the distribution of $a'_{i,x_i} \bmod \gamma$ conditioned on the event that $a'_{i,x_i} \in [(P-1)(Q-1)]$ (and in both cases it is simply the uniform distribution over $[\gamma]$). It remains to notice that

$$\begin{aligned} \Pr\left[\xi_i \in [(P-1)(Q-1)]\right] &= \Pr\left[a'_{i,x_i} \in [(P-1)(Q-1)]\right] \\ &= \frac{(P-1)(Q-1)}{N} \\ &= 1 - \tfrac{P+Q}{N} + \tfrac{1}{N} \\ &= 1 - 2^{-O(n)}, \end{aligned}$$

which completes the proof. □

CLAIM 5.11. *Let $f_{N,\vec{a}',g,r}$ be distributed according to F_n, and let $f_{N,\vec{a},g,r}$ be distributed as in the construction of $\mathcal{D}$. Then the statistical distance of $f_{N,\vec{a}',g,r}$ and $f_{N,\vec{a},g,r}$ is $n \cdot 2^{-O(n)}$.*

Proof. Let each element in $\vec{a}' = (a'_{1,0}, a'_{1,1}, a'_{2,0}, a'_{2,1}, \ldots, a'_{n,0}, a'_{n,1})$ be uniformly distributed in $[N]$. By Claim 5.10, we have that, for every $1 \leq i \leq n$, a_{i,x_i} and $a'_{i,x_i} \bmod \gamma$ are of statistical distance $2^{-O(n)}$. It follows (by the triangle inequality) that $\vec{a}$ and $\vec{a}' \bmod \gamma$ are of statistical distance $n \cdot 2^{-O(n)}$. It is then immediate that $f_{N,\vec{a},g,r}$ and $f_{N,\vec{a}',g,r}$ are of statistical distance $n \cdot 2^{-O(n)}$. □

Since applying any function (even a randomized one) does not increase the statistical distance, then, by Claim 5.11, we have that, for infinitely many n's,

$$\left|\Pr\left[\mathcal{M}^{f_{N,\vec{a}',g,r}}(1^n)=1\right]-\Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n)=1\right]\right|<n\cdot 2^{-O(n)}.$$

To complete the proof, we recall Definition 5.1 of the hybrid distribution, $H^J_{N,\vec{a},g,r}$. By Claim 5.6, we have that $g^{2^k\Pi_{i=1}^n a_{i,x_i}}=u\cdot w \bmod N$. Therefore, the value of the $(k+1)$st bit in the answer that $\mathcal{D}$ is supposed to give to the $(I+1)$st query is precisely equal to $\mathcal{B}_r(u\cdot w)$. Given that $J=j$ and that $\alpha=\mathcal{B}_r(u\cdot w)$ (where $u\cdot w$ is a quadratic-residue in $\mathbb{Z}_N^*$), the distribution that $\mathcal{M}$ sees is $H^j_{N,\vec{a},g,r}$. On the other hand, if α is random, then the distribution that $\mathcal{M}$ sees is $H^{j+1}_{N,\vec{a},g,r}$. We now have that, for infinitely many n's,

$$\begin{aligned}
&\left|\Pr\left[\mathcal{D}(v^2 \bmod N,N,r,\mathcal{B}_r(u\cdot w))=1\right]-\Pr\left[\mathcal{D}(v^2 \bmod N,N,r,b)=1\right]\right|\\
&=\frac{1}{t(n)\cdot\ell(n)}\cdot\left|\sum_{j=0}^{t\cdot\ell-1}\left(\Pr\left[\mathcal{D}(v^2,N,r,\mathcal{B}_r(u\cdot w))=1\mid J=j\right]-\Pr\left[\mathcal{D}(v^2,N,r,b)=1\mid J=j\right]\right)\right|\\
&=\frac{1}{t(n)\cdot\ell(n)}\cdot\left|\sum_{j=0}^{t\cdot\ell-1}\left(\Pr\left[\mathcal{M}^{H^j_{N,\vec{a},g,r}}(1^n)=1\right]-\Pr\left[\mathcal{M}^{H^{j+1}_{N,\vec{a},g,r}}(1^n)=1\right]\right)\right|\\
&=\frac{1}{t(n)\cdot\ell(n)}\cdot\left|\Pr\left[\mathcal{M}^{f_{N,\vec{a},g,r}}(1^n)=1\right]-\Pr\left[\mathcal{M}^{\rho_{n,\ell}}(1^n)=1\right]\right|\\
&\le\frac{\epsilon(n)}{t(n)\cdot\ell(n)}+n\cdot 2^{-O(n)},
\end{aligned}$$

where the distribution of $N=P\cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b\in_R\{0,1\}$, $u\cdot w$ is the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $(u\cdot w)^2=v^2\cdot w^2 \bmod N$, and w is a randomly chosen quadratic-residue in $\mathbb{Z}_N^*$. The proof of Lemma 5.4 is complete. □

REMARK 5.2. *From the proof of Lemma* 5.4*, we get that* $\mathcal{D}$ *works even if the distinguisher* $\mathcal{M}$ *has access to* $N, g,$ *and* r.

Reconstructing $u\cdot w$ mod N. Technically, $\mathcal{D}$ is not suitable to be used as an oracle for the Goldreich–Levin reconstruction algorithm, first because it is not a predictor for the value $\mathcal{B}_r(u\cdot w)$ but rather a distinguisher. Furthermore, the value of w potentially changes each time $\mathcal{D}$ is invoked (since it depends on v^2 and $\vec{\xi}$), which does not allow the reconstruction algorithm to retrieve $u\cdot w$ for any specific value of w. The transformation to a suitable predictor, however, is not difficult. By fixing the values of v^2 and $\vec{\xi}$ in advance, we are able to construct an algorithm, $\mathcal{D}_{N,\vec{\xi},v^2}$, which invokes $\mathcal{D}$ as a subroutine and with nonnegligible probability succeeds in predicting $\mathcal{B}_r(u\cdot w)$. On input r, $\mathcal{D}_{N,\vec{\xi},v^2}$ is defined as follows:

1. Sample two independent random bits α, β in $\{0,1\}$.
2. Invoke $\mathcal{D}$ on input (v^2,N,r,α); feed it with $\vec{\xi}$ on its random tape.
3. (a) If $\mathcal{D}$ outputs 1, then output α.
   (b) If $\mathcal{D}$ outputs 0, then output β.

Note that, now, the value of w does not change each time $\mathcal{D}_{N,\vec{\xi},v^2}$ is invoked. This means that it is possible to use $\mathcal{D}_{N,\vec{\xi},v^2}$ as an oracle in order to reconstruct $u\cdot w$.

LEMMA 5.5. *Assume there exists a probabilistic polynomial-time machine* $\mathcal{M}$ *satisfying inequality* (7)*, and let* $\mathcal{D}_{N,\vec{\xi},v^2}$ *be as above. Then, with probability* $\frac{\epsilon'(n)}{2}$

(over the choices of N, v^2, and $\vec{\xi}$), it holds that, for infinitely many n's,

$$\Pr\left[\mathcal{D}_{N,\vec{\xi},v^2}(r) = \mathcal{B}_r(u \cdot w)\right] > \frac{1}{2} + \frac{\epsilon'(n)}{4},$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly drawn from $\mathbb{Z}_N^*$, r is a random n-bit string, $\vec{\xi}$ is a random vector of $2n$ elements in $[N]$, $w = w(\vec{\xi}, v^2)$ is a quadratic-residue in $\mathbb{Z}_N^*$, and $u \cdot w$ is the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $(u \cdot w)^2 = v^2 \cdot w^2 \bmod N$.

Proof. By Lemma 5.4, $\mathcal{D}$ has an $(\epsilon'(n) - n \cdot 2^{-O(n)})$ advantage in distinguishing $\mathcal{B}_r(u \cdot w)$ from a randomly chosen bit. Using an averaging argument, it is easy to see that on at least an $\frac{\epsilon'(n)}{2}$ fraction of the choices of N, v^2, and $\vec{\xi}$, algorithm $\mathcal{D}$ has an $\frac{\epsilon'(n)}{2}$ advantage in distinguishing $\mathcal{B}_r(u \cdot w)$ from a randomly chosen bit. It is then straightforward that $\mathcal{D}_{N,\vec{\xi},v^2}$ can predict the value of $\mathcal{B}_r(u \cdot w)$ with advantage $\frac{\epsilon'(n)}{4}$ as required. ∎

The factoring algorithm. For the sake of completeness, we now turn to describe an algorithm, $\mathcal{A}$, which uses $\mathcal{D}_{N,\vec{\xi},v^2}$ as oracle and succeeds to $\epsilon''(n)$-factor (where $\epsilon''(n) = \Omega(\epsilon'(n)^2)$); this is in contradiction to Assumption 4.1 and will complete the proof.

LEMMA 5.6. *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$ satisfying inequality* (7)*. Then there exists a probabilistic polynomial-time algorithm, $\mathcal{A}$, that $\epsilon''(n)$-factors.*

Proof. On input N, $\mathcal{A}$ is defined as follows:

1. (a) Sample v uniformly at random in $\mathbb{Z}_N^*$, and compute $v^2 \bmod N$.
   (b) Sample $\vec{\xi} = (\xi_1, \xi_2, \ldots, \xi_n, \xi'_1, \xi'_2, \ldots, \xi'_n)$ by uniformly picking $2n$ elements in $[N]$.
2. Compute the value of $w = v^{(\Sigma_{i=0}^{n-1} \beta_i 2^{\ell(n-i)})} \bmod N$, where β_i are the coefficients (over $\mathbb{Z}$) of the polynomial $p(x) \stackrel{\text{def}}{=} \prod_{i=1}^{n}(\xi_i + x) = x^n + \sum_{i=0}^{n-1} \beta_i x^i$ (and are easily found given the ξ_i's).
3. Invoke the Goldreich–Levin reconstruction algorithm, $\mathcal{R}(1^n)$.
   (a) Whenever asked for $\mathcal{B}_{r_i}(z)$, invoke $\mathcal{D}_{N,\vec{\xi},v^2}$ on input r_i, and give its output as an answer. (Recall that $\mathcal{D}_{N,\vec{\xi},v^2}$ invokes $\mathcal{M}$ and answers its queries.)
   (b) Denote by z the output of $\mathcal{R}$.
4. Compute $u = z \cdot w^{-1} \bmod N$. Given that $\mathcal{R}$ outputs the correct value (i.e., $z = u \cdot w$), we have that $u^2 = v^2 \bmod N$. If $u \neq \pm v \bmod N$, output $gcd(u - v, N)$ which is indeed in $\{P, Q\}$. Otherwise, output "failed."

The running-time of $\mathcal{A}$. It is clear that steps 1, 2, and 4 can be carried out efficiently by $\mathcal{A}$ (and, in addition, are independent of $\mathcal{D}_{N,\vec{\xi},v^2}$). As for step 3, assuming that $\mathcal{D}_{N,\vec{\xi},v^2}$ has nonnegligible advantage in predicting $\mathcal{B}_r(u \cdot w)$ (which by Lemma 5.5 happens with nonnegligible probability), we are guaranteed (by Theorem 5.2) that $\mathcal{R}$ will terminate in polynomial time.

The success-probability of $\mathcal{A}$. Since, with probability $\frac{\epsilon'(n)}{2}$, $\mathcal{D}_{N,\vec{\xi},v^2}$ predicts the value of $\mathcal{B}_r(u \cdot w)$ with advantage $\frac{\epsilon'(n)}{4}$, then, by Theorem 5.2, we have that $\mathcal{R}$ retrieves the value of $u \cdot w$ with probability at least $\Omega(\epsilon'(n)^2)$ (over the choices of N, v^2, and $\vec{\xi}$). Note that $u \cdot w$ and w are both quadratic-residues in $\mathbb{Z}_N^*$; therefore, u

must also be a quadratic-residue in $\mathbb{Z}_N^*$. Given that, the probability that u does not equal $\pm v \bmod N$ is exactly 1/2. We are finally able to conclude that $\mathcal{A}$ $\epsilon''(n)$-factors as required. □

This completes the proof of Theorem 5.1. □

6. Using other hard-core bits. In Construction 4.1, we use the Goldreich–Levin hard-core bit, $\mathcal{B}_r$. The other, more natural, hard-core bit *in this context* is the $\mathcal{LSB}$ predicate, shown to be secure by Alexi et al. [1]. (The $\mathcal{LSB}$ predicate was the one originally used in the BBS construction.) The key property which we require from the $\mathcal{B}_r$ predicate is that its reconstruction algorithm *fixes* the unknown value and changes r throughout the process (see footnote 15). As pointed out to us by Roger Fischlin [9], a similar property is satisfied by the $\mathcal{LSB}$-based reconstruction techniques [1, 10] (see Theorem 6.1, where r is explicitly considered).

The above observation suggests that using the $\mathcal{LSB}$ predicate in Construction 4.1 (rather than using the $\mathcal{B}_r$ predicate) may yield a secure pseudorandom function. However, we were not able to prove it. What we are able to do is to slightly modify Construction 4.1 in order to obtain a secure pseudorandom function that is secure when using the $\mathcal{LSB}$ predicate.

Interestingly, the modified construction does not exactly follow the paradigm of applying the BBS generator to the unpredictable function $h(x) = g^{\Pi_{i=1}^n a_{i,x_i}}$ (but rather multiplies each of the elements in the sequence with r before applying the $\mathcal{LSB}$ predicate).

CONSTRUCTION 6.1 ($\mathcal{LSB}$ version of pseudorandom functions). *We define a function in the ensemble $F = \{F_n\}_{n\in\mathbb{N}}$. For every $n \in \mathbb{N}$, a key of a function in F_n is a tuple $(N, \vec{a}, g, r)$, where N is an n-bit Blum-integer, g is a quadratic-residue in $\mathbb{Z}_N^*$, r is an element in $\mathbb{Z}_N^*$, and $\vec{a} = (a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1}, \ldots, a_{n,0}, a_{n,1})$ is a sequence of $2n$ elements in $[N]$. For any n-bit input $x = x_1 x_2 \ldots x_n$ and for every integer function $\ell = \ell(n)$, the function $f_{N,\vec{a},g} : I^n \to I^{\ell(n)}$ is defined by*

$$f_{N,\vec{a},g,r}(x) \stackrel{\text{def}}{=} \mathcal{LSB}(r \cdot g^{\Pi_{i=1}^n a_{i,x_i}}), \mathcal{LSB}(r \cdot g^{2\Pi_{i=1}^n a_{i,x_i}}), \ldots, \mathcal{LSB}(r \cdot g^{2^{\ell-1}\Pi_{i=1}^n a_{i,x_i}}).$$

The distribution of functions in F_n is induced by the following distribution on their keys: $\vec{a}$ and g are uniform in their range, and the distribution of N is $FIG(1^n)$.

Sketch of proof of security. The proof of security of Construction 6.1 is similar to the case of the $\mathcal{B}_r$ hard-core predicate. Specifically it is shown how, given a distinguisher for the above pseudorandom functions, we can construct an algorithm $\mathcal{D}$ that on input $(v^2 \bmod N, N, r)$ predicts the value of $\mathcal{LSB}(r \cdot u \cdot w)$, where $u \cdot w$ is the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $(u \cdot w)^2 = v^2 \cdot w^2 \bmod N$, and w is a randomly chosen quadratic-residue in $\mathbb{Z}_N^*$ (which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses and is efficiently computable by $\mathcal{D}$). Using the reconstruction algorithm of Alexi et al. [1] (see [10] for tighter results), we are then able to retrieve $u \cdot w$ (and so u). As before, this means that we can extract square-roots in $\mathbb{Z}_N^*$ and consequently factor Blum-integers.

THEOREM 6.1 (see [1, 10]). *Let $z, r \in \mathbb{Z}_N^*$. Given an oracle that, on input r, predicts the value of $\mathcal{LSB}(r \cdot z)$ with advantage $\epsilon(n)$ (over the choice of r) in time $t(n)$, there exists a probabilistic polynomial-time algorithm with running-time $O(\frac{n^2 \cdot t(n)}{\epsilon(n)^2})$ that retrieves z with probability at least $\Omega(\epsilon(n))$.*

As in the proof of Theorem 5.1, the security of Construction 6.1 is proved using the following main lemma (which is the analogue of Lemma 5.4).

LEMMA 6.2. *Assume there exists a probabilistic polynomial-time machine $\mathcal{M}$ satisfying inequality (7). Then there exists a probabilistic polynomial-time algorithm $\mathcal{D}$ such that, for infinitely many n's,*

$$\left|\Pr\left[\mathcal{D}(v^2, N, r, \mathcal{LSB}(r \cdot u \cdot w)) = 1\right] - \Pr\left[\mathcal{D}(v^2, N, r, b) = 1\right]\right| > \epsilon'(n) - n \cdot 2^{-O(n)},$$

where the distribution of $N = P \cdot Q$ is $FIG(1^n)$, v is uniformly distributed in $\mathbb{Z}_N^*$, r is a random n-bit string, $b \in_R \{0,1\}$, $u \cdot w$ is the unique quadratic-residue in $\mathbb{Z}_N^*$ which satisfies $(u \cdot w)^2 = v^2 \cdot w^2 \bmod N$, and w is a randomly chosen quadratic-residue in $\mathbb{Z}_N^*$ (which is completely determined by the value of v^2 and $\mathcal{D}$'s internal coin tosses and is efficiently computable by $\mathcal{D}$).

Proof sketch. The proof of Lemma 6.2 is essentially identical to the proof of Lemma 5.4. Given v^2, N, r, and $J = I \cdot \ell + k$, the values of $g, \vec{a}, u$, and w are defined exactly as before. The following claim establishes the correctness of $\mathcal{D}$ (and is proved exactly in the same way as Claim 5.6).

CLAIM 6.1. *Let $\vec{a}, g, u$, and w be defined as above; then $\mathcal{LSB}(r \cdot g^{2^k \Pi_{i=1}^n a_{i,x_i}}) = \mathcal{LSB}(r \cdot u \cdot w)$.*

As before, this implies that the $(k+1)$st bit in the answer that the $f_{N,\vec{a},g,r}$ black-box is supposed to give to the $(I+1)$st query (and is answered with α instead) is equal to $\mathcal{LSB}(r \cdot u \cdot w)$. As we have already seen, this fact can be used by $\mathcal{D}$ in order to decide whether or not α equals $\mathcal{LSB}(r \cdot u \cdot w)$.

As for $\mathcal{D}$'s running-time, since the values of $g, \vec{a}, u$, and w are identical to the case of Lemma 5.4, it follows that $\mathcal{D}$ is able to efficiently compute both $r \cdot g^{2^{k+1} \Pi_{i=1}^n a_{i,x_i}}$ and $r \cdot g^{\Pi_{i=1}^n a_{i,y_i}}$ for all $y \neq x$. In particular, $\mathcal{D}$ can be implemented in time $poly(n, \ell(n)) \cdot t(n)$.

Finally, since the distribution of the key $(N, \vec{a}, g, r)$ chosen by $\mathcal{D}$ is identical to the distribution of the key chosen by the distinguisher in the proof of Lemma 5.4, the success-probability of $\mathcal{D}$ is identical to the success-probability of the distinguisher in the proof of Lemma 5.4. □

7. Further research. The proof of Theorem 5.1 is tailored to the specific cryptographic primitives which are used in Construction 4.1 (i.e., the "unpredictable" function $g^{\Pi_{i=1}^n a_{i,x_i}}$ and the BBS generator). An interesting open problem would be to provide an alternative proof for Theorem 5.1. Such a proof might make use of more general notions and different techniques and will hopefully shed more light on the reasons for which our construction yields a pseudorandom function. In particular, it may provide new constructions of pseudorandom functions based on more general (or more efficient) cryptographic primitives.

As we have demonstrated, in section 5.1, there exists an "unpredictable" function and a pseudorandom generator such that their composition is *not* a pseudorandom function. It should be interesting to recognize what precisely are the features of a function h_s (from an ensemble $H = \{h_s\}$) and of a pseudorandom sequence generator G that are needed in order to prove that our construction indeed yields pseudorandom functions.

Comparing to the DDH pseudorandom functions. As we have already mentioned, the efficiency of Construction 4.1 is comparable to the efficiency of the DDH functions by NR [20]. Apart from being slightly more efficient than our functions, the DDH functions have some additional properties:

- The simple algebraic structure of the DDH functions implies several attractive features (e.g., zero-knowledge proof for the value of the function, function sharing, and oblivious evaluation of the value of the function). In spite of the

similarity between the two constructions, we do not know how to prove that similar protocols are secure in our case.

- As opposed to the proof of Theorem 5.1, the security of the DDH functions does not decrease proportionally to the number of queries which are made by the adversary.[16] (This is due to the random self-reducibility of the DDH assumption [20].)

It is natural to consider the features of the DDH functions as guidelines for further research regarding our functions.

Acknowledgments. We would like to thank Roger Fischlin for pointing out the applicability of the $\mathcal{LSB}$ reconstruction techniques of [1, 10] to our construction (as described in section 6 and discussed in footnote 15). We would like to thank Shai Halevi for the observations discussed in Remark 4.1. Finally, we would like to thank the anonymous referees for many helpful comments.

REFERENCES

[1] W. B. Alexi, B. Chor, O. Goldreich, and C.-P. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[2] D. Angluin and D. Lichtenstein, *Provable Security of Cryptosystems: A Survey*, Tech. report 288, Department of Computer Science, Yale University, New Haven, CT, 1983.

[3] E. Biham, D. Boneh, and O. Reingold, *Breaking generalized Diffie-Hellman modulo a composite is no easier than factoring*, Inform. Process. Lett., 70 (1999), pp. 83–87.

[4] D. Boneh, *The decision Diffie-Hellman problem*, in Proceedings of the Third Algorithmic Number Theory Symposium, Lecture Notes in Comput. Sci. 1423, Springer-Verlag, New York, 1998, pp. 48–63.

[5] L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudorandom number generator*, SIAM J. Comput., 15 (1986), pp. 364–383.

[6] M. Blum and S. Goldwasser, *An efficient probabilistic public-key encryption scheme which hides all partial information*, in Advances in Cryptology—CRYPTO'84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, New York, 1985, pp. 289–302.

[7] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[8] G. Brassard, *On computationally secure authentication tags requiring short secret shared keys*, in Advances of Cryptology: Proceedings of CRYPTO'82, D. Chaum, R. L. Rivest, and A. T. Sherman, eds., Plenum Press, New York, 1983, pp. 79–86.

[9] R. Fischlin, *private communication*, Department of Mathematics, Johann Wolfgang Goethe University, Frankfurt am Main, Germany, 2000.

[10] R. Fischlin and C. P. Schnorr, *Stronger security proofs for RSA and Rabin bits*, J. Cryptology, 13 (2000), pp. 221–244.

[11] O. Goldreich, *Foundations of Cryptography—Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[12] O. Goldreich, *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, Algorithms Combin. 17, Springer-Verlag, Berlin, 1999.

[13] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.

[14] O. Goldreich and L. Levin, *A hard-core predicate for all one-way functions*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 25–32.

[15] S. Halevi, *Efficient commitment schemes with bounded sender and unbounded receiver*, J. Cryptology, 12 (1999), pp. 77–89.

[16] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.

[17] M. Luby, *Pseudo-Randomness and Applications*, Princeton University Press, Princeton, NJ, 1996.

[16] See [17] for a discussion of security preserving reductions (called polypreserving in their terminology).

[18] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[19] M. Naor and O. Reingold, *Synthesizers and their application to the parallel construction of pseudo-random functions*, J. Comput. System Sci., 58 (1999), pp. 336–375.

[20] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 458–467.

[21] M. Naor and O. Reingold, *From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MAC's*, in Advances in Cryptology—CRYPTO'98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 267–282.

[22] A. M. Odlyzko, *The future of integer factorization*, RSA CryptoBytes, 2 (1995), pp. 5–12.

[23] M. O. Rabin, *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*, Tech. report TR-212, MIT Laboratory for Computer Science, Cambridge, MA, 1979.

[24] O. Reingold, *Pseudo-Random Synthesizers, Functions and Permutations*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1998.

[25] C. P. Schnorr, *Security of 2^t-root identification and signatures*, in Advances in Cryptology—CRYPTO'96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 143–156.

[26] U. V. Vazirani and V. V. Vazirani, *Efficient and secure pseudo-random number generation*, in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1984, pp. 458–463.

[27] A. C. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1405–1423

GENERATORS OF $H_1(\Gamma_h, \mathbb{Z})$ FOR TRIANGULATED SURFACES: CONSTRUCTION AND CLASSIFICATION*

R. HIPTMAIR† AND J. OSTROWSKI‡

Abstract. We consider a bounded Lipschitz-polyhedron $\Omega \subset \mathbb{R}^3$ of general topology equipped with a tetrahedral triangulation that induces a mesh Γ_h of the surface $\partial\Omega$. We seek a maximal set of surface edge cycles that are independent in $H_1(\Gamma_h, \mathbb{Z})$ and bounding with respect to the exterior of Ω.

We present an algorithm for constructing suitable 1-cycles in Γ_h: First, representatives of a basis of the homology group $H_1(\Gamma_h, \mathbb{Z})$ are constructed, merely using the combinatorial description of the surface mesh Γ_h. Then, a duality pairing based on linking numbers is used to determine those combinations that are bounding with respect to $\mathbb{R}^3 \setminus \Omega$. This is the key to circumventing a triangulation of the exterior region $\mathbb{R}^3 \setminus \Omega$ in the computations. For shape-regular, quasi-uniform families of meshes, the asymptotic complexity of the algorithm is shown to be $O(N^2)$, where N is the number of edges of Γ_h.

The scheme provides an essential preprocessing step for all boundary element methods for eddy current simulation, which rely on discrete divergence-free vectorfields and their description through stream functions.

Key words. cellular homology, nonbounding cycles, linking numbers, surface stream functions, computational electromagnetism

AMS subject classifications. 05C10, 68R10, 35Q60, 65D32, 65D99

PII. S0097539701386526

1. Background.

This paper straddles the fields of algebraic topology, graph theory, and numerical approximation. More precisely, starting out from boundary element method computations of electromagnetic fields, we encounter a problem of discrete cellular homology and solve it via continuous fields and numerical approximations.

Throughout the paper we have in mind the following task faced in computational electromagnetism (for details see [13]): A Lipschitz-polyhedron Ω in three-dimensional Euclidean space, the conductor, is equipped with a tetrahedral triangulation Ω_h. The goal is to compute the eddy currents in Ω that are triggered by an exciting loop current. This involves the solution of partial differential equations on the entire space. A boundary element method is used to take into account the unbounded exterior $\Omega' := \mathbb{R}^3 \setminus \Omega$. It relies on a divergence-free surface current $\boldsymbol{\lambda}$ as auxiliary unknown, which is discretized by means of twisted discrete 1-forms on the surface mesh $\Gamma_h := \Omega_h \cap \partial\Omega$. These 1-forms are special representatives of Whitney-forms [24]. Their degrees of freedom are associated with the edges of the surface mesh. Thus, they can be regarded as complex-valued 1-cochains of the simplicial cellular complex provided by Γ_h.

The discrete surface current has to be a closed form. In the context of a boundary element method the most efficient way to realize this constraint is through discrete

*Received by the editors March 20, 2001; accepted for publication (in revised form) February 8, 2002; published electronically July 1, 2002.

http://www.siam.org/journals/sicomp/31-5/38652.html

†Nachwuchsgruppe SFB 382, Universität Tübingen, D-72076 Tübingen, Germany (ralf@hiptmair.de). The work of this author was supported by Deutsche Forschungsgemeinschaft as part of SFB 382.

‡Transferbereich 26, Universität Tübingen, D-72076 Tübingen, Germany (joerg@na.uni-tuebingen.de). The work of this author was supported by Deutsche Forschungsgemeinschaft as part of TFB 26.

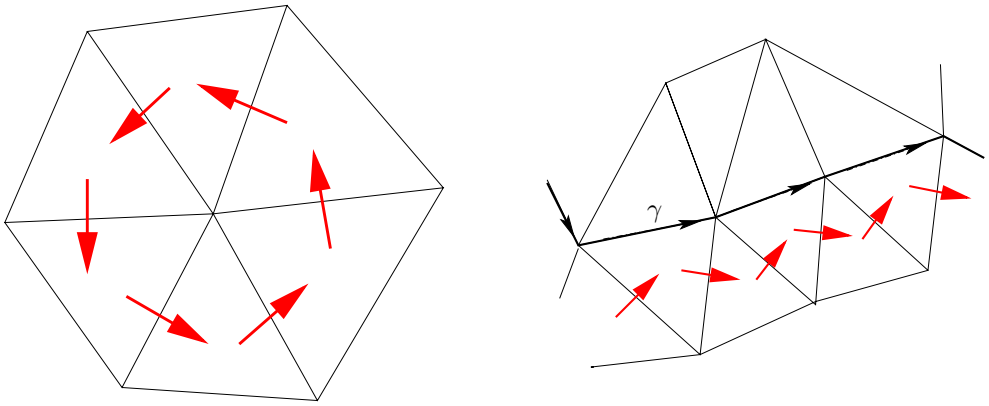


FIG. 1. *Left: Current generated by local stream function. Right: Current sheet along a* 1*-cycle* γ.

potentials (stream functions; see Figure 1); i.e., the 1-cochains are represented as exterior derivatives of 0-cochains. However, if the first Betti number $\beta_1(\Gamma)$, $\Gamma := \partial\Omega$, does not vanish, potentials do not supply all of the desired space. What is missing is the space spanned by representatives of a basis of the cohomology group $H^1(\Gamma_h, \mathbb{C})$. Its dimension is known to be equal to $\beta_1(\Gamma)$.

As $\partial\Gamma = \emptyset$ and Γ is orientable, we can exploit the (Poincaré-)duality between cohomology and homology $H^1(\Gamma_h, \mathbb{C}) \cong H_1(\Gamma_h, \mathbb{C})$. Moreover, as these groups are torsion free [15, section II], we need only consider integral coefficients, i.e., $H_1(\Gamma_h, \mathbb{Z})$. Representatives of the latter group are given by nonbounding edge cycles in Γ_h. If we know them, we can recover surface currents in $H^1(\Gamma_h, \mathbb{C})$ as sheets of current traveling along the 1-cycles as sketched in Figure 1 (left). Yet, not all of $H_1(\Gamma_h, \mathbb{Z})$ is really desired, because $\boldsymbol{\lambda}_h$ can be regarded as an approximation of the trace $\mathbf{H} \times \mathbf{n}$ ($\mathbf{n}$ exterior unit normal) of the magnetic field $\mathbf{H}$. Ampere's law tells us that

$$\int_{\partial S} \mathbf{H} \cdot d\vec{s} = \int_S \mathbf{j} \cdot d\vec{S}, \tag{1.1}$$

where S is any orientable 2-surface and $\mathbf{j}$ stands for a current. If S has its boundary on Γ, we expect

$$\int_{\partial S} \mathbf{n} \times \boldsymbol{\lambda}_h \cdot d\vec{s} = \int_S \mathbf{j} \cdot d\vec{S}. \tag{1.2}$$

Outside Ω currents cannot exist. This means that the integral of $\mathbf{n} \times \boldsymbol{\lambda}_h$ along any 1-cycle in Γ_h will vanish if that cycle is bounding relative to $\Omega' := \mathbb{R}^3 \setminus \Omega$. Current sheets skirting nonbounding surface edge cycles that are not bounding with respect to Ω' fail to meet this constraint. Thus *relevant* surface edge cycles in $H_1(\Gamma_h, \mathbb{Z})$ are marked by the property that they are bounding relative to Ω'. Zeroing in on relevant cycles, we will end up with a subgroup of $H_1(\Gamma_h, \mathbb{Z})$ of dimension $\frac{1}{2}\beta_1(\Gamma)$. To get an idea of what relevant cycles look like, imagine a ring-shaped hollow Ω, for instance the closed circular pipe of Figure 2. Only the circles γ_1 on the outer surface surrounding the central hole and γ_2 linking the inner cavity qualify as relevant independent representatives of $H_1(\Gamma, \mathbb{Z})$.

It is not only a matter of economy to discard the redundant generators of $H^1(\Gamma_h, \mathbb{C})$. On the contrary, it is essential for the applicability of the boundary element method,

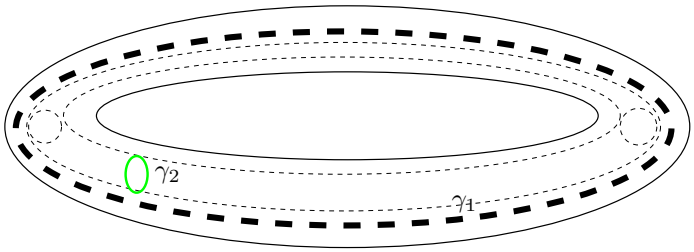


FIG. 2. *The two relevant surface* 1*-cycles* γ_1 *and* γ_2 (*modulo homology*) *for a toroidal pipe.*

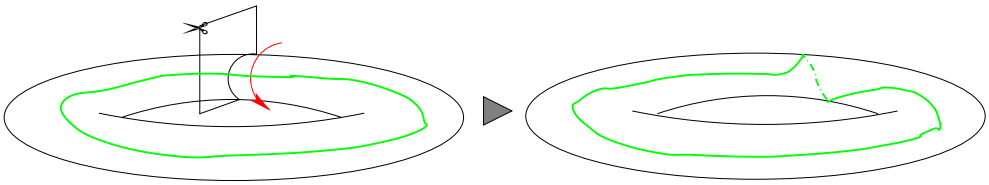


FIG. 3. *Destroying a* 1*-cycle bounding relative to* Ω'.

because in the model excitations are taken into account by prescribing the total current in a loop-shaped part of Ω, the so-called inductor. In computational practice, this is done by fixing the contribution of the relevant representative of $H^1(\Gamma_h, \mathbb{C})$ to the boundary element space. Choosing one that does not bound with respect to Ω' is physically meaningless. In Figure 2 only γ_1 is suitable to impose a loop current.

In principle, relevant 1-cycles can be constructed based on a triangulation of $\mathcal{O}\backslash\Omega$, where $\mathcal{O}$ is a sufficiently large cube containing $\bar{\Omega}$. Then there are algorithms for the construction of so-called cutting surfaces in Ω', whose boundaries are the desired 1-cycles. Pioneering work in this field was done by Brown [4]. His algorithm finds generators of $H_1(\bar{\Omega}, \mathbb{Z})$ by successive retractions applied to elements of Ω_h. However, it is not clear whether the method can cope with arbitrary meshes, and the resulting cycles are not located on Γ. On top of that, the computational costs might behave like $O(N^3)$, where N is the number of edges of Ω_h. Profound theoretical and algorithmic investigations on the construction of cutting surfaces were conducted by Kotiuga [17, 16, 15]. In [12, Chap. 3], an actual computation of cutting surfaces is presented. It involves the solution of a finite element problem on the triangulated complement Ω'. Yet, this option is not available to us. First, a rationale for using boundary element methods is to avoid meshing parts of the exterior of Ω. Second, the computations turn out to be extremely expensive. Their complexity is at least $O(N^3)$, where N now stands for the number of elements in the exterior mesh. Very effective algorithms for construction of generators of $H_1(\Gamma_h, \mathbb{Z})$ have been developed in computational topology [23, 18]. They can guarantee quite a few particular properties of the cycles but short of picking relevant cycles.

At first glance it seems that we could successfully tackle the problem in an entirely discrete setting, relying on the connectivity of Ω_h alone. Yet, consider a plain triangulated torus. Cut it at its small circle, twist by 2π, and reconnect as in Figure 3. If we had found a surface 1-cycle bounding with respect to the exterior, this operation will render it nonrelevant. However, the combinatorial description of the mesh remains the same. It is evident that it is impossible to find the desired 1-cycles merely using combinatorial information about Ω_h. Unless we want to use an exterior

mesh, we also have to rely on information about the geometry of Ω.

The main objective of this paper is to convey how blending combinatorial and geometric techniques can yield a fast algorithm for the construction and classification of generators of $H_1(\Gamma_h,\mathbb{Z})$. In section 2 we fix the framework and basic concepts of cellular homology. Then, to make the presentation self-contained, in section 3 we follow [19] and introduce the construction of generators of $H_1(\Gamma_h,\mathbb{Z})$ based on graph algorithms. Those are reminiscent of so-called cotree gauging methods, widely employed in computational electromagnetism (see [2, section 5.3], [8], and [14]). In section 4 we appeal to the venerable topological notion of linking numbers to come up with an algebraic characterization of 1-cycles bounding with respect to the exterior. We point out that a different classification problem has been treated in [7] using rather similar ideas. The numerical computation of linking numbers is detailed in section 5.

2. Setting and notations. Denote by $\mathcal{S}_0 := \mathcal{V}_h, \mathcal{S}_1 := \mathcal{E}_h$, and $\mathcal{S}_2 := \mathcal{F}_h$ the sets of vertices, edges, and faces of the surface mesh Γ_h covering Γ. From an abstract point of view, Γ_h can be regarded as a nondegenerate simplicial cellular complex [9]. We say that an l-simplex $\mathbf{x} \in \mathcal{S}_l$ is contained in another k-simplex $\mathbf{y} \in \mathcal{S}_k$, $\mathbf{x} \prec \mathbf{y}$, if all vertices of $\mathbf{x}$ are vertices of $\mathbf{y}$, too. For $\mathbf{x} \in \mathcal{S}_l$ we introduce the k-simplicial neighborhood

$$\mathcal{S}_k(\mathbf{x}) := \{\mathbf{y} \in \mathcal{S}_k,\ \mathbf{x} \prec \mathbf{y} \text{ or } \mathbf{y} \prec \mathbf{x}\}.$$

Each simplex is endowed with an interior orientation prescribed by an ordering of its vertices. From a combinatorial point of view, the oriented triangulation Γ_h can be completely described by the *incidence relations* $\iota_{l-1}^{l} : \mathcal{S}_{l-1} \times \mathcal{S}_l \mapsto \{-1,0,1\}$, where $\iota(\mathbf{x},\mathbf{y}) = 0$, if $\mathbf{x}$ is not contained in $\mathbf{y}$, and, otherwise, $\iota(\mathbf{x},\mathbf{y}) = \pm 1$, if the induced orientation of $\mathbf{x}$ with respect to $\mathbf{y}$ is the same as/opposite to the interior orientation of $\mathbf{x}$ (cf. [11, section 2.2] and [2, section 5.2]).

Mappings $\mathcal{S}_l \mapsto \mathbb{Z}$ are called surface l-chains. They form a free Abelian group $C_l(\Gamma_h,\mathbb{Z})$. We adopt the convenient sum notation $\gamma := \sum_{\mathbf{x}\in\mathcal{S}_l} \alpha_{\mathbf{x}}\mathbf{x}$, $\alpha_{\mathbf{x}} \in \mathbb{Z}$ for a surface l-chain γ with $\gamma(\mathbf{x}) = \alpha_{\mathbf{x}}$. The coefficients $\alpha_{\mathbf{x}}$ will be referred to as weights. We define the set of l-simplices contained in an l-chain as

$$\mathcal{S}_l(\gamma) := \{\mathbf{x} \in \mathcal{S}_l,\ \gamma(\mathbf{x}) \neq 0\}, \qquad \gamma \in C_l(\Gamma_h,\mathbb{Z}).$$

For $l > 0$ we define the *boundary* $\partial_l\mathbf{x}$ of $\mathbf{x} \in \mathcal{S}_l$ as

$$\partial_l\mathbf{x} := \sum_{\mathbf{y}\in\mathcal{S}_{l-1}(\mathbf{x})} \iota(\mathbf{y},\mathbf{x})\cdot\mathbf{y} \quad \Rightarrow \quad \partial_l\mathbf{x} \in C_{l-1}(\Gamma_h,\mathbb{Z})$$

and get the *boundary homomorphism* $\partial_l : C_l(\Gamma_h,\mathbb{Z}) \mapsto C_{l-1}(\Gamma_h,\mathbb{Z})$ by extension:

$$\partial_l\left(\sum_{\mathbf{x}\in\mathcal{S}_l}\alpha_{\mathbf{x}}\mathbf{x}\right) := \sum_{\mathbf{x}\in\mathcal{S}_l}\alpha_{\mathbf{x}}\partial_l\mathbf{x}.$$

We will need cycles and 1-boundaries, i.e., subgroups of $C_1(\Gamma_h,\mathbb{Z})$ defined by

$$Z_1(\Gamma_h,\mathbb{Z}) := \mathrm{Ker}(\partial_1), \qquad B_1(\Gamma_h,\mathbb{Z}) := \mathrm{Im}(\partial_2),$$

respectively (cf. [22, section IV,V] for a lucid exposition). A 1-cycle γ is said to be bounding in Γ_h if $\gamma \in B_1(\Gamma_h,\mathbb{Z})$. Two 1-cycles are called *homologous* if their difference is a boundary, that is,

$$\omega,\eta \in C_1(\Gamma_h,\mathbb{Z}): \quad \omega \sim \eta \quad :\Leftrightarrow \quad \omega - \eta \in B_1(\Gamma_h,\mathbb{Z}).$$

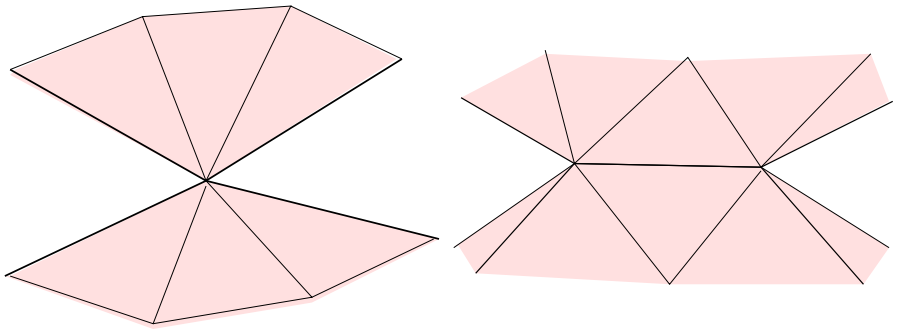


FIG. 4. *Surface configurations not possible in the case of a Lipschitz-polyhedron.*

A fundamental property of the incidence relations guarantees that $B_1(\Gamma_h, \mathbb{Z}) \subset Z_1(\Gamma_h, \mathbb{Z})$. The quotient group is called the first homology group $H_1(\Gamma_h, \mathbb{Z}) := Z_1(\Gamma_h, \mathbb{Z})/B_1(\Gamma_h, \mathbb{Z})$. It has finite rank $\beta_1(\Gamma_h)$, which is called the first Betti number of Γ_h [20, section 5.2].

As the current task reaches beyond mere combinatorics, some assumptions on the geometry of Ω have to be made. We demand that Ω has a Lipschitz-continuous boundary; that is, Γ has to have a local representation as the graph of a Lipschitz-continuous function [10, Sect 1.2.1]. Topologically speaking, this forces $\bar{\Omega}$ to be homeomorphic to a compact domain with a smooth boundary. First, this implies $\partial\Gamma = \emptyset$ and that each $\mathbf{t} \in \mathcal{F}_h$ is a face of a tetrahedron of Ω_h. A second consequence is that Γ is orientable. Thus, we can fix an orientation of $\partial\Omega$ and endow all triangles $\mathbf{t} \in \mathcal{F}_h$ with the induced orientation. Third, the surface is "locally flat" in the sense that exactly two faces in $\mathcal{F}_h$ are incident with each edge in $\mathcal{E}_h$. Situations like the "double cone" and "double ridge" depicted in Figure 4 are ruled out.

As the meshes are used for finite element computations, it also makes sense to reign in distortions of tetrahedra. We attribute the mesh Ω_h a shape-regularity measure ρ, $\rho > 0$, if

$$\max_{T\in\Omega_h} \frac{\operatorname{diam}(T)}{\sup\{r > 0 : \exists \mathbf{c} \in T, |\mathbf{x} - \mathbf{c}| < r \Rightarrow \mathbf{x} \in T\}} \leq \rho, \tag{2.1}$$

with $\operatorname{diam}(T) := \sup_{\mathbf{x},\mathbf{y}\in T} |\mathbf{x} - \mathbf{y}|$ [5, Chap. 3, section 3.1]. We infer that

$$\tfrac{4}{3}\pi \left(\operatorname{diam}(T)/\rho\right)^3 \leq \operatorname{Vol}(T) \leq \tfrac{1}{6} \operatorname{diam}(T)^3, \quad T \in \Omega_h. \tag{2.2}$$

Another common concept is q-quasi-uniformity [5], $0 < q \leq 1$, of a mesh that claims

$$q^{-1} \min_{T\in\Omega_h} \operatorname{diam}(T) \geq \max_{T\in\Omega_h} \operatorname{diam}(T). \tag{2.3}$$

The expression on the right-hand side is commonly called the meshwidth of Ω_h. Obviously, (2.1) and (2.3) hold for Γ_h instead of Ω_h with the same constants.

3. Construction of representatives of basis of $H_1(\Gamma_h, \mathbb{Z})$. This part of the algorithm is purely combinatorial and relies on interpreting Γ_h as a graph. We are using the standard definition of a graph as an ordered triple (VG, EG, Ψ) of sets VG ("vertices"), EG ("edges")[1] and an incidence function $\Psi : EG \mapsto \{VG \times VG\}$ [1, Chap. 1].

[1] Be aware of the ambiguity of the terms "vertex" and "edge," which may refer either to a graph or simplices of a mesh.

We interpret Γ_h as a graph in two different ways: The incidence function $\Psi_{\mathfrak{D}} : \mathcal{E}_h \mapsto \mathcal{F}_h$ of the graph $\mathfrak{D} := (\mathcal{F}_h, \mathcal{E}_h, \Psi_{\mathfrak{D}})$ is given by $\Psi_{\mathfrak{D}}(\mathbf{e}) := \mathcal{S}_2(\mathbf{e})$. Due to $\sharp\mathcal{S}_2(\mathbf{e}) = 2$ for all $\mathbf{e} \in \mathcal{E}_h$ it is well defined.[2] This graph $\mathfrak{D}$ is commonly known as the dual graph of the mesh Γ_h. The other graph is the actual edge graph $\mathfrak{G} := (\mathcal{V}_h, \mathcal{E}_h, \Psi_{\mathfrak{G}})$ with incidence function $\Psi_{\mathfrak{G}} : \mathcal{E}_h \mapsto \mathcal{V}_h$, $\Psi_{\mathfrak{G}}(\mathbf{e}) := \mathcal{S}_0(\mathbf{e})$.

The components $\mathfrak{D}^i$, $i = 1, \ldots, p$, $p \in \mathbb{N}$, of $\mathfrak{D}$ correspond to the different connected components of the surface Γ. In a straightforward manner they give rise to components $\Gamma_h^1, \ldots, \Gamma_h^p$ of Γ_h. If a 1-cycle is bounding in Γ_h all its restrictions to connected components have to be bounding, too. Otherwise, two triangles of different components would have to be adjacent, which is impossible. Therefore,

$$H_1(\Gamma_h, \mathbb{Z}) = \bigoplus_{i=1}^{p} H_1(\Gamma_h^i, \mathbb{Z}). \tag{3.1}$$

Hence, without loss of generality, we can focus on a single component Γ_h^i. Its sets of simplices will be tagged by a superscript i. Note that the geometric constraints enforce $\mathcal{V}_h^i \cap \mathcal{V}_h^j = \emptyset$ for $i \neq j$. Thus, a natural partitioning of the vertex-graph $\mathfrak{G}$ into components $\mathfrak{G}^1, \ldots, \mathfrak{G}^p$ is also implied.

Let $T_{\mathfrak{D}}^i$ be a spanning tree of $\mathfrak{D}^i$, i.e., a spanning subgraph of $\mathfrak{D}^i$ that is a tree (cf. Figure 8). It induces a natural partitioning of $\mathcal{E}_h^i$ into edges contained in $T_{\mathfrak{D}}^i$ and the complement set $\mathcal{E}_\flat^i$.

LEMMA 3.1. *No nontrivial* 1*-cycle only comprising edges in* $\mathcal{E}_\flat^i$ *can be bounding in* Γ_h^i.

Proof. Assume that there is a nontrivial bounding 1-cycle ω in $\mathcal{E}_\flat^i$; i.e., there is $\varphi \in C_2(\Gamma_h^i, \mathbb{Z})$, $\varphi \neq 0$ such that $\partial_2 \varphi = \omega$. If two triangles $\mathbf{t}, \mathbf{t}' \in \varphi$ share an edge $\mathbf{e}$ then $\iota(\mathbf{e}, \mathbf{t})\iota(\mathbf{e}, \mathbf{t}') = -1$ as $\mathbf{t}$ and $\mathbf{t}'$ carry the orientation induced by Γ. If $\mathbf{e}$ is not contained in $\mathcal{S}_1(\omega)$, this means that $\mathbf{t}$ and $\mathbf{t}'$ must have the same weight in φ. Eventually,

$$\mathbf{t}, \mathbf{t}' \in \mathcal{F}_h : \quad (\mathcal{S}_1(\mathbf{t}) \cap \mathcal{S}_1(\mathbf{t}')) \subset \mathcal{E}_h^i \quad \Rightarrow \quad \varphi(\mathbf{t}) = \varphi(\mathbf{t}').$$

Hence, two triangles connected by a path in $T_{\mathfrak{D}}^i$ have the same weights in φ. This will be true for all $\mathbf{t} \in \mathcal{F}_h$, since $T_{\mathfrak{D}}^i$ is a spanning tree: $\varphi = \kappa \cdot \sum_{\mathbf{t} \in \mathcal{F}_h^i} \mathbf{t}$ for some $\kappa \in \mathbb{Z} \setminus \{0\}$. However, the boundary of Γ_h^i, which is considered as a 2-chain, is empty. This refutes the assumption. ∎

LEMMA 3.2. *Each* $\gamma \in Z_1(\Gamma_h^i, \mathbb{Z})$ *is homologous to a* 1*-cycle comprising only edges in* $\mathcal{E}_\flat^i$.

Proof. Fix an arbitrary $\mathbf{t}_r \in \mathcal{F}_h$ as "root" of $T_{\mathfrak{D}}^i$. This makes it possible to assign to each triangle $\mathbf{t}$ of $T_{\mathfrak{D}}^i$ a unique number $d(\mathbf{t}) \in \mathbb{N}$, its distance to the root, i.e., the length of the unique path in $T_{\mathfrak{D}}^i$ connecting $\mathbf{t}$ and $\mathbf{t}_r$. Then set

$$d(\mathbf{e}) := \min\{d(\mathbf{t}),\, \mathbf{e} \prec \mathbf{t}\}, \qquad \mathbf{e} \in \mathcal{E}_h.$$

Please observe that (identifying $T_{\mathfrak{D}}^i$ with its edge set)

$$\mathbf{t} \in \mathcal{F}_h,\ d(\mathbf{t}) > 0 \quad \Rightarrow \quad \begin{cases} \exists_1 \mathbf{e} \in \mathcal{S}_1(\mathbf{t}) :\ d(\mathbf{e}) = d(\mathbf{t}) - 1, \\ d(\mathbf{e}') = d(\mathbf{t})\ \forall \mathbf{e}' \in (\mathcal{S}_1(\mathbf{t}) \cap T_{\mathfrak{D}}^i) \setminus \{\mathbf{e}\} \end{cases} \tag{3.2}$$

by elementary properties of the tree.

[2] $\sharp$ extracts the cardinality of a finite set.

We sort the edges of $T^i_{\mathfrak{D}}$ with respect to this "distance function," get a sequence $(\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_M)$, and write $\mathcal{TE}_k := \{\mathbf{e}_1, \dots, \mathbf{e}_k\}$, $k = 1, \dots, M$. Let j be the smallest index of an edge in $T^i_{\mathfrak{D}}$ contained in a cycle γ, i.e., $\mathcal{S}_1(\gamma) \cap \mathcal{TE}_{j-1} = \emptyset$. Select

$$\mathbf{t} \in \mathcal{S}_2(\mathbf{e}_j) \text{ such that } d(\mathbf{t}) = \max\{d(\mathbf{t}'),\, \mathbf{t}' \in \mathcal{S}_2(\mathbf{e}_j)\} = d(\mathbf{e}_j) + 1.$$

Transform γ into a homologous 1-cycle γ' according to

$$\gamma' := \gamma - \gamma(\mathbf{e}_j) \cdot \iota(\mathbf{e}_j, \mathbf{t}) \cdot \partial_2 \mathbf{t}.$$

By definition of the boundary operator $\gamma'(\mathbf{e}_j) = 0$, but γ' will probably contain the other edges of $\mathbf{t}$, which might belong to $T^i_{\mathfrak{D}}$. Fortunately, as we see from (3.2), their distance to the root will be greater than that of $\mathbf{e}_j$ in case they belong to $T^i_{\mathfrak{D}}$. This establishes $\mathcal{E}(\gamma') \cap \mathcal{TE}_j = \emptyset$. Repeating this construction, we arrive at a path $\tilde{\gamma}$ with $\tilde{\gamma} \sim \gamma$ and $\mathcal{E}(\tilde{\gamma}) \cap \mathcal{TE}_M = \emptyset$, sloppily speaking, a homologous cycle in $\mathcal{E}^i_\flat$. □

LEMMA 3.3. *The subgraph* $\mathfrak{G}^i_\flat := (\mathcal{V}^i_h, \mathcal{E}^i_\flat, \Psi_{\mathfrak{G}\,|\mathcal{E}^i_\flat})$ *is connected.*

Proof. Assume that there are two components in $\mathfrak{G}^i_\flat$, spawning a partition $\mathcal{V}^i_h = \mathcal{V}_1 \cup \mathcal{V}_2$. Define

$$\mathcal{E}_B := \{\mathbf{e} \in \mathcal{E}^i_h,\, \mathcal{S}_0(\mathbf{e}) \cap \mathcal{V}_1 \neq \emptyset \text{ and } \mathcal{S}_0(\mathbf{e}) \cap \mathcal{V}_2 \neq \emptyset\}$$

and note that $\mathcal{E}_B \neq \emptyset$ as $\mathfrak{G}^i$ is connected. Next, set

$$\mathcal{F}_B := \{\mathbf{t} \in \mathcal{F}^i_h,\, \mathcal{S}_1(\mathbf{t}) \cap \mathcal{E}_B \neq \emptyset\}.$$

By the definition of components of a graph we know $\mathcal{E}_B \cap \mathcal{E}^i_\flat = \emptyset$, hence $\mathcal{E}_B \subset T^i_{\mathfrak{D}}$, where, for convenience, the tree is identified with the set of edges it contains. For all $\mathbf{t} \in \mathcal{F}_B$ both $\mathcal{S}_0(\mathbf{t}) \cap \mathcal{V}_1 \neq \emptyset$ and $\mathcal{S}_0(\mathbf{t}) \cap \mathcal{V}_2 \neq \emptyset$, which establishes

$$\sharp(\mathcal{S}_1(\mathbf{t}) \cap \mathcal{E}_B) = 2.$$

We conclude that in the subgraph $(\mathcal{F}_B, \mathcal{E}_B, \Psi_{\mathfrak{D}\,|\mathcal{E}_B})$ of $T^i_{\mathfrak{D}}$, which is a valid graph as $\sharp(\mathcal{S}_2(\mathbf{e}) \cap \mathcal{F}_B) = 2$ holds for all $\mathbf{e} \in \mathcal{E}_B$, every vertex has degree 2 such that it must contain a circuit. This contradicts its being contained in the tree $T^i_{\mathfrak{D}}$. □

According to the previous lemma we can find a single spanning tree $T^i_{\mathfrak{G}}$ of $\mathfrak{G}^i_\flat$ (see Figure 9). It also defines a subset of $\mathcal{E}^i_\flat$ in a natural fashion. Viewing $T^i_{\mathfrak{G}}$ as a set of edges, we write $\mathcal{E}^i_* := \mathcal{E}^i_\flat \setminus T^i_{\mathfrak{G}}$ for the remainder. Since $T^i_{\mathfrak{G}}$ is a spanning tree in $\mathfrak{G}^i_\flat$, for each $\mathbf{e} \in \mathcal{E}^i_*$ we can find a unique fundamental circuit $\mathbf{s}_\mathbf{e}$ in $\mathfrak{G}^i_\flat$ that, except for $\mathbf{e}$, only contains edges in $T^i_{\mathfrak{G}}$ [1, section 1.8] (see Figure 10).

LEMMA 3.4. *Every vertex in* $\mathbf{s}_\mathbf{e}$ *belongs to exactly two edges of* $\mathbf{s}_\mathbf{e}$, $\mathbf{e} \in \mathcal{E}^i_*$.

Proof. The proof is a straightforward consequence of [1, Thm. 1.2]. □

From the above lemma we infer that the $\mathbf{s}_\mathbf{e}$, $\mathbf{e} \in \mathcal{E}^i_*$, do not intersect themselves. Thus, it confirms that $\mathbf{s}_\mathbf{e}$ can be easily converted into a 1-cycle $\gamma_\mathbf{e}$ by fixing $\gamma_\mathbf{e}(\mathbf{e}) = 1$ and assigning weights ± 1 to its edges. The assignment of weights can proceed sequentially along $\mathbf{s}_\mathbf{e}$.

THEOREM 3.5. $\{\gamma_\mathbf{e},\, \mathbf{e} \in \mathcal{E}^i_*\}$ *represents a basis of* $H_1(\Gamma^i_h, \mathbb{Z})$.

Proof. Lemma 3.1 tells us that $\gamma_\mathbf{e} \notin B_1(\Gamma^i_h, \mathbb{Z})$. Moreover, as $\gamma_\mathbf{e}(\mathbf{e}') = \delta_{\mathbf{e},\mathbf{e}'}$, $\mathbf{e}, \mathbf{e}' \in \mathcal{E}^i_*$, the different 1-cycles are independent in $H_1(\Gamma^i_h, \mathbb{Z})$.

Let $\eta \in Z_1(\Gamma^i_h, \mathbb{Z})$. Due to Lemma 3.2 we can find a homologous 1-cycle η' covering only edges in $\mathcal{E}^i_\flat$. Set

$$\pi := \eta' - \sum_{\mathbf{e} \in \mathcal{E}^i_*} \eta'(\mathbf{e}) \cdot \gamma_\mathbf{e}.$$

$\mathcal{D}_h := \mathcal{F}_h$; $\mathcal{E}_h^D := \emptyset$;
while $(\mathcal{D}_h \neq \emptyset)$ {
  Pick $\mathbf{t} \in \mathcal{D}_h$; Q $:= \emptyset$; Q.push_back($\mathbf{t}$); $\mathcal{D}_h := \mathcal{D}_h \setminus \{\mathbf{t}\}$;
  while $(Q \neq \emptyset)$ {
    $\mathbf{t}$:= Q.pop_front();
    foreach $(\mathbf{e}' \in \mathcal{S}_1(\mathbf{t}))$ {
      $\{\mathbf{t}'\} := \mathcal{S}_2(\mathbf{e}') \setminus \{\mathbf{t}\}$;
      if $(\mathbf{t}' \in \mathcal{D}_h)$ { $\mathcal{D}_h := \mathcal{D}_h \setminus \{\mathbf{t}'\}$; Q.push_back($\mathbf{t}'$); $\mathcal{E}_h^D := \mathcal{E}_h^D \cup \{\mathbf{e}'\}$; }
    }}}

FIG. 5. *Algorithm: Construction of trees* $T_{\mathfrak{D}}^i$. *Q is a FIFO container as supplied by the* deque-*container of the STL* [21, *section* 17.2.3].

$\mathcal{L}_h := \mathcal{V}_h$; $\mathcal{E}_*^i := \emptyset$;
while $(\mathcal{L}_h \neq \emptyset)$ {
  Pick $\mathbf{v} \in \mathcal{L}_h$; $\mathbf{v}$.depth $:= 0$; $\mathcal{L}_h := \mathcal{L}_h \setminus \{\mathbf{v}\}$; Q $:= \emptyset$; Q.push_back($\mathbf{v}$);
  while $(Q \neq \emptyset)$ {
    $\mathbf{v}$:= Q.pop_front();
    foreach $(\mathbf{e}' \in \mathcal{S}_1(\mathbf{v}) \setminus \mathcal{E}_h^D)$ {
      $\{\mathbf{v}'\} := \mathcal{S}_0(\mathbf{e}') \setminus \{\mathbf{v}\}$;
      if $(\mathbf{v}' \in \mathcal{L}_h)$ { $\mathcal{L}_h := \mathcal{L}_h \setminus \{\mathbf{v}'\}$; Q.push_back($\mathbf{v}'$); $\mathbf{v}'$.depth = $\mathbf{v}$.depth+1; }
      else { $\mathcal{E}_*^i := \mathcal{E}_*^i \cup \{\mathbf{e}'\}$; }
    }}}

FIG. 6. *Algorithm: Construction of trees* $T_{\mathfrak{G}}^i$ *and sets* $\mathcal{E}_*^i$.

Assume that π is not trivial. Then π is a 1-cycle and $\mathcal{S}_1(\pi)$ is completely contained in the tree $T_{\mathfrak{G}}^i$. In the subgraph $\mathfrak{G}_{|\mathcal{S}_1(\pi)}$ every vertex must have degree ≥ 2, which is implied by $\pi \in Z_1(\Gamma_h, \mathbb{Z})$. Hence, there is a circuit in $\mathfrak{G}_{|\mathcal{S}_1(\pi)}$. At the same time it is supposed to be contained in a tree, which, by contradiction, forces $\pi = 0$.

In sum, every cycle is homologous to a combination of the $\gamma_{\mathbf{e}}$. □

Be aware that we have found generators of $H_1(\Gamma_h, \mathbb{Z})$ that, by construction, have a geometric meaning as closed curves in Γ, too.

An outline of the combinatorial algorithms for the construction of the circuits $\mathbf{s}_{\mathbf{e}}$, $\mathbf{e} \in \mathcal{E}_*^i$, is given in Figures 5–7. The sets $\mathcal{D}_h$, $\mathcal{E}_h^D$, and $\mathcal{L}_h$, and the related queries can be efficiently realized through marking simplices of Γ_h. The individual steps of the algorithm in the case of a triangulated torus are illustrated in Figures 8–10.

If access to local incidence information is possible in constant time, the total computational effort will be proportional to $\sharp\mathcal{F}_h + (1 + \beta_1(\Gamma)) \cdot \sharp\mathcal{V}_h$. The algorithm produces ordered sequences $\mathbf{s}_{\mathbf{e}}$ of edges, $\mathbf{e} \in \mathcal{E}_*^i$, that represent circuits. Notice that the algorithm from Figure 5 permits us to identify the connected components of Γ_h without additional effort.

Remark. By a cheap algorithm we can always replace each $\gamma_{\mathbf{e}}$, $\mathbf{e} \in \mathcal{E}_*^i$, by a homologous edge cycle that does not contain two edges of the same surface triangle. For these "straightened" cycles, still denoted by $\Gamma_{\mathbf{e}}$, will hold

$$\sharp\mathcal{E}_h(\gamma_{\mathbf{e}}) \leq \sharp\mathcal{F}_h(\gamma_{\mathbf{e}}).$$

Often, this estimate is extremely pessimistic, as in most cases the cycles $\gamma_{\mathbf{e}}$, $\mathbf{e} \in \mathcal{E}_*^i$, will cover only a fraction of the edges in $\mathcal{E}_h$. The example of Figure 11 is typical. However, this cannot be guaranteed, in particular for very coarse surface meshes that

```
foreach (e ∈ E*^i) {
  list<Edge> s_e; s_e := ∅; {x, y} := S_0(e); s_e.push_back(e);
  do {
    while (x.depth > y.depth) {
      foreach (e' ∈ S_1(x) \ E_h^D) {
        {z} := S_0(e') \ {x};
        if (z.depth < x.depth)   { s_e.push_back(e'); x ← z; break; }}}
    while (x.depth ≤ y.depth and x ≠ y) {
      foreach (e' ∈ S_1(y) \ E_h^D) {
        {z} := S_0(e') \ {y};
        if (z.depth ≤ y.depth)   { s_e.push_front(e'); y ← z; break; }}}
  }
  while (x ≠ y); }
```

FIG. 7. *Algorithm: Construction of circuits* $\mathbf{s_e}$, $\mathbf{e} \in \mathcal{E}^i_*$. *They are assembled in STL-list-containers* [21, *section* 17.2.2].

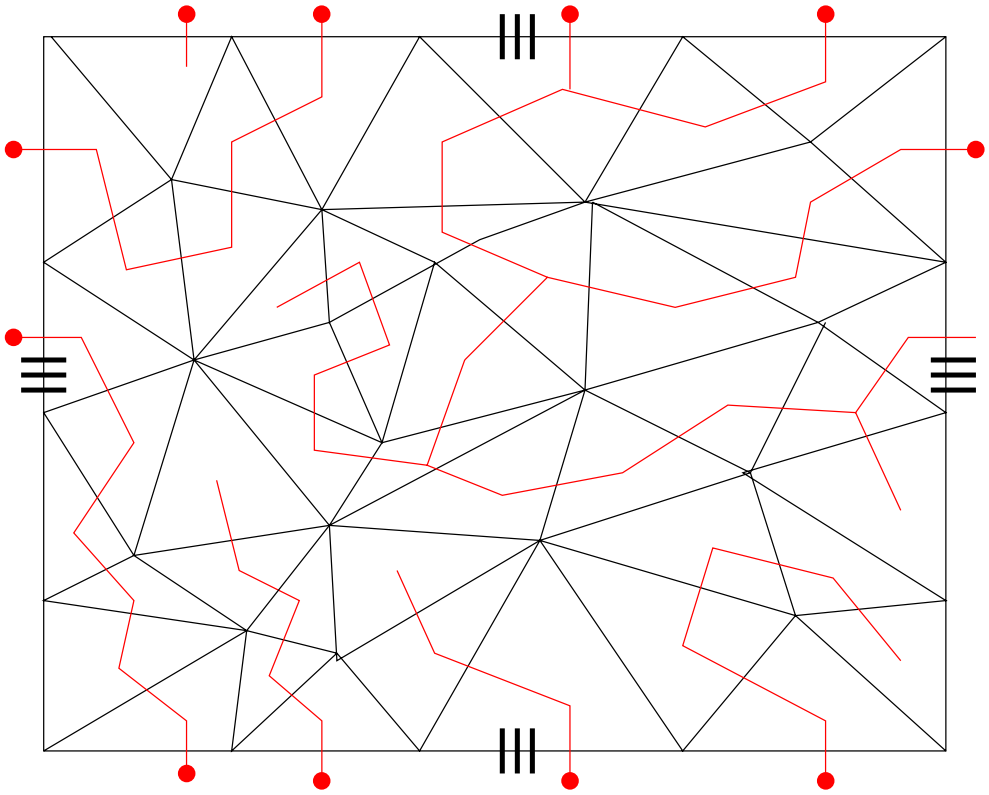


FIG. 8. *Tree* $T_{\mathfrak{D}}$ *in the case of a triangulated torus, which is represented by identifying opposite sides of a rectangle. Output of the algorithm from Figure* 5.

barely resolve the topology of Γ.

4. Identification of relevant cycles. As pointed out in the first section, finding generators of $H_1(\Gamma_h, \mathbb{Z})$ that bound relative to Ω' involves geometric considerations. This forces us to resort to concepts of homology theory on subsets of the

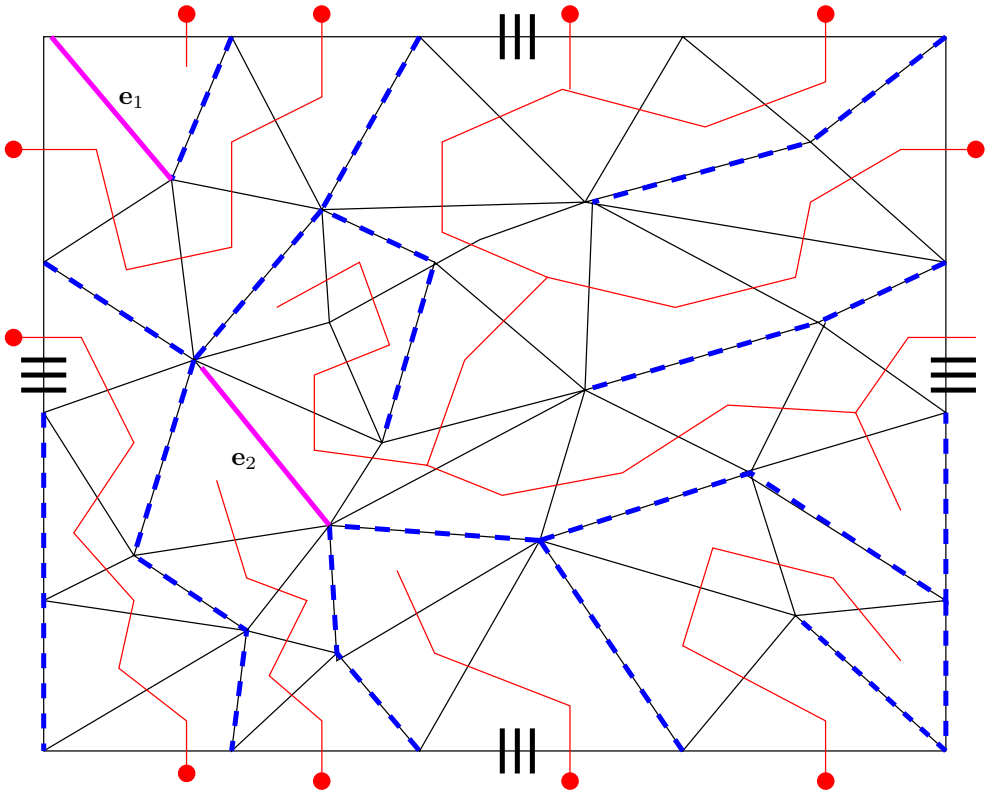


FIG. 9. *Construction of* $T_{\mathfrak{G}}$ (*dashed lines*) *on the torus according to the algorithm from Figure* 6, *yielding* $\mathcal{E}^i_* := \{\mathbf{e}_1, \mathbf{e}_2\}$.

Euclidean space $\mathbb{R}^3$. For a domain $D \subset \mathbb{R}^3$ we adopt the notation $Z_1(D,\mathbb{Z})$ for the group of 1-cycles, i.e., oriented closed curves in D. An element of $Z_1(\Gamma_h,\mathbb{Z})$ is said to bound relative to Ω (or Ω') if it can be represented by an oriented closed edge curve on Γ and if it is the boundary of an orientable 2-surface in Ω (or Ω').

The linking number (cf. [20, section 5.2], [4, Appendix I]) will be a crucial device.

DEFINITION 4.1. *Given two piecewise differentiable disjoint* 1*-cycles* $\gamma, \gamma' \in Z_1(\mathbb{R}^3,\mathbb{Z})$ *we define their* relative linking number *by*

$$L(\gamma,\gamma') := -\int_\gamma \int_{\gamma'} \operatorname{grad}_{\mathbf{y}} G(\mathbf{x},\mathbf{y}) \cdot (d\vec{s}(\mathbf{x}) \times d\vec{s}(\mathbf{y})), \quad G(\mathbf{x},\mathbf{y}) := \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|}.$$

Obviously $L(\gamma,\gamma') = L(\gamma',\gamma)$. Moreover, $L(\gamma,\gamma') \in \mathbb{Z}$. The linking number can also be expressed as $L(\gamma,\gamma') = \int_\gamma \mathbf{H}_{\gamma'}$ where $\mathbf{H}_{\gamma'}$ is a 1-form defined on $\mathbb{R}^3 \setminus \gamma'$ by

$$\langle \mathbf{H}_{\gamma'}(\mathbf{x}), \mathbf{v} \rangle := -\int_{\gamma'} \operatorname{grad}_{\mathbf{y}} G(\mathbf{x},\mathbf{y}) \cdot (\mathbf{v} \times d\vec{s}(\mathbf{y})), \qquad \mathbf{v} \in \mathbb{R}^3.$$

Straightforward computations show that $\mathbf{H}_{\gamma'}$ is closed, as the **curl** of its vector proxy vanishes on $\mathbb{R}^3 \setminus \gamma'$. If two 1-cycles $\gamma_1, \gamma_2 \in Z_1(\mathbb{R}^3 \setminus \gamma', \mathbb{Z})$ are homologous, i.e., the boundary of an oriented surface S, then, by Stokes' theorem,

$$L(\gamma_1,\gamma') - L(\gamma_2,\gamma') = \int_{\gamma_1-\gamma_2} \mathbf{H}_{\gamma'} = \int_{\partial S} \mathbf{H}_{\gamma'} = \int_S d\mathbf{H}_{\gamma'} = 0. \tag{4.1}$$

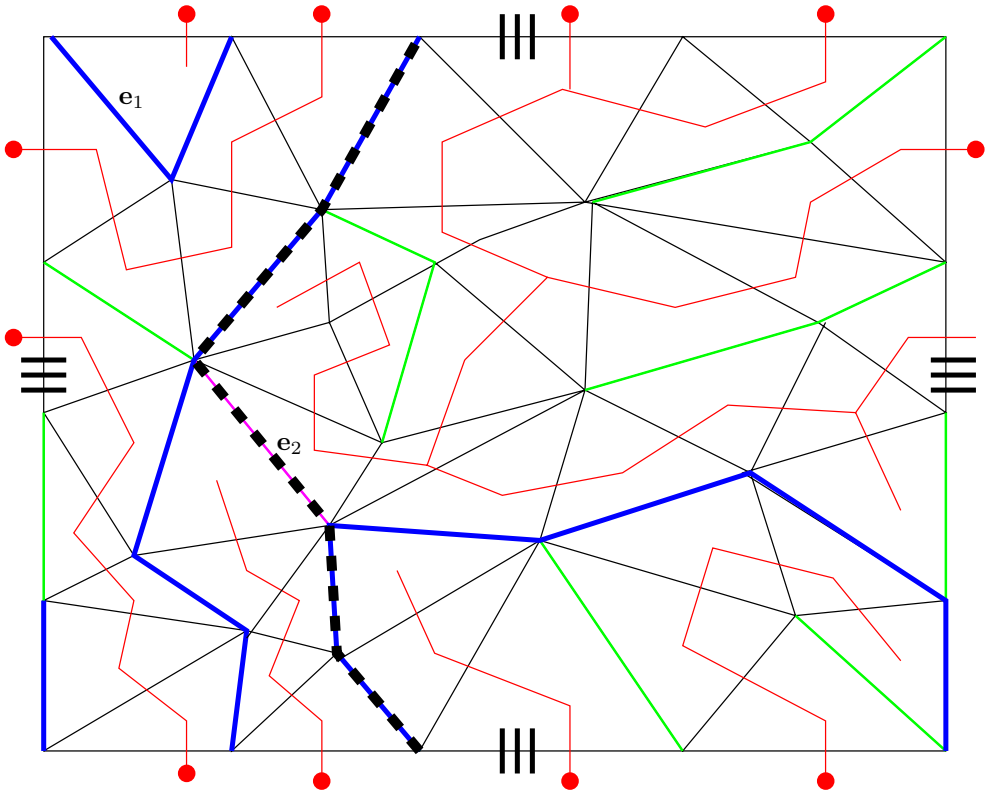


FIG. 10. *Two circuits* $\mathbf{s}_{\mathbf{e}_1}$ (*solid*) *and* $\mathbf{s}_{\mathbf{e}_2}$ (*dashed*) *on the triangulated torus as produced by the algorithm from Figure* 7.

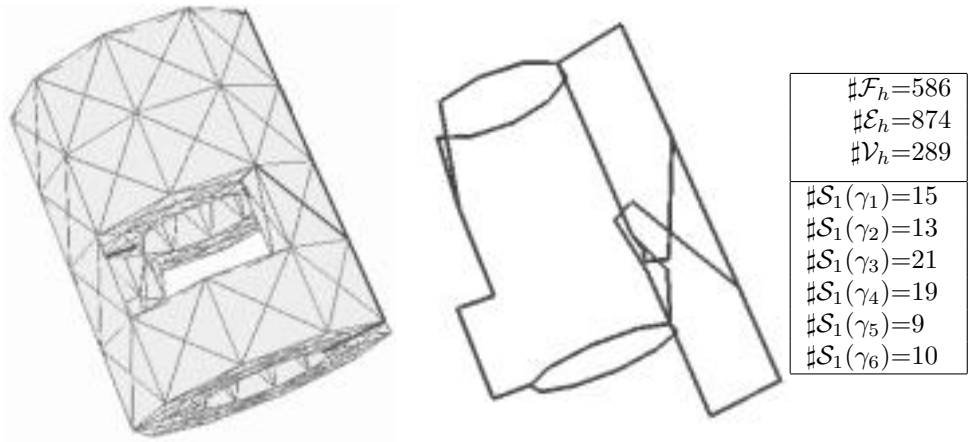


FIG. 11. *Surface mesh* (*left*) *and cycles* $\gamma_1, \ldots, \gamma_6$ (*right*) *in the case of a cyclinder with two intersecting holes drilled through it.*

Thus, the linking number defines a (bilinear) pairing

$$L : H_1(\Omega, \mathbb{Z}) \times H_1(\Omega', \mathbb{Z}) \mapsto \mathbb{Z} \tag{4.2}$$

between homology classes of 1-cycles in the interior and exterior of Ω. Recall that for a connected Γ an argument based on the Mayer–Vietoris exact sequence [3, section 2] shows

$$H_1(\Gamma,\mathbb{Z}) \cong H_1(\Omega',\mathbb{Z}) \oplus H_1(\Omega,\mathbb{Z}). \tag{4.3}$$

In addition, we know [20, section 6.6] that the first Betti numbers of Ω and its complement agree, i.e., $\beta_1(\Omega) = \beta_1(\Omega')$, and set $L := \beta_1(\Omega)$. Thanks to (4.3), this means that a basis of $H_1(\Gamma,\mathbb{Z})$ can be represented by L 1-cycles $\pi'_1,\dots,\pi'_L \in H_1(\Omega',\mathbb{Z})$ bounding relative to Ω and L 1-cycles $\pi_1,\dots,\pi_L \in H_1(\Omega,\mathbb{Z})$ bounding relative to Ω'. As exposed in [20, section 6.8], the pairing defined in (4.2) is nondegenerate in the sense that

$$L(\gamma,\gamma') = 0 \quad \forall \gamma' \in H_1(\Omega',\mathbb{Z}) \quad \Rightarrow \quad \gamma = 0 \quad \text{in } H_1(\Omega,\mathbb{Z}),$$

and vice versa. Thus, we can always choose the above 1-cycles such that

$$L(\pi_i,\pi'_j) = \delta_{ij}, \qquad 1 \le i,j \le L. \tag{4.4}$$

Retraction will give us 1-cycles $\in Z_1(\Gamma,\mathbb{Z})$ located on the boundary that are homologous to $\pi_1,\dots,\pi_L,\pi'_1,\dots,\pi'_L$ in $\bar\Omega$ and $\bar\Omega'$, respectively. We are calling them fundamental 1-cycles $\phi_1,\dots,\phi_L,\phi'_1,\dots,\phi'_l$, and note that still

$$L(\pi_i,\phi'_j) = \delta_{ij}, \qquad 1 \le i,j \le L.$$

Besides, the fundamental cycles provide a basis for $H_1(\Gamma,\mathbb{Z})$. By simple retraction to the 1-skeleton of Γ_h we also get a basis of $H_1(\Gamma_h,\mathbb{Z})$ for which we keep the notation $\{\phi_1,\dots,\phi_L,\phi'_1,\dots,\phi'_L\}$.

Linking numbers are well defined for disjoint 1-cycles only. Otherwise, the formula from Definition 4.1 will produce noninteger values. Therefore, we need a tool to disentangle surface 1-cycles.

DEFINITION 4.2. *For $\gamma \in Z_1(\Gamma_h,\mathbb{Z})$ we define the* submerged 1-cycle $\gamma\downarrow$ *as the homology class of γ in $H_1(\bar\Omega,\mathbb{Z})$.*

In the current setting, in which $\bar\Omega$ is homeomorphic to a compact set with smooth boundary, $H_1(\bar\Omega,\mathbb{Z}) = H_1(\Omega,\mathbb{Z})$ and $\gamma\downarrow$ can as well be regarded as an element of $H_1(\Omega,\mathbb{Z})$. We note the following consequences:

$$\phi_i\downarrow = \pi_i \text{ in } H_1(\Omega,\mathbb{Z}), \qquad \phi'_i\downarrow = 0, \qquad i = 1,\dots,L. \tag{4.5}$$

This is due to the fact that the exterior cycles π'_i are bounding relative to $\mathbb{R}^3$ and, consequently, ϕ'_i is bounding relative to $\bar\Omega$. Submersion is crucial for introducing the following "bilinear" pairing:

$$\langle\cdot,\cdot\rangle : \begin{cases} Z_1(\Gamma_h,\mathbb{Z}) \times Z_1(\Gamma_h,\mathbb{Z}) & \mapsto \quad \mathbb{Z} \\ (\eta_1,\eta_2) & \mapsto \quad L(\eta_2\downarrow,\eta_1). \end{cases} \tag{4.6}$$

Due to (4.1) it is well defined as $L(\beta,\eta) = 0$ for any 1-cycle $\beta \in Z_1(\Omega,\mathbb{Z})$ bounding relative to Ω and $\eta \in Z_1(\Gamma_h,\mathbb{Z})$. From (4.5) we conclude that $\langle\eta,\phi'_i\rangle = 0$ for all $i = 1,\dots,L$.

The point about this pairing is that it offers a criterion for telling the type of 1-cycles.

LEMMA 4.3. *$\gamma \in Z_1(\Gamma_h,\mathbb{Z})$ is bounding relative to Ω' if and only if $\langle\gamma,\phi_i\rangle = 0$ for all $i = 1,\dots,L$.*

Proof. The 1-cycle γ has a unique (modulo homology) representation with respect to the fundamental cycles

$$\gamma \sim \sum_{l=1}^{L} (\alpha_l \phi_l + \beta_l \phi'_l), \qquad \alpha_l, \beta_l \in \mathbb{Z}.$$

Recall that the 1-cycles $\phi_1, \ldots, \phi_L$, which have been "projected" onto the boundary of Ω from the interior, are bounding with respect to the exterior, whereas this is not true for $\phi'_1, \ldots, \phi'_L$. Thus, γ is bounding relative to Ω', if and only if it does not contain any contributions from $\phi'_1, \ldots, \phi'_L$, since the latter cannot be bounding relative to Ω' by definition. In short, γ is bounding relative to Ω' if and only if $\beta_l = 0$ for all $l = 1, \ldots, L$.

$$\begin{aligned}\langle \gamma, \phi_i \rangle = L(\phi_i \downarrow, \gamma) &= L\left(\phi_i \downarrow, \sum_{l=1}^{L} \alpha_l \phi_l + \beta_l \phi'_l\right) \\ &= \sum_{l=1}^{L} \alpha_l L(\phi_i \downarrow, \phi_l) + \sum_{l=1}^{L} \beta_l L(\phi_i \downarrow, \phi'_l) = \sum_{l=1}^{L} \beta_l \delta_{il} = \beta_i,\end{aligned}$$

as $L(\phi_i \downarrow, \phi_l) = 0$, because ϕ_l is bounding relative to $\bar{\Omega}'$, and $L(\phi_i \downarrow, \phi'_l) = L(\pi_i, \pi'_l) = \delta_{il}$ according to (4.4). □

The lemma seems to be of limited value, since we do not know the fundamental cycles explicitly. However, as $\langle \eta, \phi'_i \rangle = 0$ throughout, we need only a basis of $H_1(\Gamma_h, \mathbb{Z})$.

COROLLARY 4.4. *$\gamma \in Z_1(\Gamma_h, \mathbb{Z})$ is bounding relative to Ω' if and only if $\langle \gamma, \tau_i \rangle = 0$ for any basis $\{\tau_1, \ldots, \tau_{2L}\}$ of $H_1(\Gamma_h, \mathbb{Z})$.*

Proof. The basis property ensures that each ϕ_i, $i = 1, \ldots, L$, can be represented as a combination of τ_j (modulo homology). □

We have already got a basis at our disposal, namely the generators $\gamma_1, \ldots, \gamma_{2L}$ created by the graph algorithm of section 3. It goes without saying that any desired cycle can be obtained as an integral combination of these generators. So

$$\delta = \sum_{j=1}^{2L} \kappa_j \gamma_j \text{ bounding with respect to } \Omega' \quad \Leftrightarrow \quad \sum_{j=1}^{2L} \kappa_j \langle \gamma_j, \gamma_i \rangle = 0, \quad i = 1, \ldots, 2L.$$

In other words, $\delta \in Z_1(\Gamma_h, \mathbb{Z})$ is bounding with respect to $\bar{\Omega}'$ if and only if $\vec{\kappa} = (\kappa_1, \ldots, \kappa_{2L})^T \in \mathbb{Z}^{2L}$ satisfies

$$\vec{\kappa} \in \mathrm{Ker}(G^T), \qquad G := (\langle \gamma_j, \gamma_i \rangle)_{i,j=1}^{2L} \in \mathbb{Z}^{2L,2L}.$$

This suggests the following approach to the construction of the relevant surface 1-cycles:

1. Determine $\gamma_1, \ldots, \gamma_{2L}$ through spanning tree techniques (see section 3).
2. Compute the matrix G by evaluation of the pairing $\langle \cdot, \cdot \rangle$ (see section 5).
3. Use Gaussian elimination with full pivoting in $\mathbb{Z}$ to obtain an integral basis for $\mathrm{Ker}(G^T)$. Every basis vector will define a combination of $\gamma_1, \ldots, \gamma_{2L}$ that provides a relevant cycle.

The independence of the cycles thus obtained is clear from the linear independence of the basis vectors of $\mathrm{Ker}(G^T)$. However, do we get all L possible nonbounding surface

1-cycles (up to homology) that are bounding with respect to Ω'? The next result gives an affirmative answer.

THEOREM 4.5.

$$\dim \operatorname{Ker}(G^T) = L.$$

Proof. Start with a representation by means of fundamental cycles

$$\gamma_i \sim \sum_{l=1}^{L} (\alpha_{il}\phi_l + \beta_{il}\phi'_l), \qquad \alpha_{il}, \beta_{il} \in \mathbb{Z}, \quad i = 1 \ldots, 2L. \tag{4.7}$$

Observe that for $i, j = 1, \ldots, 2L$

$$\begin{aligned}\langle \gamma_i, \gamma_j \rangle &= \sum_{l,k=1}^{L} (\alpha_{il}\alpha_{jk} \langle \phi_l, \phi_k \rangle + \alpha_{il}\beta_{jk} \langle \phi_l, \phi'_k \rangle + \beta_{il}\alpha_{jk} \langle \phi'_l, \phi_k \rangle + \beta_{il}\beta_{jk} \langle \phi'_l, \phi'_k \rangle) \\ &= \sum_{l=1}^{L}\sum_{k=1}^{L} \beta_{il}\alpha_{jk} \langle \phi'_l, \phi_k \rangle = \sum_{l=1}^{L} \beta_{il}\alpha_{jl},\end{aligned}$$

as $\langle \phi_l, \phi_k \rangle = \langle \phi_l, \phi'_k \rangle = \langle \phi'_l, \phi'_k \rangle = 0$ and $\langle \phi'_l, \phi_k \rangle = \delta_{lk}$. In short, we may write

$$G = B \cdot A^T, \quad A = (\alpha_{il}) \in \mathbb{Z}^{2L,L}, \quad B = (\beta_{il}) \in \mathbb{Z}^{2L,L}.$$

As both $\phi_1, \ldots, \phi_l, \phi'_1, \ldots, \phi'_L$ and $\gamma_1, \ldots, \gamma_{2L}$ are bases of $H_1(\Gamma_h, \mathbb{Z})$, the matrix $(A|B) \in \mathbb{Z}^{2L,2L}$ has full rank. This also holds for A and B: $\operatorname{rank}(A) = \operatorname{rank}(B) = L$. From this we infer $\operatorname{rank}(G) = L$, which means $\dim \operatorname{Ker}(G) = L$. ∎

5. Evaluation of pairing. In order to obtain the matrix G, we have to provide an algorithm for computing representatives of the submerged cycles $\gamma_i\downarrow$, $i = 1, \ldots, 2L$. It turns out to be most efficient to split the task into two parts:

1. The computation of shifted surface 1-cycles $\hat{\gamma}_i$, $i = 1, \ldots, 2L$, that clear all vertices of Γ_h and satisfy $\hat{\gamma}_i = \gamma_i$ in $H_1(\Gamma, \mathbb{Z})$. This resembles the barycentric perturbation proposed in [7, section 4.1].
2. Replacing every γ_i by a 1-cycle $\check{\gamma}_i$ such that $\check{\gamma}_i = \gamma_i$ in $H_1(\bar{\Omega}, \mathbb{Z})$, $i = 1, \ldots, 2L$, and $\check{\gamma}_i \cap \hat{\gamma}_j = \emptyset$ for all i, j.

As the linking numbers are well defined on homology classes, by construction

$$\langle \gamma_i, \gamma_j \rangle = L(\check{\gamma}_j, \hat{\gamma}_i). \tag{5.1}$$

For the discussion of the algorithmic details pick some $i \in \{1, \ldots, 2L\}$ and set $\gamma := \gamma_i$. Recall that γ is a 1-chain $\in Z_1(\Gamma_h, \mathbb{Z})$ with weights $\in \{-1, 0, 1\}$ and that it is naturally associated with a closed non-self-intersecting edge curve in Γ_h. Denote by $\mathcal{V}_h(\gamma)$ and $\mathcal{E}_h(\gamma)$ the set of vertices and edges "contained" in γ. As γ has an orientation we can number the edges in $\mathcal{E}_h(\gamma) := \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N\}$, $N \in \mathbb{N}$. Then, to each vertex $\mathbf{v} \in \mathcal{V}_h(\gamma)$ assign the number of that edge $\mathbf{e} \in \mathcal{E}_h(\gamma)$ for which $\iota(\mathbf{v}, \mathbf{e}) \cdot \gamma(\mathbf{e}) = -1$. By the construction of γ this is well defined (cf. Lemma 3.4) and furnishes a numbering of the vertices of γ: $\mathcal{V}_h(\gamma) = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$. By setting

$$\mathbf{t}_i := \text{triangle } \mathbf{t} \in \mathcal{F}_h : \ \iota(\mathbf{e}_i, \mathbf{t}) \cdot \gamma(\mathbf{e}_i) = 1, \qquad i = 1, \ldots, N,$$

we fix a sequence $(\mathbf{t}_1, \ldots, \mathbf{t}_N)$ of triangles. Be aware that one triangle might occur twice in this sequence. The triangles are located on one side of γ, because all triangles in $\mathcal{F}_h$ have the same orientation inherited from Γ (see Figure 12).

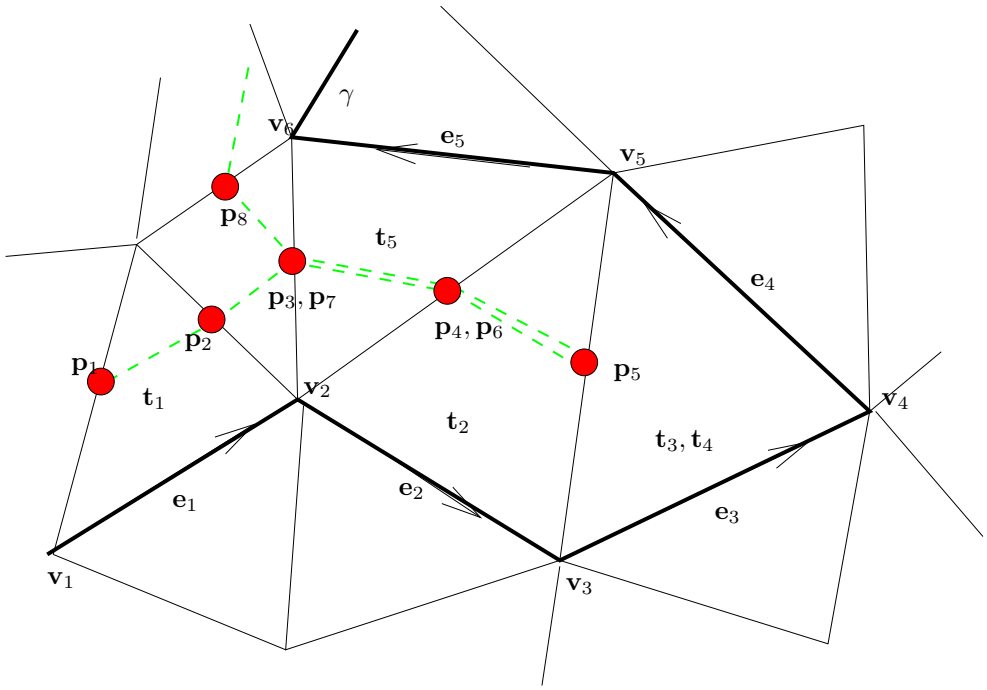


FIG. 12. *Numbering of edges, vertices, and triangles in connection with a path* γ. *Shifted path as polygon* $(\mathbf{p}_1, \mathbf{p}_2, \dots)$.

```
list<Point> U:= ∅;
for (i = 1; i ≤ N; i++) {
  U.push_back(v_i);
  T := tetrahedron adjacent to v_i and v_{i+1} ;
  U.push_back(center of gravity of T);}
```

FIG. 13. *Submerging of a cycle* $(\mathbf{v}_1, \dots, \mathbf{v}_N)$. *Output is polygon* U.

The 1-cycles $\check{\gamma}$ and $\hat{\gamma}$ will be computed as closed oriented polygons $U := (\mathbf{u}_1, \dots, \mathbf{u}_{2N})$ and $P := (\mathbf{p}_1, \dots, \mathbf{p}_M)$, $M \in \mathbb{N}$, given by sequences of points $\mathbf{p}_i, \mathbf{u}_i \in \mathbb{R}^3$ (which are supposed to be connected by straight lines). This is sufficient, since we need only U and P for the evaluation of path integrals. Note that, throughout, all sequences (of simplices and points) will be assumed to be cyclic.

We get the polygon U by starting with the sequence of vertices $(\mathbf{v}_1, \dots, \mathbf{v}_N)$. Then we insert the center of gravity of a tetrahedron adjacent to two consecutive vertices $\mathbf{v}_i$, $\mathbf{v}_{i+1}$ after $\mathbf{v}_i$, $i = 1, \dots, N$. The simple algorithm is shown in Figure 13. Only at this stage do we make use of the volume mesh Ω_h!

The polygon P defining the shifted cycle $\hat{\gamma}$ runs through midpoints of certain edges in $\bigcup_{\mathbf{v} \in \mathcal{V}_h(\gamma)} \mathcal{S}_1(\mathbf{v}) \setminus \mathcal{E}_h(\gamma)$. It is constructed by the algorithm sketched in Figure 14. Obviously, P provides a 1-cycle $\sim \gamma$ in $H_1(\Gamma, \mathbb{Z})$. Formally, this can be proved by considering a refined surface mesh $\tilde{\Gamma}_h$ generated by subdividing each triangle of Γ_h into four congruent smaller triangles. Viewing γ as an element of $Z_1(\tilde{\Gamma}_h, \mathbb{Z})$ the shifted cycle $\hat{\gamma}$ is seen to emerge by successively adding the boundaries of triangles of $\tilde{\Gamma}_h$ to γ. We skip the tedious details. We remark that the "dead end" conspicuous in Figure 12

```
list<Point> P := ∅;
for (i = 1, i ≤ N, i++) {
  P.push_back(midpoint of that edge of t_i adjacent to v_i);
  {e} := S_1(t_i) \ S_1(v_i); t := t_i;
  while (e ∉ S_1(t_{i+1})) {
    P.push_back(midpoint of e); {t} ← S_2(e)\{t}; {e} ← (S_1(t)∩S_1(v_{i+1}))\{e}; }}
```

FIG. 14. *Construction of an oriented polygon P representing a shifted cycle. Note that we identify $\mathbf{t}_{N+1} := \mathbf{t}_1$. See also Figure* 12.

cannot occur with straightened 1-cycles (cf. the remark at the end of section 3).

It is straightforward that the computational effort it takes to get U and P is proportional to $N + \sharp\mathcal{F}_h$. Moreover, a bounded shape-regularity measure ρ from (2.2) implies that only a small number of edges can be incident on a vertex. Assuming a certain shape regularity we end up with a complexity $O(N)$ for the construction of P and U.

Next, we have to compute the relative linking number of a submerged 1-cycle $\check{\gamma}$ and a shifted 1-cycle $\hat{\gamma}$. They are supplied as closed polygons $U = (\mathbf{u}_1, \ldots, \mathbf{u}_{2N})$, $N \in \mathbb{N}$, and $P = (\mathbf{p}_1, \ldots, \mathbf{p}_M)$, $M \in \mathbb{N}$. Recalling Definition 4.1 and (5.1), we have to compute the linking numbers

$$\begin{aligned} L(\check{\gamma}, \hat{\gamma}) &= -\sum_{i=1}^{2N} \sum_{j=1}^{M} \int_{[\mathbf{u}_i, \mathbf{u}_{i+1}]} \int_{[\mathbf{p}_i, \mathbf{p}_{i+1}]} \operatorname{grad}_{\mathbf{y}} G(\mathbf{x}, \mathbf{y}) \cdot (d\vec{s}(\mathbf{x}) \times d\vec{s}(\mathbf{y})) \\ &= -\sum_{i=1}^{2N} \sum_{j=1}^{M} \int_0^1 \int_0^1 \operatorname{grad}_{\mathbf{y}} G(\mathbf{x}_j(\sigma), \mathbf{y}_i(\tau)) \cdot \mathbf{z}_{ij} \, d\sigma d\tau, \end{aligned}$$

where $\mathbf{y}_i(\tau) := \mathbf{u}_i + \tau(\mathbf{u}_{i+1} - \mathbf{u}_i)$, $\mathbf{x}_j(\sigma) := \mathbf{p}_j + \sigma(\mathbf{p}_{j+1} - \mathbf{p}_j)$, $\mathbf{z}_{i,j} := (\mathbf{p}_{j+1} - \mathbf{p}_j) \times (\mathbf{u}_{i+1} - \mathbf{u}_i)$. Analytic expressions are available for the inner integrals

$$f_{ij}(\tau) := \int_0^1 \operatorname{grad}_{\mathbf{y}} G(\mathbf{x}_j(\sigma), \mathbf{y}_i(\tau)) \cdot \mathbf{z}_{ij} \, d\sigma.$$

For the outer integrals we have to resort to numerical quadrature. Here, it is important to take into account the singular behavior of the kernel $G(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} \to \mathbf{y}$. It entails an *adaptive* approach to quadrature: We use a Gauß–Legendre quadrature formula of order $2n$ (i.e., with n nodes), $n \in \mathbb{N}$, on each interval of an equidistant subdivision of $[0;1]$ into k_{ij}, $k_{ij} \in \mathbb{N}$, parts. Adaptivity will be achieved by adjusting k_{ij} depending on the relative position of the line segments $[\mathbf{u}_i, \mathbf{u}_{i+1}]$ and $[\mathbf{p}_i, \mathbf{p}_{i+1}]$.

The Gauß–Legendre formulas give us an approximation $\tilde{I}_{ij}$ for $\int_0^1 f_{ij}(\tau)\, d\tau$. Standard estimates of the quadrature error [6, Thm. 9.5] yield

$$\epsilon_{ij} := \left| \int_0^1 f_{ij}(\tau)\, d\tau - \tilde{I}_{ij} \right| \le \frac{1}{2n-1} \left(\frac{1}{2k_{ij}} \right)^{2n} \sup_{\xi \in [0;1]} \left| \frac{d^{2n} f_{ij}}{d\tau^{2n}}(\xi) \right|. \tag{5.2}$$

Using

$$\frac{d^m f_{ij}}{d\tau^m}(\tau_0) = \int_0^1 D_{\mathbf{y}}^{m+1} G(\mathbf{x}_j(\sigma), \mathbf{y}_i(\tau_0)) (\underbrace{\mathbf{u}_{i+1} - \mathbf{u}_i, \ldots, \mathbf{u}_{i+1} - \mathbf{u}_i}_{m \text{ times}}, \mathbf{z}_{ij})\, d\sigma$$

and, for $\mathbf{x} \neq \mathbf{y}$, $\mathbf{w}_l \in \mathbb{R}^3$,

$$D_{\mathbf{y}}^{m+1} G(\mathbf{x}, \mathbf{y})(\mathbf{w}_1, \dots, \mathbf{w}_{m+1}) = (-1)^{m+1}(m+1)! \, \frac{1}{|\mathbf{x}-\mathbf{y}|^{m+2}} \, w_1^r \cdot \dots \cdot w_{m+1}^r,$$

where $w_l^r := |\mathbf{x}-\mathbf{y}|^{-1}(\mathbf{x}-\mathbf{y}) \cdot \mathbf{w}_l$, we can estimate

$$\epsilon_{ij} \leq |\mathbf{u}_{i+1} - \mathbf{u}_i|^{m+1} |\mathbf{p}_{j+1} - \mathbf{p}_j| (m+1)! \, \frac{1}{d_{ij}^{m+2}},$$

with the distance

$$d_{ij} := \inf\{|\mathbf{x}-\mathbf{y}|, \, \mathbf{x} \in [\mathbf{u}_i, \mathbf{u}_{i+1}], \, \mathbf{y} \in [\mathbf{p}_j, \mathbf{p}_{j+1}]\}.$$

Finally,

$$\epsilon_{ij} \leq \frac{(2n+1)!}{2n-1} \left(\frac{1}{2k_{ij}} \right)^{2n} |\mathbf{u}_{i+1} - \mathbf{u}_i|^{2n+1} |\mathbf{p}_{j+1} - \mathbf{p}_j| \, \frac{1}{d_{ij}^{2n+2}}.$$

The exact linking numbers are integers. We must be able to recover them despite the impact of quadrature errors. This is guaranteed if $2NM\epsilon_{ij} \leq \theta$ for some tolerance $\theta < \frac{1}{2}$. We get the following a priori criterion for choosing k_{ij}:

$$k_{ij} \geq \frac{1}{2} \left(\frac{(2n+1)!}{2n-1} \, \frac{|\mathbf{u}_{i+1} - \mathbf{u}_i|^{2n+1} |\mathbf{p}_{j+1} - \mathbf{p}_j|}{d_{ij}^{2n+2}} \, \theta^{-1} 2NM \right)^{\frac{1}{2n}}.$$

We will gauge the costs of evaluating $L(\check{\gamma}, \hat{\gamma})$ for a q-quasi-uniform, $0 < q \leq 1$, mesh Ω_h of meshwidth $h > 0$, and shape-regularity measure $\rho > 0$. Then $|\mathbf{u}_{i+1} - \mathbf{u}_i| \leq h$ and $|\mathbf{p}_{j+1} - \mathbf{p}_j| \leq h$. We find that $k_{ij} = 1$ is sufficient if

$$d_{ij} \geq h S_n, \quad S_n := \left(\frac{(2n+1)!}{2n-1} \, \frac{2NM}{\theta} \right)^{\frac{1}{2(n+1)}} \leq (2n+1) \left(\frac{2NM}{\theta} \right)^{\frac{1}{2(n+1)}}.$$

The geometric restrictions on Ω_h also force two edges of U and P to be farther apart than $C_2 h$ with a geometric constant $C_2 = C_2(\rho)$, $1 > C_2 > 0$. Fix j and take into account that due to shape regularity there are fewer than

$$\frac{\pi(\frac{4}{3}d^3 + |\mathbf{p}_j - \mathbf{p}_{j+1}| d^2)}{\min_{T \in \Omega_h} \mathrm{Vol}(T)} \leq \left(\frac{\rho}{qh} \right)^3 d^2 (\tfrac{4}{3}d + h) \leq \left(\frac{\rho}{qh} \right)^3 d^3 (\tfrac{4}{3} + C_2^{-1}) = C(\rho, q) \left(\frac{d}{h} \right)^3$$

tetrahedra in a d-neighborhood, $d > C_2 h$, of $[\mathbf{p}_j, \mathbf{p}_{j+1}]$. This is also an upper bound for (half) the number of edges of U contained in that neighborhood. Hence, the total number of subintervals required for a sufficiently accurate quadrature on the "near edges" (edges $[\mathbf{u}_i, \mathbf{u}_{i+1}]$ with $d_{ij} < h S_n$) is as follows:

$$\begin{aligned} K_j &\leq 2C(\rho, q) \int_{C_2 h}^{S_n h} \frac{d^3}{h^3} \left(\left(\frac{(2n+1)!}{2n-1} \, \frac{h^{2n+2}}{d^{2n+2}} \, \theta^{-1} 2NM \right)^{\frac{1}{2n}} + 1 \right) dd \\ &\leq 2C(\rho, q) \int_{C_2}^{S_n} s^{2-\frac{1}{n}}(\dots) + s^3 \, ds \leq 4C(\rho, q) \left(S_n^4 - C_2^4 \right). \end{aligned}$$

Taking into account that also $N, M \le C(\rho, q)h^{-2}$, we end up with

$$S_n \le C(\rho, q, \theta)(2n+1)h^{-2/n} \quad \Rightarrow \quad K_j \le C(\rho, q, \theta)n^4 h^{-8/n}.$$

This translates into an asymptotic total effort (measured in terms of quadrature nodes) of $O(n^4 h^{-(2+8/n)})$ for the treatment of all "near edges." The "far edges" take at most $2NM$ evaluations. This teaches us that we should choose $n \ge 4$ in order to balance the effort required to treat "near" and "far" pairs of line segments. The bottom line is that assuming a certain shape regularity and quasi-uniformity of Ω_h, at worst we encounter total computational costs of the order $O((\sharp\mathcal{F}_h)^2) \cdot \beta_1(\Gamma_h)^2$. As pointed out before, in most practical computations the costs will be much lower.

Remark. In [7, section 4.1] the linking numbers are computed by projecting the edge cycles onto a plane and counting oriented intersections of the projected curves. This obviously allows us to compute the relative linking number of U and P with an effort proportional to $N \cdot M$, because all pairs of projected edges have to be examined. Yet, carrying out the projection in a stable fashion might be challenging in finite precision arithmetic.

REFERENCES

[1] C. Bonnington and C. Little, *The Foundations of Topological Graph Theory*, Springer-Verlag, New York, 1995.

[2] A. Bossavit, *Computational Electromagnetism. Variational Formulation, Complementarity, Edge Elements*, Electromagnetism 2, Academic Press, San Diego, CA, 1998.

[3] R. Bott and L. Tu, *Differential Forms in Algebraic Topology*, Springer-Verlag, New York, 1982.

[4] M. Brown, *Scalar potentials in multiply connected regions*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 665–680.

[5] P. Ciarlet, *The Finite Element Method for Elliptic Problems*, Stud. Math. Appl. 4, North-Holland, Amsterdam, 1978.

[6] P. Deuflhard and A. Hohmann, *Numerische Mathematik. Eine algorithmisch orientierte Einführung*, DeGruyter, Berlin, 1991.

[7] T. Dey and S. Guha, *Computing homology groups of simplicial complexes in* $\mathbb{R}^3$, J. ACM, 45 (1998), pp. 266–287.

[8] F. Dubois, *Discrete vector potential representation of a divergence-free vector field in three dimensional domains: Numerical analysis of a model problem*, SIAM J. Numer. Anal., 27 (1990), pp. 1103–1141.

[9] R. Fritsch and R. Piccini, *Cellular Structures in Topology*, Cambridge Stud. Adv. Math. 19, Cambridge University Press, Cambridge, UK, 1990.

[10] P. Grisvard, *Elliptic Problems in Nonsmooth Domains*, Pitman, Boston, 1985.

[11] P. Gross, *Efficient Finite Element-Based Algorithms for Topological Aspects of* 3*-Dimensional Magnetoquasistatic Problems*, Ph.D. thesis, College of Engineering, Boston University, Boston, MA, 1998.

[12] P. Gross and P. Kotiuga, *Finite element-based algorithms to make cuts for magnetic scalar potentials: Topological constraints and computational complexity*, in Geometric Methods for Computational Electromagnetics, PIER, Vol. 32, F. Teixeira, ed., EMW, Cambridge, MA, 2001, pp. 207–245.

[13] R. Hiptmair, *Symmetric coupling for eddy current problems*, SIAM J. Numer. Anal., 40 (2002), pp. 41–65.

[14] L. Kettunen, K. Forsman, and A. Bossavit, *Formulation of the eddy current problems in multiply connected regions in terms of* $\mathbf{h}$, Internat. J. Numer. Methods Engrg., 41 (1998), pp. 935–954.

[15] P. Kotiuga, *On making cuts for magnetic scalar potentials in multiply connected regions*, J. Appl. Phys., 61 (1987), pp. 3916–3918.

[16] P. Kotiuga, *Toward an algorithm to make cuts for magnetic scalar potentials in finite element meshes*, J. Appl. Phys., 63 (1988), pp. 3357–3359.

[17] P. Kotiuga, *An algorithm to make cuts for magnetic scalar potentials in tetrahedral meshes based on the finite element method*, IEEE Trans. Magnetics, 25 (1989), pp. 4129–4131.

[18] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust, *Computing a canonical polygonal schema of an orientable triangulated surface*, in Proceedings of the Seventeenth Annual ACM Symposium on Computational Geometry, ACM, New York, 2001, pp. 80–89.

[19] Z. Ren and A. Razek, *Boundary edge elements and spanning tree techniques in three-dimensional electromagnetic field computation*, Internat. J. Numer. Methods Engrg., 36 (1993), pp. 2877–2893.

[20] A. Schwarz, *Topology for Physicists*, Grundlehren Math. Wiss. 308, Springer-Verlag, Berlin, 1994.

[21] B. Stroustrup, *The C++ Programming Language*, 3rd ed., Addison Wesley Longman, Reading, MA, 1997.

[22] T. Tarhasaari and L. Kettunen, *Topological approach to computational electromagnetism*, in Geometric Methods for Computational Electromagnetics, PIER, Vol. 32, F. Teixeira, ed., EMW, Cambridge, MA, 2001, pp. 189–206.

[23] G. Vegter and C. Yap, *Computational complexity of combinatorial surfaces*, in Proceedings of the Sixth ACM Annual Symposium on Computational Geometry, ACM, New York, 1990, pp. 102–111.

[24] H. Whitney, *Geometric Integration Theory*, Princeton University Press, Princeton, NJ, 1957.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1424–1437

A THEOREM ON SENSITIVITY AND APPLICATIONS IN PRIVATE COMPUTATION*

ANNA GÁL† AND ADI ROSÉN‡

Abstract. In this paper we prove a theorem that gives an (almost) tight upper bound on the sensitivity of a multiple-output Boolean function in terms of the sensitivity of its coordinates and the size of the range of the function. We apply this theorem to get improved lower bounds on the time (number of rounds) to compute Boolean functions by private protocols. These bounds are given in terms of the sensitivity of the function being computed and the amount of randomness used by the private protocol. These lower bounds are tight (up to constant factors) for the case of the `xor` function and together with the results in [E. Kushilevitz and A. Rosén, *SIAM J. Discrete Math.*, 11 (1998), pp. 61–80.] establish a tight (up to constant factors) tradeoff between randomness and time in private computation.

Key words. sensitivity, private computation, randomness, lower bounds

AMS subject classifications. 68R05, 94A60, 68M10

PII. S0097539701385296

1. Introduction.

An important characteristic of a Boolean function is its sensitivity. Informally, the sensitivity of a function is the maximum number of input variables such that changing the value of just one variable at a time changes the value of the function as well. The sensitivity of Boolean functions and its relation to other complexity measures have been studied extensively. A number of important results have been achieved via arguments about sensitivity. For example, lower bounds on the time required for CREW PRAM computation [33, 10, 28] and lower bounds on the size of reliable circuits from unreliable gates [11, 30, 31, 14, 15] were given in terms of the sensitivity of the function being computed. A generalization of sensitivity, block sensitivity, was defined by Nisan [28]. Studying block sensitivity revealed that sensitivity provides lower bounds for several other measures, including Boolean decision tree complexity [28] and the degree of real polynomials representing Boolean functions [29]. The relation between sensitivity and block sensitivity has been studied in a number of papers [28, 17, 32, 20]. In several settings, average sensitivity is an important measure. It has been shown that the average sensitivity of a function is related to its Fourier coefficients [19] and that the average sensitivity of functions computable by constant depth circuits must be low [26]. Bounds on the sensitivity of various classes of functions were given in [34, 35].

In this paper we prove a theorem on the sensitivity of multiple-output Boolean functions. We give an almost tight upper bound on the sensitivity of such functions, in terms of the sensitivity of each coordinate, and the size of the range of the function.

*Received by the editors February 20, 2001; accepted for publication (in revised form) October 29, 2001; published electronically July 1, 2002. An early version of this paper appeared in *Proceedings of the* 31*st Annual ACM Symposium on Theory of Computing*, Atlanta, GA, 1999, pp. 348–357.
http://www.siam.org/journals/sicomp/31-5/38529.html

†Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 (panni@cs.utexas.edu). Part of this author's work was done while on leave at the Department of Computer Science of the University of Toronto and the Fields Institute and was partially supported by ITRC, an Ontario Centre of Excellence, and NSF CAREER Award CCR-9874862.

‡Department of Computer Science, Technion, Haifa 32000, Israel (adiro@cs.technion.ac.il). Part of this author's research was done while with the Department of Computer Science, University of Toronto.

More formally, we prove the following theorem:

Let $F : \{0,1\}^n \to \{0,1\}^m$ be an m-output Boolean function with coordinate functions $f_j : \{0,1\}^n \to \{0,1\}$ such that $s(f_j) \leq k$ for each $1 \leq j \leq m$, where $s(f)$ is the sensitivity of f. If the range of F contains D different values, then the sensitivity of F is at most $k \cdot 4(\log_2 D + 2)$.

Note that the restriction on the size of the range of F can be interpreted as a condition on the "correlation" between the coordinate functions f_j. Without this restriction, the number of possible values of an m-output Boolean function is 2^m. It is easy to see that $s(F) \leq km$ must always hold. Similarly, if we restrict the range of F to be a q-dimensional subcube of $\{0,1\}^m$, that is, we require that $m - q$ of the m coordinate functions are constants (and the other q coordinate functions can be arbitrary functions), then $s(F) \leq kq$ must hold. Our results show that even if the range of F is an *arbitrary subset* of size 2^q of $\{0,1\}^m$, the sensitivity of F cannot be much larger. Our bound is almost tight, as for q independent coordinates the sensitivity kq is achieved.

We use the above theorem to prove lower bounds in information-theoretic private computation. We believe, however, that the theorem is of independent interest and may find additional applications. Using the above theorem and the machinery of [25] we prove improved lower bounds on the number of rounds required to privately compute a Boolean function. The lower bound is given in terms of the sensitivity of the function being computed and the amount of randomness used by the protocol overall. For the case of the function `xor` (exclusive or) these lower bounds are tight, up to a small constant factor. A private protocol to compute a Boolean function f allows a number of players, each possessing a single input bit, to compute the value of the function f on their combined input in a way that no single player learns any "unnecessary" information (in particular, the inputs of the other players).[1] Private computation in this setting was first considered by Yao [36] and has been the subject of a considerable amount of research [1, 2, 4, 5, 7, 8, 9, 12, 13, 16, 22, 21, 23].

Using randomness any function can be computed privately if the number of players is at least three. On the other hand, for most functions (except very simple ones), randomness is necessary in order to compute the function privately. Randomness as a resource has been the subject of extensive research in the past decade. In the context of private computation, the main questions addressed about randomness as a resource have been the minimum number of random bits required for private computation of different functions and tradeoffs between the amount of randomness and the amount of time (i.e., number of rounds) required for the computation [5, 22, 25, 24, 6]. It is worthwhile to note that one can also characterize the class of functions computable by linear size circuits in terms of the amount of randomness required for their private computation [24]. The question of whether private computations in general can be carried out in constant number of rounds was addressed in [1, 3, 18].

A lower bound on the number of rounds required for the private computation of Boolean functions was given by Kushilevitz and Rosén [25]. They proved that it takes at least $\Omega(\log s(f)/d)$ rounds to privately compute a function f of sensitivity $s(f)$ using d random bits overall. They also gave protocols to compute the function `xor` that use a small number of random bits and at the same time are efficient in terms of rounds: for any $d \geq 2$, they provided a protocol to privately compute the

[1] In the literature, a more general notion of t-privacy is used, requiring that no coalition of t players learns extra information. Here we discuss the case of $t = 1$. See section 3 for a formal definition of private protocols.

xor of n bits, using $O(\log n/\log d) = O(\log s(\texttt{xor})/\log d)$ rounds and d random bits. However, the exact tradeoff between randomness and rounds was left as an open question. Using our theorem on sensitivity we prove that it takes $\Omega(\log s(f)/\log d)$ rounds to privately compute a Boolean function f using d random bits overall.

2. A theorem on sensitivity. In this section we prove a theorem that gives an upper bound on the sensitivity of a Boolean function $F : \{0,1\}^n \to \{0,1\}^m$, in terms of the sensitivity of the coordinate functions $f_j : \{0,1\}^n \to \{0,1\}$, and the size of the range of F.

DEFINITION 2.1 (sensitivity).

- *For a binary vector y, denote by $y^{(i)}$ the binary vector obtained from y by flipping the ith entry.*
- *A function f is* sensitive *to its ith variable on input y if $f(y) \neq f(y^{(i)})$.*
- *$s_y(f)$ is the number of variables to which the function f is sensitive on input y.*
- *The* sensitivity *of a function f is $s(f) = \max_y s_y(f)$.*
- *The* average sensitivity *of a function f is $as(f) = \frac{1}{2^n}\sum_{y\in\{0,1\}^n} s_y(f)$.*

Note that for a multiple-output Boolean function $F : \{0,1\}^n \to \{0,1\}^m$ the value of F on each input $x \in \{0,1\}^n$ is a binary vector of length m, and $F(x) \neq F(y)$ if and only if $f_j(x) \neq f_j(y)$ for at least one of the coordinate functions f_j, $1 \leq j \leq m$.

FACT 1. *Let $F : \{0,1\}^n \to \{0,1\}^m$ be an m-output Boolean function with coordinate functions $f_j : \{0,1\}^n \to \{0,1\}$ such that $s(f_j) \leq k$ for each $1 \leq j \leq m$. Then $s(F) \leq km$.*

Proof. The statement follows from the simple fact that $s(F) \leq \sum_{j=1}^m s(f_j)$. □

Note that this bound is tight: taking $f_j(x) = x_j$, we have $s(f_j) = 1 = k$ and $s(F) = m = km$.

The proof of Fact 1 shows that if we restrict the range of F to be a q-dimensional subcube of $\{0,1\}^m$, then $s(F) \leq kq$ must hold. In the following theorem we prove that the sensitivity of F cannot be much larger, even if the range of F is an *arbitrary subset* of size 2^q of $\{0,1\}^m$.

THEOREM 2.2. *Let $F : \{0,1\}^n \to \{0,1\}^m$ be an m-output Boolean function with coordinate functions $f_j : \{0,1\}^n \to \{0,1\}$ such that $s(f_j) \leq k$ for each $1 \leq j \leq m$. If the range of F contains D different values, then the sensitivity of F is at most $k \cdot 4(\log_2 D + 2)$.*

We observe that this bound is almost tight. Consider again the functions $f_j(x) = x_j$, each of sensitivity $k = 1$. In this case the sensitivity of F is $k \cdot \log_2 D$, as $D = 2^n = 2^m$. Note that our bound holds for every k and D, independent of the values of n and m.

In the following the function F and the functions f_j are as defined in the above theorem. In particular, we always assume that $s(f_j) \leq k$.

For our argument it is convenient to use the following definition.

DEFINITION 2.3 (sensitivity restricted to a set). *For $F(x) = (f_1(x), \ldots, f_m(x))$ and $x \in \{0,1\}^n$, let $S(x)$ denote the set of vectors $y \in \{0,1\}^n$ that are at Hamming distance 1 from x and for which $F(y) \neq F(x)$. Let $U \subseteq \{0,1\}^n$. We use the notation $S(x,U) = S(x) \cap U$ and say that $|S(x,U)|$ is the sensitivity of F on x restricted to the set U.*

Notation. Let $V \subseteq \{0,1\}^n$ be a subset of input vectors. We partition the set V into levels with respect to a fixed vector $v_0 \in V$. The set V_i is the set of vectors in V that are at Hamming distance i from v_0 and is called the ith level of V with respect to v_0. Note that $V_0 = \{v_0\}$. The notation used for the levels V_i does not

indicate their dependence on the choice of v_0. This should nevertheless be clear from the context. For $x \in V_i$ we use the notation $\sigma_{V,v_0}(x) = |S(x, V_{i+1})|$ and $\sigma_{V,v_0}(V_i) = min_{x \in V_i} \sigma_{V,v_0}(x)$. Note that σ_{V,v_0} depends on the choice of both V and $v_0 \in V$, since the levels V_i are defined with respect to v_0, and if $x \in V_i$, then $\sigma_{V,v_0}(x)$ is the sensitivity of F on x restricted to the set V_{i+1}. We omit from our notation the dependence of the set $S(x)$ and $\sigma_{V,v_0}(x)$ on F.

2.1. The proof. Our argument proceeds in two stages. First we choose a subset $V \subseteq \{0,1\}^n$ of inputs with certain properties that we define below. Informally these properties say that if F has sensitivity s on some input v_0, then we can start with v_0 and build a set of inputs V such that for "many" levels V_i of V (partitioning V into levels with respect to v_0) the following holds: on each input from V_i the sensitivity of F restricted to V_{i+1} is "high" (in terms of s). That is, not only the sensitivity of F is high on each selected input, but it remains high even when restricted to the next level of the set V.

In the second stage we use the set V to demonstrate a large number (in terms of s) of different values for F. This will give us the relation between the sensitivity s and the number of possible values D.

We start with a lemma stating that if the sensitivity of F on an input x is s, then the sensitivity must be "almost" s on many inputs from the set $S(x)$. Moreover, this holds even if we consider the sensitivity with respect to a partition of the inputs into levels.

LEMMA 2.4. *Let $F : \{0,1\}^n \to \{0,1\}^m$ be an m-output Boolean function with coordinate functions $f_j : \{0,1\}^n \to \{0,1\}$ such that $s(f_j) \leq k$ for $1 \leq j \leq m$. Let $S(x,U)$ be defined with respect to F as above. Let $B = \{0,1\}^n$, and let $v_0 \in B$. Let B_i denote the ith level of B with respect to v_0, that is, the set of vectors in B that are at Hamming distance i from v_0. Let $x \in B_i$, and let $S \subseteq S(x, B_{i+1})$ such that $|S| = \sigma$. Then there are at least $\sigma/2$ nodes $v \in S$ such that $|S(v, B_{i+2})| \geq \sigma - 4k$.*

Proof. We think of B as the n-dimensional hypercube, with an edge connecting two nodes if and only if their Hamming distance is exactly 1. We number the nodes in S from 1 to σ and name them v_ℓ, $1 \leq \ell \leq \sigma$. (Recall that $S \subseteq B_{i+1}$.) Let $v_{(\ell,j)}$ for $1 \leq \ell, j \leq \sigma$, $\ell \neq j$, be the node in B_{i+2} whose Hamming distance to both v_ℓ and v_j is 1. (Note that by our notation $v_{(\ell,j)}$ and $v_{(j,\ell)}$ denote the same node.) Let $e_{\ell,j}$ be the edge that connects v_ℓ to $v_{(\ell,j)}$.

Call an edge *sensitive* if it connects nodes x and y such that $F(x) \neq F(y)$. First we show that among the $\sigma(\sigma-1)$ edges $e_{\ell,j}$ at most $\sigma 2(k-1)$ are not sensitive edges. To this end we partition the $\sigma(\sigma-1)$ edges into σ sets. The set E_ℓ contains the edges $e_{j,\ell}$ for $j \neq \ell$, $1 \leq j \leq \sigma$; that is, the set E_ℓ contains for each node v_j, $j \neq \ell$, the edge that connects v_j to the node $v_{(j,\ell)}$.

Now we claim that in each set E_ℓ there are at most $2(k-1)$ edges which are not sensitive. Since $v_\ell \in S$, there is some coordinate t such that $f_t(v_\ell) \neq f_t(x)$. Without loss of generality assume that $f_t(x) = 0$ and $f_t(v_\ell) = 1$. For an edge $e_{j,\ell}$ which is not sensitive, we have that $F(v_j) = F(v_{(\ell,j)})$, and, in particular, $f_t(v_j) = f_t(v_{(\ell,j)})$. There can be at most $k-1$ such edges with $f_t(v_j) = 1$, since the sensitivity of f_t is at most k, and together with v_ℓ there are at most k nodes adjacent to x on which f_t is 1. Similarly, together with x there are at most k nodes adjacent to v_ℓ on which f_t is 0. It follows that in E_ℓ there are at most $2(k-1)$ edges which are not sensitive. Thus, among the $\sigma(\sigma-1)$ edges $e_{\ell,j}$ at most $\sigma 2(k-1)$ are not sensitive. By a simple averaging argument there are at most $\sigma/2$ nodes in S which are adjacent to at least $4(k-1)$ nonsensitive edges in $\cup_{\ell=1}^{\sigma} E_\ell$.

It follows that there are at least $\sigma/2$ nodes in S which are adjacent to more than $\sigma - 1 - 4(k-1) \geq \sigma - 4k$ sensitive edges in $\cup_{\ell=1}^{\sigma} E_\ell$. That is, there are at least $\sigma/2$ nodes in S such that the sensitivity of F restricted to B_{i+2} is at least $\sigma - 4k$ on each of them. □

Now we are ready to prove the existence of a set V of input vectors with the desired properties.

LEMMA 2.5. *If the sensitivity of F on v_0 is s, then there is a set $V \subseteq B = \{0,1\}^n$ such that $\sigma_{V,v_0}(V_i) \geq s/4$ for $i = 0, \ldots, \lceil s/(8k) \rceil - 1$.*

Proof. We construct the set V by building its levels V_i inductively. (Recall that V_i denotes the ith level of V with respect to v_0; that is, V_i is the set of vectors from V that are at Hamming distance i from v_0.) We have $V_0 = \{v_0\}$. We also partition the set B into levels with respect to v_0.

We prove by induction on i that we can build sets $V_0, \ldots, V_i$ such that

- for $0 \leq j < i$ and any $x \in V_j$, we have $\sigma_{V,v_0}(x) \geq (1/2)s - j2k$;
- for any $x \in V_i$, we have $\sigma_{B,v_0}(x) \geq s - i4k$.

If the sensitivity of F on v_0 is s, then by Lemma 2.4 there is a set $V_1 \subseteq S(v_0, B_1)$ of at least $s/2$ nodes such that for any $x \in V_1$ we have $|S(x, B_2)| = \sigma_{B,v_0}(x) \geq s - 4k$. This proves the statement for $i = 1$.

Now assume we have built the set V by levels up to level i. We now build level $i+1$. Consider a node $x \in V_i$. By the induction hypothesis $\sigma_{B,v_0}(x) \geq s - i4k$ for $x \in V_i$, that is, $|S(x, B_{i+1})| \geq s - i4k$. By Lemma 2.4 there is a set G_x of at least $(1/2)(s - i4k)$ nodes $v \in S(x, B_{i+1})$ such that $|S(v, B_{i+2})| \geq (s - i4k) - 4k = s - (i+1)4k$.

To build the set V_{i+1} we let

$$V_{i+1} = \cup_{x \in V_i} G_x \ .$$

It follows that each node $x \in V_i$ has at least $(1/2)s - i2k$ neighbors y in V_{i+1} for which $F(x) \neq F(y)$; that is, for any $x \in V_i$ it holds that

$$\sigma_{V,v_0}(x) \geq (1/2)s - i2k \ .$$

Moreover, for all the nodes $y \in V_{i+1}$, it holds that $\sigma_{B,v_0}(y) \geq s - (i+1)4k$. Thus our induction step is complete. Now, as long as $i \leq s/(8k)$ we have that $\sigma_{V,v_0}(x)$, for any $x \in V_i$, is at least $(1/2)s - (s/(8k))2k = (1/4)s$. □

This completes the first stage of our proof. Next we will show, using the set V guaranteed by the above lemma, that F must take many different values. We start by a claim that selects a number of different values among the neighbors of a single input vector.

CLAIM 1. *Let $x \in \{0,1\}^n$ and $S \subseteq S(x)$. If $|S| \geq \xi$, then for some ℓ, such that $\xi/k \leq \ell \leq m$, we can find ℓ vectors $y_1, \ldots, y_\ell$ among the vectors in the set S, and ℓ coordinates $j_1, \ldots, j_\ell$ among the m coordinates of F, such that $f_{j_a}(y_a) \neq f_{j_a}(x)$ for $a = 1, \ldots, \ell$ and $f_{j_a}(y_b) = f_{j_a}(x)$ for $1 \leq a < b \leq \ell$.*

The above claim says, for example, that if $F(x)$ is 0 in all coordinates, then we can find ℓ vectors $y_1, \ldots, y_\ell$ in S, for $\xi/k \leq \ell \leq m$, and a set of ℓ coordinates such that if we appropriately reorder the coordinates of F, then the values of F on these vectors have the following form: $F(y_1) = (1.....)$, $F(y_2) = (01....)$, $F(y_3) = (001...)$, $F(y_4) = (0001..)$, and so on.

Proof of Claim 1. We can take any vector from S as y_1, and let j_1 be the first coordinate where $F(y_1)$ and $F(x)$ differ, that is, $f_{j_1}(y_1) \neq f_{j_1}(x)$. Since the sensitivity of each f_j is at most k, there are at most $k-1$ other vectors in S on which the value of F differs from $F(x)$ in the same coordinate j_1. Pick any of the remaining vectors from

S as y_2, and let j_2 be the first coordinate where $F(y_2)$ and $F(x)$ differ. Thus we can continue this process for at least $\ell \geq \xi/k$ steps and find the vectors and coordinates required by the claim. Note that $\ell \leq m$, since there are only m coordinates. □

Selecting different values. We now define a procedure to select input vectors from the set V guaranteed by Lemma 2.5 in a way that F has a different value on each selected vector.

DEFINITION 2.6.

- *A* signature *of a node x is a subset $T \subseteq [1...m]$ of coordinates, and a set of cardinality $|T|$ of binary values $\epsilon_j \in \{0,1\}$, such that $f_j(x) = \epsilon_j$ for $j \in T$. The signature of x serves as a witness of the value of F on x. We say that the length of the signature is $|T|$.*
- *We say that the signatures of the nodes x and y are* inconsistent *if there is a coordinate j that participates in both signatures but $f_j(x) \neq f_j(y)$.*
- *We say that a node z* preserves *the signature of the node x if for each coordinate j participating in the signature of x it holds that $f_j(z) = f_j(x)$.*
- *For $x \in V_i$, let $S_x \subseteq S(x, V_{i+1})$ be the subset of $S(x, V_{i+1})$ consisting of the nodes that preserve the signature of x.*

The following observation follows directly from the definitions above.

OBSERVATION 1. *For any two nodes x and y, if the signatures of x and y are inconsistent, then $S_x \cap S_y = \emptyset$.*

The next observation requires a simple proof.

OBSERVATION 2. *If the length of the signature of a node $x \in V_i$ is ν, then $|S_x| \geq |S(x, V_{i+1})| - \nu k$.*

Proof. Since the sensitivity of each f_j is at most k, there are at most νk vectors in $S(x, V_{i+1})$ on which F takes a value that differs from $F(x)$ in at least one of the ν coordinates that belong to the signature of x. □

We now describe the procedure for selecting input vectors with different values. In this process we will assign to each selected vector a *signature* (defined above) and an *address*, which is a sequence of integers, representing the way the vector was selected, as defined in what follows. The procedure selects from each level V_i a subset of the nodes that we denote by Z_i. The procedure starts with selecting the node v_0 to which we assign a signature of size 0 and address of size 0 (i.e., an empty signature and an empty address). We thus have $Z_0 = \{v_0\}$. For any i, the procedure selects the set Z_{i+1} after the set Z_i has been determined. To determine the set Z_{i+1} we start with the sets $S_x \subseteq S(x, V_{i+1})$ for $x \in Z_i$. (Recall that the nodes in S_x preserve the signature of x by definition.) If S_x is not empty, we apply Claim 1 to x and S_x to get nodes to be included in Z_{i+1}. We get a set Y_x of at least $\ell \geq |S_x|/k$ vectors, $Y_x = \{y_1, \ldots, y_\ell\}$. Note that Claim 1 selects ℓ coordinates, such that for every $1 \leq a \leq \ell$ the value of $F(y_a)$ differs from the value of $F(x)$ on the ath selected coordinate, and the value of a coordinates are fixed. On the other hand, since for each vector $y \in S_x$ the value $F(y)$ must be consistent with the signature of x, none of the ℓ coordinates chosen by Claim 1 participates in the signature of x. We thus set the *signature* of y_a by adding the additional a coordinates and their values fixed by Claim 1 to the signature of x (which is consistent with $F(y_a)$). Thus, if the length of the signature of x is ν, then the length of the signature of y_a is $\nu + a$. We obtain the *address* of y_a by appending the integer a to the address of x. The set Z_{i+1} is defined to be the union of the vectors selected for each node $x \in Z_i$, that is, $Z_{i+1} = \cup_{x \in Z_i} Y_x$. We continue this procedure as long as the last set Z_i is not empty.

Next we analyze the properties of our procedure. First note that the address

of a node $x \in Z_i$ consists of exactly i integers. We also have the following simple observation.

OBSERVATION 3. *For $x \in Z_i$, the length of the signature of x is exactly $\sum_{u=1}^{i} t_u$ if the address of x is $(t_1, t_2, \ldots, t_i)$.*

The following two claims show that the values of F on all inputs in $Z = \cup_{i \geq 0} Z_i$ are all different.

CLAIM 2. *For any two vectors y and z from the same level Z_i $(i \geq 1)$, the signatures of y and z are inconsistent.*

Proof. We prove the claim by induction on i. The statement holds for $i = 1$, since the vectors in Z_1 and their signatures were selected by applying Claim 1 to v_0. Now assume that we have proved the statement for i; that is, for any $x_1, x_2 \in Z_i$ the signatures of x_1 and x_2 are inconsistent. Then by Observation 1 we have $S_{x_1} \cap S_{x_2} = \emptyset$. Recall that every $y \in Z_{i+1}$ must belong to a set S_x for some $x \in Z_i$. Thus for every $y \in Z_{i+1}$ we have a unique $x \in Z_i$ such that $y \in S_x$. Let $y, z \in Z_{i+1}$. If $y, z \in S_x$ for $x \in Z_i$, then the signatures of y and z must be inconsistent in the part that was appended to the signature of x after applying Claim 1 to x and S_x. If $y \in S_{x_1}$ and $z \in S_{x_2}$, for $x_1 \neq x_2$, then the signatures of y and z must be inconsistent in the part that was "inherited" from x_1 and x_2, respectively. ∎

COROLLARY 1. *For any two vectors y and z from the same level Z_i $(i \geq 1)$, we have $F(y) \neq F(z)$.*

We will think of the set $Z = \cup_{i \geq 0} Z_i$ as a tree rooted at v_0. For a node $z \in Z_{i+1}$, we define its *parent* to be the node $x \in Z_i$ such that $z \in S_x$. Note that there is exactly one such node since for any $x, y \in Z_i$ we have $S_x \cap S_y = \emptyset$ by Observation 1 and Claim 2. Thus Z indeed forms a rooted tree. We say that y is an *ancestor* of z if there is a path from z to y in Z such that each step along the path leads from a node to its parent.

CLAIM 3. *For any two vectors from different levels $y \in Z_{i_1}$ and $z \in Z_{i_2}$, $i_1 \neq i_2$, we have $F(y) \neq F(z)$.*

Proof. Assume that there are vectors $y \in Z_i$ and $z \in Z_j$ for $i < j$ such that $F(y) = F(z)$. This is only possible if y is an ancestor of z, since each vector we select must preserve the signature of its parent and by Claim 2 the signatures are all inconsistent within each level. For the case that y is an ancestor of z, note that z must preserve the signature of each of its ancestors on the path in Z from z to y, in particular the signature of y' for some $y' \in S_y \subseteq S(y, V_{i+1})$. However, this is not possible if $F(y) = F(z)$ because y' and its signature were selected by applying Claim 1 to y; thus the signature of y' contains a coordinate $\ell \in [1...m]$ such that $f_\ell(y') \neq f_\ell(y)$. ∎

Counting the number of selected values. Since no two selected input vectors are mapped to the same value by F, it remains to show that we select a "large" number of vectors. We will prove the following theorem.

THEOREM 2.7. *If for a set $V \subseteq \{0,1\}^n$ and vector $v_0 \in V$ the sensitivity of F restricted to the levels of V satisfies $\sigma_{V,v_0}(V_i) \geq \xi = \Delta k$ for $i = 0, \ldots, i^* - 1$, then F takes at least $\sum_{i=0}^{i^*} \binom{\Delta}{i}$ different values.*

Proof. Let V be a set that satisfies the conditions set forth in the theorem. We will show that by applying the procedure described above to the set V, we select at least $\sum_{i=0}^{i^*} \binom{\Delta}{i}$ input vectors. Let $Z = \cup_{i \geq 0} Z_i$ be the set obtained by applying our procedure to the set V.

We say that the *degree* of a node $z \in Z$ is the number of vectors selected by our procedure from the set S_z (by applying Claim 1 to z and S_z), that is, the number of

z's children in the tree.

OBSERVATION 4. *If for some $0 \le i \le i^* - 1$ and $x \in Z_i$ the length of the signature of x is ν, then the degree of x is at least $\Delta - \nu$.*

Proof. Since $\sigma_{V,v_0}(V_i) \ge \xi = \Delta k$ for $i = 0, \ldots, i^*-1$, we have that $|S(x, V_{i+1})| \ge \xi$. Thus, $|S_x| \ge \xi - \nu k = k(\Delta - \nu)$ must hold by Observation 2. It follows from Claim 1 that the degree of x is at least $\Delta - \nu$. □

As described in the above selection procedure, we number the children of a given node by the number of additional coordinates we fix when applying Claim 1 to this node. Thus the address $(t_1, t_2, \ldots, t_i)$ of a node in Z_i specifies the path to follow on the tree from v_0 to reach this node. The path contains one node from each level Z_ℓ for $0 \le \ell \le i$ such that the node from Z_j on the path is the t_jth child of the node from Z_{j-1}.

The following lemma is helpful in counting the number of vertices in Z_i.

LEMMA 2.8. *If $t_u \ge 1$ is an integer for $1 \le u \le i \le i^*$ and $(t_1, t_2, \ldots, t_i)$ satisfies*

$$\sum_{u=1}^{i} t_u \le \Delta \,, \tag{2.1}$$

then $(t_1, t_2, \ldots, t_i)$ is a valid address in the tree Z.

Proof. We prove by induction on i that if $(t_1, t_2, \ldots, t_i)$ is a solution of (2.1) such that $t_u \ge 1$ is an integer for $1 \le u \le i \le i^*$, then $(t_1, t_2, \ldots, t_i)$ is a valid address in the tree.

To prove the statement for $i = 1$, we need to observe that v_0 has at least Δ children; thus if $t_1 \le \Delta$, then (t_1) is a valid address.

Assume the statement is true for $i \le i^* - 1$, and let $(t_1, t_2, \ldots, t_i, t_{i+1})$ be a solution to $\sum_{u=1}^{i+1} t_u \le \Delta$. It follows that

$$\sum_{h=1}^{i} t_h \le \Delta - t_{i+1} < \Delta \,;$$

thus by our induction hypothesis $(t_1, t_2, \ldots, t_i)$ is a valid address that leads to a node v. By Observation 3, the length of the signature of v is $\sum_{h=1}^{i} t_h$, and by Observation 4 its degree in the tree is at least $\Delta - \sum_{h=1}^{i} t_h$. If $(t_1, t_2, \ldots, t_i, t_{i+1})$ is a solution to $\sum_{u=1}^{i+1} t_u \le \Delta$, then $1 \le t_{i+1} \le \Delta - \sum_{h=1}^{i} t_h$, thus v has indeed at least t_{i+1} children and $(t_1, t_2, \ldots, t_{i+1})$ is a valid address. □

We now prove the following lemma.

LEMMA 2.9. *The number of nodes selected in level $i \le i^*$, $|Z_i|$, satisfies*

$$|Z_i| \ge \binom{\Delta}{i} \,.$$

Proof. The number of nodes $|Z_i|$ at level i is the number of different sequences of length i $(t_1, t_2, \ldots, t_i)$, where $t_u \ge 1$ is an integer for $1 \le u \le i$, and such that $(t_1, t_2, \ldots, t_i)$ corresponds to a valid address in the tree.

There are exactly $\binom{\Delta}{i}$ different solutions for (2.1) where $t_u \ge 1$ is integer for $1 \le u \le i$ (cf. [27]). To see this, note that we can specify a solution for (2.1) by choosing i distinct integers between 1 and Δ, and this gives a bijection between solutions of (2.1) with the required properties and sets of i distinct integers between 1 and Δ. Suppose that the i distinct integers are $1 \le n_1 < \cdots < n_i \le \Delta$. Then

by taking $t_1 = n_1$ and $t_u = n_u - n_{u-1}$ we get a solution for (2.1) with the required properties. Thus by Lemma 2.8 we have that the number of nodes selected at level $i \le i^*$ is $|Z_i| \ge \binom{\Delta}{i}$. □

Since we have shown that F takes different values on each input in Z, Theorem 2.7 follows from the above lemma. □

We can now conclude this section with the following proof.

Proof of Theorem 2.2. Let the sensitivity of F be s. Then there is at least one node v_0 that has sensitivity s. Using Lemma 2.5 we construct a set $V \subseteq \{0,1\}^n$ such that $\sigma_{V,v_0}(V_i) \ge s/4$ for any $0 \le i \le \lceil s/(8k) \rceil - 1$.

We apply Theorem 2.7 to the set V with parameters $i^* = \lceil s/(8k) \rceil$ and $\Delta = \lfloor s/(4k) \rfloor$.

We get that the number of values that F takes is at least

$$\sum_{i=0}^{\lceil s/(8k) \rceil} \binom{\lfloor s/(4k) \rfloor}{i} \ge 2^{\lfloor s/(4k) \rfloor - 1} .$$

Since by the hypothesis of the theorem the number of values that F takes is D, we get

$$2^{\lfloor s/(4k) \rfloor - 1} \le D$$

and

$$s \le k \cdot 4(\log_2 D + 2) . \quad \square$$

3. Applications to private computation. In this section we apply the theorem on sensitivity of the previous section to give improved lower bounds on the number of rounds required for the private computation of Boolean functions, given a certain amount of randomness. Comparing our results with known protocols [25] shows that our lower bounds are tight (up to a small constant factor) in terms of the sensitivity of the functions and the amount of randomness. We first briefly define private protocols and the complexity measures that we are interested in.

Let $f : \{0,1\}^n \to \{0,1\}$ be any Boolean function. A set of n players P_i ($1 \le i \le n$), each possessing a single input bit x_i (known *only* to P_i), collaborate in a protocol to compute the value of $f(x)$. The protocol operates in rounds. In each round each player may toss some coins and then sends messages to the other players. (Messages are sent over private channels so that other than the intended receiver no other player can listen to them.) After sending its messages, each player receives the messages sent to it by the other players in the current round. In addition, each player chooses to locally output the value of the function at a certain round. We say that the protocol computes the function $f : \{0,1\}^n \to \{0,1\}$ if for every input $x \in \{0,1\}^n$ the output produced by each player is $f(x)$. Sometimes it is more convenient to model the coin tossing done by each player, as a set of binary random tapes R_i, each R_i being provided to player P_i. The number of random coins tossed by player P_i is the number of random bits it reads from its random tape.

Notation. We denote by R_i a specific random tape provided to player P_i and by $\vec{R} = (R_1, \ldots, R_n)$ the vector of the random tapes of all the players. By $\vec{r} = (r_1, \ldots, r_n)$ we denote the random variable for these tapes and vector of tapes. Note that if we fix $\vec{R}$, we obtain a deterministic protocol. By C_i we denote a specific sequence of messages received by P_i, and c_i denotes the random variable (depending on $\vec{r}$) for the sequence of messages received by P_i.

Informally, *privacy* with respect to player P_i means that player P_i cannot learn anything (in particular, the inputs of the other players) from the messages it receives, except what is implied by its input bit, and the value of the function computed. Formally,

DEFINITION 3.1 (privacy). *A protocol $\mathcal{A}$ for computing a function f is* private *with respect to player P_i if for any two input vectors x and y, such that $f(x) = f(y)$ and $x_i = y_i$, for any sequence of messages C_i, and for any random tape R_i provided to P_i,*

$$Pr[c_i = C_i | R_i, x] = Pr[c_i = C_i | R_i, y],$$

where the probability is over the random tapes of all other players.

A protocol is said to be private if it is private with respect to all players.

DEFINITION 3.2 (randomness complexity). *A d-random protocol is a protocol such that for any input assignment the total number of coins tossed by all players in any execution is at most d.*

Let $View_i^t$ be the view of player i at round t, that is, the input bit to this player and all the messages it has seen until round t. Note that $View_i^t$, for any i and t, is a function of the input assignment x and the random tapes of all the players. We can thus write it as $View_i^t(x, \vec{r})$. We denote by $T_i(x, \vec{r})$ the round number in which player P_i outputs its result as a function of the input to all players and of the random tapes given to all players.

DEFINITION 3.3 (rounds complexity). *A ρ-round protocol is a protocol such that for all i, x, $\vec{R}$ we have $T_i(x, \vec{R}) \leq \rho$.*

We will also make use of the following definition.

DEFINITION 3.4 (expected rounds complexity).[2] *An expected ρ-round protocol is a protocol such that there exists a player P_i such that, for all x, $E_{\vec{r}}[T_i(x, \vec{r})] \leq \rho$.*

We are interested in tradeoffs between the rounds complexity (and expected rounds complexity) and the randomness complexity of private protocols.

Our proof follows the line of proof of [25]. We get (almost) tight lower bounds by using our result of the previous section on the sensitivity of multiple-output Boolean functions. For completeness we give below the full proof. We will prove the following theorem.

THEOREM 3.5. *Let $\mathcal{A}$ be a ρ-round d-random ($d \geq 2$) private protocol to compute a Boolean function f. Then, $\rho \geq \Omega(\frac{\log s(f)}{\log d})$, where $s(f)$ is the sensitivity of f.*

To prove the above theorem we use the following lemma from [25].

LEMMA 3.6 ([25, Lemma 4.11]). *Consider a private d-random protocol to compute a Boolean function f. Fix random tapes $\vec{R} = (R_1, \ldots, R_n)$. Then, for any P_i, $View_i^t(x, \vec{R})$ can assume at most 2^{d+2} different values (over the values of x).*

Using the above lemma we can prove our next lemma, following the proof in [25], but using our tight bound on the sensitivity of multiple-output Boolean functions.

LEMMA 3.7. *Consider a private d-random protocol to compute a Boolean function f, and consider a specific vector of random tapes $\vec{R}$ and the deterministic protocol derived by it. Then for every player P_i, the function $View_i^t(x, \vec{R})$ (as a function of x only) has sensitivity at most $Q(t) \stackrel{\triangle}{=} (4(d+4))^{t-1}$.*

Proof. First note that since we fix the random tapes the views of the players are functions of the input assignment x only. (We regard each bit of the view as a

[2] We adopt this weak definition, rather than requiring the property to hold for *all* players, as we are interested here in lower bounds.

Boolean function of x.) We prove the lemma by induction on t. For $t = 1$ the view of any player depends only on its single input bit. Thus, the claim is obvious. For $t > 1$ assume the claim holds for any $\ell < t$ and for any player. For $t > 1$ the view of any P_i is composed of the input bit of P_i, and bits that were received as messages, each one in some round $\ell < t$. Each such message bit is a function of the view of the player that sent it at the round in which it was sent. That is, each such message bit is a function of a view with sensitivity at most $Q(t-1)$, and hence the sensitivity of each such message bit (regarded as a Boolean function of x) is at most $Q(t-1)$. Clearly the input bit of P_i has sensitivity 1 which is at most $Q(t-1)$. Thus, the view under consideration is composed of coordinates each having sensitivity at most $Q(t-1)$. Moreover, by Lemma 3.6 the view can assume at most 2^{d+2} values. It follows from Theorem 2.2 that the sensitivity of the view under consideration is at most $Q(t-1) \cdot 4(d+4) = Q(t)$. □

We can now give the proof of the main theorem of this section.

Proof of Theorem 3.5. Consider the deterministic protocol obtained from a d-random private protocol ($d \geq 2$) by fixing the vector of random tapes to be a given vector $\vec{R}$.

We prove that for any player P_i there is at least one input assignment x such that

$$T_i(x, \vec{R}) \geq \log s(f)/(2 + \log(d+4)) + 1 \ .$$

This proves our theorem, since for every player it shows a run of the protocol where the player outputs its value only after $\log s(f)/(2 + \log(d+4)) + 1$ rounds.

Let y be an input assignment on which the sensitivity $s(f)$ is obtained. That is, y has $s(f)$ neighbors (in the inputs hypercube) where the value of f is different from $f(y)$. Denote this set of neighbors by $S(y)$. Denote by t the value of $T_i(y, \vec{R})$, i.e., the time step at which P_i outputs the value of f when the input to all players is y.

Now consider the view of P_i at round t, denoted $View^t(x, \vec{R})$, and the sensitivity of this view. Assume towards a contradiction that the sensitivity of $View^t(x, \vec{R})$ is less than $s(f)$. Then, in particular, the sensitivity of this view on y is less than $s(f)$. It follows that for at least one input assignment $z \in S(y)$, $View^t(y, \vec{R}) = View^t(z, \vec{R})$, and P_i would output the same value for f on inputs y and z, contradicting the fact that it is a correct protocol.

Thus the sensitivity of $View^t(x, \vec{R})$ on input y is at least $s(f)$, and t is such that $s(View_i^t(x, \vec{R})) \geq s(f)$. By Lemma 3.7, we get $(4(d+4))^{t-1} \geq s(f)$, i.e., $t \geq \frac{\log s(f)}{2+\log(d+4)} + 1$. It follows that $T_i(y, \vec{R}) \geq \frac{\log s(f)}{2+\log(d+4)} + 1$. □

Using similar techniques, that also follow the proofs from [25], we can obtain the following improved bound on the expected number of rounds of private protocols.

THEOREM 3.8. *Let $\mathcal{A}$ be an expected ρ-round, d-random ($d \geq 2$) private protocol to compute a Boolean function f. Then, $\rho \geq \Omega(\frac{as(f)}{n} \cdot \frac{\log as(f)}{\log d})$.*

Proof. To prove the theorem we consider a protocol $\mathcal{A}$ and fix any player P_i. We say that the protocol is *late* on input x and vector of random tapes $\vec{R}$ if $T_i(x, \vec{R}) \geq \frac{\log as(f)}{4+2\log(d+4)} + 1$. We define a $0-1$ random variable $L(x, \vec{r})$ to be 1 if and only if the protocol is late on x and $\vec{r}$. For the purpose of this proof we also define a uniform distribution on the 2^n input assignments. (This is not to say that the input assignments are actually drawn according to such distribution.)

We first show that for any deterministic protocol to compute f, derived from a private protocol by fixing $\vec{R}$, not only is there at least one input on which the protocol is late but that this happens for a large fraction of the inputs.

LEMMA 3.9. *Consider a player P_i and any fixed vector of random tapes $\vec{R} = (R_1, \ldots, R_n)$. Then*

$$E_x[L(x, \vec{R})] \geq \frac{as(f) - \sqrt{as(f)}}{2n} .$$

Proof. Consider the views of P_i, $View_i^t$, given the vector of random tapes $\vec{R}$. For any round t such that $t < \frac{\log as(f)}{4+2\log(d+4)} + 1$, by Lemma 3.7, we get that $s(View_i^t) < (4(d+4))^{\frac{\log as(f)}{4+2\log(d+4)}} = \sqrt{as(f)}$. Any function g computed from such a view can have at most the same sensitivity and thus clearly an average sensitivity of at most $\sqrt{as(f)}$. Such function g can have the same values as f on at most $2^n(1 - \frac{as(f)-\sqrt{as(f)}}{2n})$ input assignments. It follows that at least $2^n \frac{as(f)-\sqrt{as(f)}}{2n}$ input assignments are late. ∎

The lower bound on the expected number of rounds now follows. Since at least $2^n \frac{as(f)-\sqrt{as(f)}}{2n}$ input assignments are late for any set of random tapes, $E_{\vec{r},x}[L(x, \vec{r})] \geq \frac{as(f)-\sqrt{as(f)}}{2n}$. Hence, there is at least one input assignment x for which $E_{\vec{r}}[L(x, \vec{r})] \geq \frac{as(f)-\sqrt{as(f)}}{2n}$. For such x we get

$$E_{\vec{r}}[T_i(x, \vec{r})] \geq \left(\frac{as(f) - \sqrt{as(f)}}{2n}\right) \cdot \left(\frac{\log as(f)}{4 + 2\log(d+4)} + 1\right),$$

as needed. ∎

4. Conclusions. In this paper we prove an almost tight upper bound on the sensitivity of multiple-output Boolean functions, in terms of the sensitivity of each output coordinate, and the size of the range of the function. Using this bound, we establish improved lower bounds on the number of rounds of private protocols, in terms of the sensitivity of the function that they compute, and the amount of randomness that they use. These lower bounds are tight (up to a small constant factor) for the function `xor`.

We believe that the theorem on the sensitivity is of independent interest, and it would be interesting to see if it can find additional applications. Also, it would be interesting to close the remaining (small constant factor) gap in our bound on sensitivity. In fact, we conjecture that the right bound is $k\lceil \log_2 D \rceil$, which can be achieved for $D = 2^q$ by a construction of q independent coordinate functions.

Acknowledgments. We thank Eyal Kushilevitz for many useful discussions on privacy and randomness and on the theorem on sensitivity presented here. We also thank Dimitri Achlioptas for useful discussions.

REFERENCES

[1] J. BAR-ILAN AND D. BEAVER, *Non-cryptographic fault-tolerant computing in a constant number of rounds*, in Proceedings of the 8th ACM Symposium on Principles of Distributed Computing, 1989, pp. 201–209.

[2] D. BEAVER, *Perfect Privacy for Two-Party Protocols*, Technical Report-11-89, Harvard University, Cambridge, MA, 1989.

[3] D. BEAVER, J. FEIGENBAUM, J. KILIAN, AND P. ROGAWAY, *Security with Low Communication Overhead*, in Advances in Cryptology—CRYPTO '90, Lecture Notes in Comput. Sci. 537, Springer-Verlag, Berlin, 1991, pp. 62–76.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness theorems for non-cryptographic fault-tolerant distributed computation*, in Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 1–10.

[5] C. Blundo, A. De Santis, G. Persiano, and U. Vaccaro, *On the number of random bits in totally private computations*, in Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 944, Springer-Verlag, Berlin, 1995, pp. 171–182.

[6] R. Canetti, E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Randomness vs. fault-tolerance*, J. Cryptology, 13 (2000), pp. 107–142.

[7] D. Chaum, C. Crepeau, and I. Damgard, *Multiparty unconditionally secure protocols*, in Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 11–19.

[8] B. Chor and E. Kushilevitz, *A zero-one law for Boolean privacy*, SIAM J. Discrete Math., 4 (1991), pp. 36–47.

[9] B. Chor, M. Geréb-Graus, and E. Kushilevitz, *Private computations over the integers*, SIAM J. Comput., 24 (1995), pp. 376–386.

[10] S. Cook, C. Dwork, and R. Reischuk, *Upper and lower time bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput., 15 (1986), pp. 87–97.

[11] R. L. Dobrushin and S. I. Ortyukov, *Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements*, Probl. Inf. Transm., 13 (1977), pp. 59–65.

[12] U. Feige, J. Kilian, and M. Naor, *A minimal model for secure computation*, in Proceedings of the 26th ACM Symposium on the Theory of Computing, 1994, pp. 554–563.

[13] M. Franklin and M. Yung, *Communication complexity of secure computation*, in Proceedings of the 24th ACM Symposium on the Theory of Computing, 1992, pp. 699–710.

[14] P. Gács and A. Gál, *Lower bounds for the complexity of reliable Boolean circuits with noisy gates*, IEEE Trans. Inform. Theory, 40 (1994), pp. 579–583.

[15] A. Gál, *Lower bounds for the complexity of reliable Boolean circuits with noisy gates*, in Proceedings of the 32nd Annual Symposium on Foundation of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 594–601.

[16] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, in Proceedings of the 19th ACM Symposium on the Theory of Computing, 1987, pp. 218–229.

[17] C. Gotsman and N. Linial, *The equivalence of two problems on the cube*, J. Combinatorial Theory Ser. A, 61 (1992), pp. 142–146.

[18] Y. Ishai and E. Kushilevitz, *Private simultaneous messages protocols with applications*, in Proceedings of the Fifth Israel Symposium on Theory of Computing and Systems, Ramat-Gan, Israel, 1997, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 174–184.

[19] J. Kahn, G. Kalai, and N. Linial, *The influence of variables of Boolean functions*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 68–80.

[20] C. Kenyon, *Sensitivity vs. Block Sensitivity of Boolean Functions*, DIMACS Technical Report 90-18, Rutgers University, Piscataway, NJ, 1990.

[21] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky, *Reducibility and completeness in private computations*, SIAM J. Comput., 29 (2000), pp. 1189–1208.

[22] E. Kushilevitz and Y. Mansour, *Randomness in private computations*, SIAM J. Discrete Math., 10 (1997), pp. 647–661.

[23] E. Kushilevitz, *Privacy and communication complexity*, SIAM J. Discrete Math., 5 (1992), pp. 273–284.

[24] E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Characterizing linear size circuits in terms of privacy*, J. Comput. System Sci., 58 (1999), pp. 129–136.

[25] E. Kushilevitz and A. Rosén, *A randomness-rounds tradeoff in private computation*, SIAM J. Discrete Math., 11 (1998), pp. 61–80.

[26] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM, 40 (1993), pp. 607–620.

[27] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, Cambridge University Press, Cambridge, UK, 1992.

[28] N. Nisan, *CREW PRAMs and decision trees*, SIAM J. Comput., 20 (1991), pp. 999–1007.

[29] N. Nisan and M. Szegedy, *On the degree of Boolean functions as real polynomials*, Comput. Complexity, 4 (1994), pp. 301–313.

[30] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis, *On a lower bound for the redundancy of reliable networks with noisy gates*, IEEE Trans. Inform. Theory, 37 (1991), pp. 639–643.

[31] R. Reischuk and B. Schmeltz, *Reliable computation with noisy circuits and decision trees—A general n log n lower bound*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 602–611.

[32] D. Rubinstein, *Sensitivity vs. block sensitivity of Boolean functions*, Combinatorica, 15 (1995), pp. 297–299.

[33] H. U. Simon, *A tight* $\Omega(\log\log n)$ *bound on the time for parallel RAM's to compute nondegenerate Boolean functions*, in Proceedings of the 1983 International Foundations of Computation Theory Conference, Lecture Notes in Comput. Sci. 158, Springer-Verlag, Berlin, 1983, pp. 439–444.

[34] G. Turán, *The critical complexity of graph properties*, Inform. Process. Lett., 18 (1984), pp. 151–153.

[35] I. Wegener, *The critical complexity of all (monotone) Boolean functions and monotone graph properties*, Inform. and Control, 67 (1985), pp. 212–222.

[36] A. C. Yao, *Protocols for secure computations*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, IEEE, New York, 1982, pp. 160–164.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1438–1455

A LINEAR ALGORITHM FOR FINDING THE INVARIANT EDGES OF AN EDGE-WEIGHTED GRAPH*

F. M. MALVESTUTO† AND M. MEZZINI‡

Abstract. Given an edge-weighted graph where all weights are nonnegative reals, an edge reweighting is an assignment of nonnegative reals to edges such that, for each vertex, the sums of given and new weights assigned to the edges incident on the vertex do coincide. An edge is then said to be *invariant* if its weight is the same for any edge reweighting. We show that the set of invariant edges of an arbitrary edge-weighted graph can be determined in time linear in the size of the underlying graph. Moreover, an application to the security of statistical data is discussed.

Key words. linear algebra, graph algorithms, matroid theory

AMS subject classifications. 05B35, 05C50, 15A03, 62Q05, 68R10

PII. S0097539700376068

1. Introduction.

Let $G = (V, E)$ be a graph without isolated vertices (where self-loops and parallel edges may exist), and let $\boldsymbol{w} = (w(e))_{e\in E}$ be a vector of nonnegative reals. The pair $\Gamma = (G, \boldsymbol{w})$ is referred to as an *edge-weighted graph* EWG. Let $\mathbf{A}$ be the incidence matrix of G and let $\boldsymbol{b} = (b(v))_{v\in V}$ be the vector of nonnegative reals such that, for each vertex v of G, $b(v)$ equals the sum of the weights of the edges incident to v. Consider the following system of linear equations:

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}. \tag{1}$$

For every edge of G, let $L[x(e)]$ and $U[x(e)]$ denote, respectively, the minimum and the maximum of the variable $x(e)$ over the nonnegative solutions of equation system (1). An edge e of G is an *invariant edge* of Γ if $L[x(e)] = U[x(e)]$. Thus, an edge e of G is an invariant edge of Γ if and only if $x(e) = w(e)$ for every nonnegative solution of equation system (1). The following two examples show two EWGs which have all and no invariant edges, respectively.

EXAMPLE 1. *Consider the EWG $\Gamma = (G, \boldsymbol{w})$ shown in Figure 1, where α, β, and γ are any positive reals. By making use of standard algebraic methods, one finds there is no nonnegative solution of equation system (1) other than $\boldsymbol{w}$. Therefore, each edge of G is an invariant edge of Γ.* □

EXAMPLE 2. *Consider the EWG $\Gamma = (G, \boldsymbol{w})$ shown in Figure 2. The general expression of a nonnegative solution of equation system (1) is*

$$x(1,1) = 1 - 2\lambda, \qquad x(1,2) = x(1,3) = \lambda, \qquad x(2,3) = 1 - \lambda,$$

where the parameter λ ranges from 0 to 1/2. Therefore, one has

$$\begin{aligned} &L[x(1,1)] = 0, && U[x(1,1)] = 1,\\ &L[x(1,2)] = L[x(1,3)] = 0, && U[x(1,2)] = U[x(1,3)] = 1/2,\\ &L[x(2,3)] = 1/2, && U[x(2,3)] = 1, \end{aligned}$$

and hence, no edge of G is an invariant edge of Γ. □

*Received by the editors July 28, 2000; accepted for publication (in revised form) March 14, 2002; published electronically August 1, 2002.
http://www.siam.org/journals/sicomp/31-5/37606.html

†Dipartimento di Scienze dell'Informazione, Università "La Sapienza" di Roma, Italy (mal@dsi.uniroma1.it).

‡Wireline Services, Telecom Italia, Italy (mauro.mezzini@telecomitalia.it).

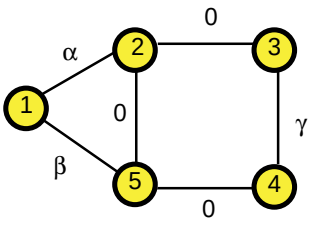


FIG. 1.

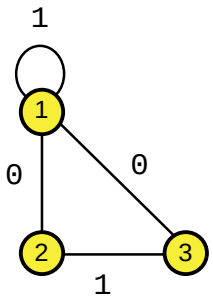


FIG. 2.

The problem addressed in this paper lies in finding the set of invariant edges of an arbitrary EWG. The following obvious fact allows us to limit our considerations to EWGs with underlying simple graphs (i.e., graphs without parallel edges).

Fact 1. Let $\Gamma = (G, \boldsymbol{w})$ be an EWG, where $G = (V, E)$ is a nonsimple graph. Let S be a set of two or more parallel edges and let e_0 be an arbitrarily chosen element of S. Let $G' = (V, E)$ be the graph with edge set $E' = (E - S) \cup \{e_0\}$. Consider the EWG $\Gamma' = (G', \boldsymbol{w}')$, where $\boldsymbol{w}'$ is defined as follows:

$$w'(e) = \begin{cases} w(e), & e \notin S, \\ \displaystyle\sum_{e' \in S} w(e'), & e = e_0. \end{cases}$$

Then, an edge not in S is an invariant edge of Γ if and only if it is an invariant edge of Γ', and an edge in S is an invariant edge of Γ if and only if $w'(e_0) = 0$ and e_0 is an invariant edge of Γ'.

The problem of finding the set of invariant edges of an EWG arises in the security analysis of statistical data, which will be discussed in section 6, and Gusfield [7] proved that if G is bipartite, then the set of invariant edges of Γ can be determined in time linear in the size of G. Here we present a linear time algorithm which finds the set of invariant edges of an arbitrary EWG.

2. Background. Let $G = (V, E)$ be a simple graph with vertex-edge incidence matrix $\mathbf{A}$. For any vector $\boldsymbol{x} = (x(e))_{e \in E}$, the *support* of $\boldsymbol{x}$ is the set $S = \{e \in E \colon x(e) \neq 0\}$, and the *signed support* of $\boldsymbol{x}$ is the ordered set pair (S^+, S^-), where $S^+ = \{e \in E \colon x(e) > 0\}$ and $S^- = \{e \in E \colon x(e) < 0\}$; moreover, the set $E - S$ is called the *zero set* of $\boldsymbol{x}$. The nonzero solutions of the homogeneous equation system $\mathbf{A}\boldsymbol{y} = \mathbf{0}$ are referred to as *circulations* in G and the linear space of the solutions of the homogeneous equation system $\mathbf{A}\boldsymbol{y} = \mathbf{0}$ is referred to as the *circulation space*.

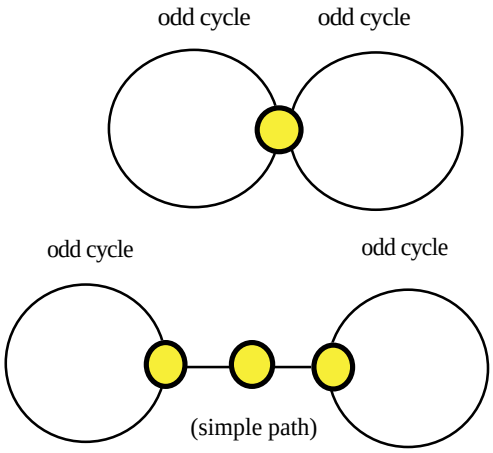


FIG. 3.

Thus, a nonempty subset S of E corresponds to a set of columns of $\mathbf{A}$ that are linearly dependent (over the field of reals) if and only if S contains the support of a circulation in G. A *minimal circulation* in G is a circulation in G with inclusion-minimal support. The following is a well-known result of linear algebra.

PROPOSITION 1 (e.g., see page 107 in [3]). *Let S and (S^+, S^-) be the support and the signed support of a circulation in G, respectively. For each edge e in S, there is a minimal circulation in G with support C and signed support (C^+, C^-) such that e is in C, $C^+ \subseteq S^+$ and $C^- \subseteq S^-$.*

The set of supports of minimal circulations in G can be viewed as the family of *circuits* of a matroid [20], which we denote by $\boldsymbol{\mathcal{M}}(G)$, whose rank (i.e., the rank of $\mathbf{A}$) is given by $|V| - q$, where q is the number of connected components of G that are bipartite (in that they contain no odd cycles); see Theorem 1, page 421 in [6], or [19]. Explicitly, a subset of E is a circuit of $\boldsymbol{\mathcal{M}}(G)$ if and only if it is the edge set of either an even simple cycle or a pair of two edge-disjoint odd simple cycles that either have exactly one vertex in common or are vertex-disjoint and are connected by a simple path (see Figure 3) [5].

Let Z be a (proper or improper) subset of E. We say that a circuit of $\boldsymbol{\mathcal{M}}(G)$ is *Z-traversable* if it is the support of a (minimal) circulation whose signed support (C^+, C^-) is such that $Z \cap C^- = \emptyset$.

Consider now the vectors that are linear combinations of rows of $\mathbf{A}$. The inclusion-minimal supports of these vectors are the *cocircuits* of $\boldsymbol{\mathcal{M}}(G)$; that is, they are minimal edge sets whose removal decreases the rank of $\boldsymbol{\mathcal{M}}(G)$ [20]. Moreover, an edge e of G is a *coloop* of $\boldsymbol{\mathcal{M}}(G)$ if the singleton $\{e\}$ is a cocircuit of $\boldsymbol{\mathcal{M}}(G)$. In other words, an edge e of G is a coloop of $\boldsymbol{\mathcal{M}}(G)$ if and only if the incidence vector of $\{e\}$ is a linear combination of rows of $\mathbf{A}$ or, equivalently, if and only if e is not in any circuit of $\boldsymbol{\mathcal{M}}(G)$ [20].

3. Invariant edges. In this section, we state a few characteristic properties of invariant edges of an arbitrary EWG which will be used later on. We need some preliminary definitions and results.

Let $\Gamma = (G, \boldsymbol{w})$ be an EWG with $G = (V, E)$ and let Z be the zero set of $\boldsymbol{w}$. A circulation in G with signed support (S^+, S^-) is said to be *legal* in Γ if $Z \cap S^- = \emptyset$. Accordingly, a circuit of $\boldsymbol{\mathcal{M}}(G)$ is Z-traversable if and only if it is the support of a

(minimal) circulation in G which is legal in Γ. It should be noted that if the weights of the edges of G are all positive, then Z is empty so that each circulation in G is legal in Γ.

THEOREM 1. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG. An edge of G is not an invariant edge of Γ if and only if it belongs to the support of a circulation in G which is legal in Γ.*

Proof. ("only if") Let e be an edge of G that is not an invariant edge of Γ. Then, there exists a nonnegative solution $\boldsymbol{x}$ of (1) with $x(e) \neq w(e)$. The vector $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{w}$ is then a circulation in G. Let S and (S^+, S^-) be the support and the signed support of $\boldsymbol{y}$, respectively, and let Z be the zero set of $\boldsymbol{w}$. Then e is in S and, since $y(e') = x(e') \geq 0$ for each e' in Z, one has $Z \cap S^- = \emptyset$; that is, e belongs to the support of a circulation which is legal in Γ.

("if") Let $\boldsymbol{y}$ be a legal circulation with support S and signed support (S^+, S^-), and let e be in S. Consider the solution $\boldsymbol{x} = \boldsymbol{w} + \boldsymbol{y}$ of equation system (1). If $\boldsymbol{x}$ is nonnegative everywhere, then the statement follows from the fact that e is in S, which implies $x(e) \neq w(e)$. Otherwise, let

$$S' = \{e' \colon x(e') < 0\} \qquad \text{and} \qquad \lambda = \min\{-w(e')/y(e') \colon e' \in S'\}.$$

Since S' is a subset of S^- and $\boldsymbol{y}$ is a legal circulation, λ is positive. Then the vector $\boldsymbol{y}' = \lambda \boldsymbol{y}$ is a circulation in G having the same support and the same signed support as $\boldsymbol{y}$. Consider the solution $\boldsymbol{x}' = \boldsymbol{w} + \boldsymbol{y}'$ of equation system (1). It is easily seen that $\boldsymbol{x}'$ is nonnegative everywhere since, for each e' not in S, one has trivially $x'(e') \geq 0$ and, for each e' in S', one has

$$x'(e') = w(e') + \lambda y(e') = -y(e')(-w(e')/y(e') - \lambda) \geq 0.$$

Finally, since e is in the support S of $\boldsymbol{y}$, one has

$$x'(e) = w(e) + \lambda\, y(e) \neq w(e),$$

which proves the statement. □

THEOREM 2. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG and let Z be the zero set of $\boldsymbol{w}$. An edge of G is not an invariant edge of Γ if and only if it belongs to some Z-traversable circuit of $\boldsymbol{\mathcal{M}}(G)$.*

Proof. ("if") It follows from the "if" part of Theorem 1.

("only if") If e is not an invariant edge of Γ, then by the "only-if" part of Theorem 1 there is a circulation in G with support S and signed support (S^+, S^-) such that e is in S and $Z \cap S^- = \emptyset$. But, by Proposition 1, there is a minimal circulation in G such that its support contains e and its signed support (C^+, C^-) is such that $C^- \subseteq S^-$. Therefore, one has $Z \cap C^- \subseteq Z \cap S^- = \emptyset$, which proves the statement. □

EXAMPLE 1 (continued). *The zero set of $\boldsymbol{w}$ is $Z = \{(2,3), (2,5), (4,5)\}$. The minimal circulations in G are summarized in Figure 4 by taking λ to be any nonzero real. So, $\boldsymbol{\mathcal{M}}(G)$ contains one circuit which is not Z-traversable. By Theorem 2, each edge of G is an invariant edge of Γ.* □

EXAMPLE 2 (continued). *The zero set of $\boldsymbol{w}$ is $Z = \{(1,2), (1,3)\}$. The minimal circulations in G are summarized in Figure 5 by taking λ to be any nonzero real. So, $\boldsymbol{\mathcal{M}}(G)$ contains one circuit which is Z-traversable. By Theorem 2, no edge of G is an invariant edge of Γ.* □

Note that if the zero set Z of $\boldsymbol{w}$ is empty, then, by Theorem 2, an edge of G is an invariant edge of Γ if and only if it is not in any circuit of $\boldsymbol{\mathcal{M}}(G)$; that is, if and

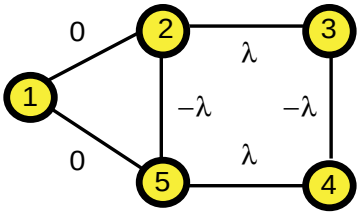


FIG. 4.

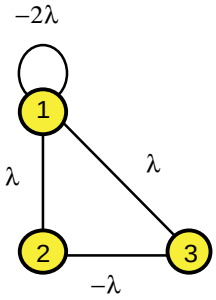


FIG. 5.

only if it is a coloop of $\boldsymbol{\mathcal{M}}(G)$. We now prove that the same holds in a more general case. In what follows, by the *kernel* of Γ [15] we mean the intersection of Z with the set of invariant edges of Γ □

LEMMA 1. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG whose kernel is empty. An edge of G is an invariant edge of Γ if and only if it is a coloop of $\boldsymbol{\mathcal{M}}(G)$.*

Proof. ("if") If an edge of G is a coloop of $\boldsymbol{\mathcal{M}}(G)$, then it is an invariant edge of Γ by the "only-if" part of Theorem 2.

("only if") Let e be an invariant edge of Γ. Suppose by contradiction that e is in some circuit of $\boldsymbol{\mathcal{M}}(G)$. Then, as is shown below, e should belong to the support of a legal circulation in Γ, which contradicts Theorem 1. To show that, suppose that e is in the circuit C_0 of $\boldsymbol{\mathcal{M}}(G)$. By the "only-if" part of Theorem 2, C_0 cannot be Z-traversable, where Z is the zero set of $\boldsymbol{w}$. Thus, if $\boldsymbol{c}_0$ is any minimal circulation in G with support C_0 and signed support (C_0^+, C_0^-), one has $Z \cap C_0^- \neq \emptyset$. Let $Z \cap C_0^- = \{e_1, \ldots, e_p\}$. Since the kernel of Γ is empty, no edge in Z is an invariant edge of Γ. Then, by the "only-if" part of Theorem 1, for each e_i, $1 \leq i \leq p$, there is a legal circulation $\boldsymbol{y}_i$ of Γ such that, if S_i is the support of $\boldsymbol{y}_i$, e_i is in S_i; moreover, if (S_i^+, S_i^-) is the signed support of $\boldsymbol{y}_i$, then $y_i(e_i) > 0$, since $Z \cap S_i^- = \emptyset$. Now consider the circulation

$$\boldsymbol{c}_i = [-c_0(e_i)/y_i(e_i)]\boldsymbol{y}_i.$$

Since $c_0(e_i) < 0$ (recall that $e_i \in C_0^-$) and $y_i(e_i) > 0$, $\boldsymbol{c}_i$ has the same support and signed support as $\boldsymbol{y}_i$, and hence is legal in Γ. Let

$$\boldsymbol{y} = \boldsymbol{c}_0 + \sum_{i=1,\ldots,p} \boldsymbol{c}_i.$$

Since the circulation space of $\mathbf{A}$ is a linear space, $\boldsymbol{y}$ is still a circulation in G. Let S and (S^+, S^-) be the support and signed support of $\boldsymbol{y}$, respectively. Finally, we now prove that (i) e is in S, and (ii) the circulation $\boldsymbol{y}$ is legal in Γ.

Proof of (i). Since e is an invariant edge of Γ, by the "if" part of Theorem 2, e is in the support of none of the legal circulations $\boldsymbol{c}_i$ so that $c_i(e) = 0$ for each i, $1 \leq i \leq p$. Therefore, $y(e) = c_0(e)$ and, since e is in C_0, one has that e is also in S.

Proof of (ii). In order to prove that the intersection of Z with S^- is empty, we separately examine the edges $e_1, \ldots, e_p$ in $Z \cap C_0^-$ and the edges in $Z - C_0^-$. For each i and j, $1 \leq i, j \leq p$, $c_j(e_i) \geq 0$, since $Z \cap S_j^- = \emptyset$. Moreover, for each i, $1 \leq i \leq p$, one has

$$c_i(e_i) = -c_0(e_i)$$

and, hence,

$$y(e_i) = c_0(e_i) + c_i(e_i) + \sum_{j \neq i} c_j(e_i) = \sum_{j \neq i} c_j(e_i) \geq 0.$$

Therefore, each e_i is not in S^-.

We now consider the edges in $Z - C_0^-$. If e' is such an edge, then $c_0(e') \geq 0$. Moreover, since $Z \cap S_i^- = \emptyset$, $1 \leq i \leq p$, one has $c_i(e') \geq 0$. Therefore

$$y(e') = c_0(e') + \sum_{i=1,\ldots,p} c_i(e') \geq 0,$$

and, hence, e' is not in S^-.

After proving (i) and (ii), by the "if" part of Theorem 1, one has that e is not an invariant edge of Γ (a contradiction). □

As a consequence of Lemma 1, we obtain the following characterization of invariant edges of an EWG.

THEOREM 3. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG with kernel K. The set of invariant edges of Γ is the union of K with the set of coloops of $\boldsymbol{\mathcal{M}}(G - K)$.*

Proof. Let $\Gamma' = (G', \boldsymbol{w}')$, where $G' = G - K$, and let $\boldsymbol{w}'$ be the restriction of $\boldsymbol{w}$ to the edge set of G'. It is clear that an edge of G is an invariant edge of Γ if and only if either it is in K or it is an invariant edge of Γ'. On the other hand, the kernel of Γ' is empty so that, by Lemma 1, the invariant edges of Γ' are exactly the coloops of $\boldsymbol{\mathcal{M}}(G')$. □

By Theorem 3, the set of invariant edges of $\Gamma = (G, \boldsymbol{w})$ can be found by determining first the kernel K of Γ and, then, the set of coloops of $\boldsymbol{\mathcal{M}}(G - K)$. We shall solve the problem of the kernel of an EWG in section 5 and, in the next section, we shall give a linear algorithm for finding the set of coloops of the matroid on a graph.

4. Finding the coloop set. Let $G = (V, E)$ be a simple graph. Bearing in mind that a subset of E is a cocircuit of $\boldsymbol{\mathcal{M}}(G)$ if and only if it is a minimal edge set whose removal decreases the rank of $\boldsymbol{\mathcal{M}}(G)$, one easily obtains the following proposition.

PROPOSITION 2 (see [5]). *An edge of G is a coloop of $\boldsymbol{\mathcal{M}}(G)$ if and only if its removal creates one more bipartite connected component.*

Let e be a coloop of $\boldsymbol{\mathcal{M}}(G)$. The graph $G - e$ has or has not one more connected component than G. By Proposition 2, in the former case e must be a bridge, which we call an *algebraic bridge* of G, and in the latter case, as is shown below, e is an *odd edge*, by which we mean that e is common to all odd cycles of the connected component G containing e.

LEMMA 2. *An edge of a simple graph G is a coloop of $\boldsymbol{\mathcal{M}}(G)$ if and only if it is either an algebraic bridge or an odd edge.*

Proof. The statement is trivial if the graph is bipartite since, by Proposition 2, each coloop of $\boldsymbol{\mathcal{M}}(G)$ is a bridge and vice versa. Consider now a graph G which is not bipartite. Without loss of generality, we assume G is connected. It is sufficient to prove that a coloop e of $\boldsymbol{\mathcal{M}}(G)$ is not a bridge if and only if it is an odd edge. If e is not a bridge, then, by Proposition 2, $G - e$ is bipartite and connected and, hence, every odd cycle of G must contain e; that is, e is an odd edge of G. On the other hand, if e is an odd edge of G, then $G - e$ is connected and contains no odd cycles so that, by Proposition 2, e is a coloop of $\boldsymbol{\mathcal{M}}(G)$. □

EXAMPLE 1 (continued). *The coloops of $\boldsymbol{\mathcal{M}}(G)$ are the two edges missing from the even simple cycle supporting the minimal circulations shown in Figure* 4. *Both of them are odd edges.* □

EXAMPLE 2 (continued). *$\boldsymbol{\mathcal{M}}(G)$ has no coloops (see Figure* 5*).* □

Let $G = (V, E)$ be a simple graph which without loss of generality we assume to be connected. We first show that the problem of finding the set of coloops of $\boldsymbol{\mathcal{M}}(G)$ is polynomial; next, we shall give a linear algorithm based on Lemma 2.

In [14, 16, 17] an $O(|E|)$ algorithm is given to decide whether the incidence vector of a given subset of E is orthogonal to the space of circulations in G. By applying that algorithm to each singleton, one can determine the set of coloops of $\boldsymbol{\mathcal{M}}(G)$ in $O(|E|^2)$ time. In the next two subsections, we give two linear algorithms for finding the algebraic bridges and the odd edges of G; so, by Lemma 2, determining the whole set of coloops of $\boldsymbol{\mathcal{M}}(G)$ requires $O(|E|)$ time.

4.1. Algebraic bridges. Let $G = (V, E)$ be a connected simple graph, and let B be the set of bridges of G. Consider the tree $T = (N, A)$ whose nodes represent the connected components of $G - B$ and whose arcs represent the bridges of G. A node n of T is marked if the corresponding connected component of $G - B$ is not bipartite. If no node of T is marked, then G is bipartite and the bridges of G are all and the only algebraic bridges. Otherwise, there is at least one marked node of T; then, arbitrarily choose a marked node r of T and let T_r be the directed tree obtained by rooting T at r. For each node n of T_r, $n \neq r$, let $par(n)$ be the parent of n in T_r. Of course, a bridge of G is algebraic if and only if the (directed) arc $\langle par(n), n\rangle$ of T_r is such that the subtree of T_r rooted at n contains no marked nodes. Thus, in order to get the algebraic bridges of G, it is sufficient to perform a postorder traversal of T_r[1]: when node n is examined, $n \neq r$; if n is marked, then the edge of G corresponding to the arc $\langle par(n), n\rangle$ is removed from B and the vertex $par(n)$ is marked if it was unmarked. So, the ultimate value of B is exactly the set of algebraic edges of G. Now, since the construction of T and B and the postorder traversal of T_r require $O(|E|)$ time, we have the following theorem.

THEOREM 4. *The set of algebraic bridges of a connected simple graph can be found in time linear in the number of its edges.*

4.2. Odd edges. Let $G = (V, E)$ be a connected simple graph. Trivially, if G is bipartite, then G contains no odd edges. In the case where G is not bipartite, we shall show that the set of odd edges of G can be found in $O(|E|)$. To achieve this, we need the following technical lemmas, the first two of which refer to general properties of the symmetric difference ($\oplus$) of cycles.

LEMMA 3 (see, e.g., [1]). *The symmetric difference of two distinct nondisjoint cycles is a set of edge-disjoint cycles.*

LEMMA 4. *If the symmetric difference of two or more cycles contains an odd number of edges, then the number of such cycles having odd lengths is odd.*

Proof. It easily follows from the fact that, for every two sets C and C', $|C \oplus C'|$ is odd if and only if $|C|$ and $|C'|$ have different parities. □

Let T be the edge set of a spanning tree of G. For each back-edge e (i.e., e not in T), the set $T \cup [e]$ contains exactly one simple cycle; such simple cycles, one for each back-edge, are called the *fundamental cycles* of G with respect to T [18].

LEMMA 5 (see, e.g., page 251 in [1]). *Let T be the edge set of a spanning tree of a simple graph G. Every cycle of G can be expressed as a symmetric difference of fundamental cycles of G with respect to T.*

LEMMA 6. *Let G be a nonbipartite connected simple graph and let T be a spanning tree of G. An edge of G is an odd edge of G if and only if it is in all odd fundamental cycles with respect to T and in no even fundamental cycle with respect to T.*

Proof. ("if") Let e be an edge of G that is in all odd fundamental cycles with respect to T and in no even fundamental cycle with respect to T. Let C be any odd cycle. By Lemma 5, C can be expressed as symmetric difference of fundamental cycles with respect to T, and, by Lemma 4, the number of odd fundamental cycles in its expression is odd so that, since e is in all of them and in no even fundamental cycle with respect to T, e belongs to C.

("only if") Let e be an odd edge of G. Of course e is in all odd fundamental cycles with respect to T. Suppose by contradiction that there is an even fundamental cycle C' with respect to T that contains e. Let C be an odd cycle containing e. By Lemma 3, $C \oplus C'$ contains an odd cycle, say C'', because the lengths of C and C' have different parities. So, since e is in both C and C', e is not in C'', which contradicts the hypothesis that e is in all odd cycles of G. □

From a computational point of view, the fundamental cycles of G with respect to a given spanning tree can be constructed using an $O(|V|^3)$ algorithm (see, e.g., Algorithm 8.10 in [18]). So, by Lemma 6 one can resort to that algorithm to find the set of odd edges of G in $O(|V|^3)$ time. However, we shall use Lemma 6 to work out an algorithm which runs in $O(|E|)$ time. It consists of two phases.

Phase I. Arbitrarily choose a vertex r of G and perform a traversal of G with the depth-first search (DFS) technique to produce

- the edge set T of a directed spanning tree of G,
- the set B of back-edges that create odd fundamental cycles of G with respect to T, and
- a vertex table which, for each vertex v, reports the following information items:
  - the DFS number of v, denoted by $n(v)$;
  - a label, denoted by $col(v)$, which is set to "white" or "black" depending on whether the length of the path from r to v in the spanning tree is even or odd;
  - if $v \neq r$, the parent of v, denoted by $par(v)$;
  - if $v \neq r$, the tree-edge $\langle par(v), v \rangle$, denoted by $arc(v)$.

Phase II. First of all, join a back-edge to *Odd* if it is the unique element of B. Next, in order to decide if a tree-edge e can be joined to *Odd*, compute

- the number of the even fundamental cycles that contain e, denoted by NEC[e], and
- the number of the odd fundamental cycles that contain e, denoted by NOC[e],

as follows. For each vertex u, let $N(u)$ be the set of neighbors of u in G and let $C(u)$ be the set of children of u in T. Then, set (see Figure 6)

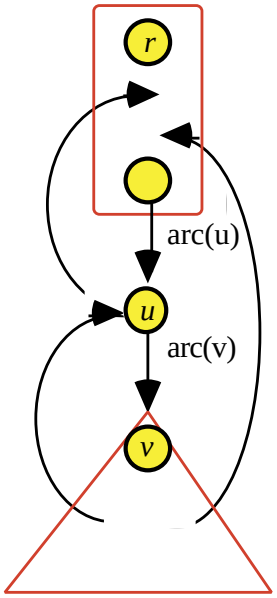


FIG. 6.

$$\text{NEC}[arc(u)] = |P_{even}(u)| + \sum_{v \in C(u)} \text{NEC}[arc(v)] - |S_{even}(u)|, \tag{2}$$

where

$$P_{even}(u) = \{v \in N(u) \colon par(u) \neq v \text{ and } col(v) \neq col(u) \text{ and } n(v) < n(u)\},$$
$$S_{even}(u) = \{v \in N(u) \colon par(v) \neq u \text{ and } col(v) \neq col(u) \text{ and } n(v) > n(u)\},$$

and

$$\text{NOC}[arc(u)] := |P_{odd}(u)| + \sum_{v \in C(u)} \text{NOC}[arc(v)] - |S_{odd}(u)|, \tag{3}$$

where

$$P_{odd}(u) = \{v \in N(u) \colon par(u) \neq v \text{ and } col(v) = col(u) \text{ and } n(v) < n(u)\},$$
$$S_{odd}(u) = \{v \in N(u) \colon par(v) \neq u \text{ and } col(v) = col(u) \text{ and } n(v) > n(u)\}.$$

After calculating the quantities NEC[e] and NOC[e] for each edge e in T, determine the set of odd edges, denoted by *Odd*, as follows (see Lemma 6): for each edge e in T, join e to *Odd* if NEC[e] = 0 and NOC[e] = $|B|$.

The following algorithm details the steps of Phase II.

ALGORITHM 1.

Input: A nonbipartite, connected simple graph $G = (V, E)$, a vertex r of G, T, B and the vertex table of G.

Output: The set *Odd* of odd edges of G.

(1) Set $Odd := \emptyset$. Set $k := |B|$. If $k = 1$, then $Odd := Odd \cup B$.
For each edge e in T, set NEC[e] := NOC[e] := 0.

(2) For each child u of r in T, TRAVERSE (G, u).

(3) For each edge e in T, if $\mathrm{NEC}[e] = 0$ and $\mathrm{NOC}[e] = k$, then add e fo *Odd*.

PROCEDURE TRAVERSE (G, u).

For each neighbor v of u, do:

begin

If v is a child of u, then do:

begin

TRAVERSE (G, v);

$\mathrm{NEC}[arc(u)] := \mathrm{NEC}[arc(u)] + \mathrm{NEC}[arc(v)]$;

$\mathrm{NOC}[arc(u)] := \mathrm{NOC}[arc(u)] + \mathrm{NOC}[arc(v)]$

end;

otherwise, if v is not the parent of u, then do:

Case 1. if $n(v) > n(u)$ and $col(v) \neq col(u)$, then set

$$\mathrm{NEC}[arc(u)] := \mathrm{NEC}[arc(u)] - 1;$$

Case 2. if $n(v) > n(u)$ and $col(v) = col(u)$, then set

$$\mathrm{NOC}[arc(u)] := \mathrm{NOC}[arc(u)] - 1;$$

Case 3. if $n(v) < n(u)$ and $col(v) \neq col(u)$, then set

$$\mathrm{NEC}[arc(u)] := \mathrm{NEC}[arc(u)] + 1;$$

Case 4. if $n(v) < n(u)$ and $col(v) = col(u)$, then set

$$\mathrm{NOC}[arc(u)] := \mathrm{NOC}[arc(u)] + 1$$

end.

THEOREM 5. *Let G be a nonbipartite, connected simple graph. The value of Odd computed by Algorithm 1 with input G and vertex r is exactly the set of odd edges of G.*

Proof. It is sufficient to prove that the quantities $\mathrm{NEC}[e]$ and $\mathrm{NOC}[e]$, for each tree-edge e, equal the number of even fundamental cycles containing e and the number of odd fundamental cycles containing e, respectively. The statement is proven by structural induction.

Basis. Assume that u is a leaf of T. Then, u has no children so that if v is a neighbor of u, then v must be an ancestor of u. If $v = u$, then the self-loop (u, u) contributes to neither $\mathrm{NEC}[arc(u)]$ nor $\mathrm{NOC}[arc(u)]$. If v is a proper ancestor of the parent of u, then $n(v) < n(u)$ and the back-edge (u, v) correctly adds 1 to either $\mathrm{NEC}[arc(u)]$ or $\mathrm{NOC}[arc(u)]$.

Inductive step. Let u be not a leaf of T and assume the statement holds for each one of the children of u. Thus, if v is a child of u, then values of both $\mathrm{NEC}[arc(v)]$ and $\mathrm{NOC}[arc(v)]$ are right. It is then easily seen that, by formulae (2) and (3), the statement also holds for u. □

From the complexity-theoretic point of view, it is easily seen that the time of Algorithm 1 is dominated by the time required by the DFS traversal and, hence, is $O(|E|)$. So, by Theorem 5 one has the following corollary.

COROLLARY 1. *Let $G = (V, E)$ be a nonbipartite, connected simple graph. The set of odd edges of G can be found in $O(|E|)$ time.*

5. Finding the kernel. Let $\Gamma = (G, \boldsymbol{w})$ be an EWG with $G = (V, E)$ and kernel K. If the zero set Z of $\boldsymbol{w}$ is empty, then K is empty too and we are done. Assume that

Z is not empty. If G is bipartite, Gusfield [7] proved that K equals the set of directed edges joining strongly connected components of the *mixed graph* $G(Z)$ obtained from G by directing all the edges in Z from one side of the bipartition to the other one so that it can be computed in time linear in the size of G. In this section we show that, even in the case where G is not bipartite, the kernel of Γ can be computed in time linear in the size of G.

With Γ we associate a bipartite EWG $\Gamma' = (G', \boldsymbol{w}')$, which we call a *bipartite EWG associated* with Γ. The graph $G' = (V', E')$ is constructed as follows. Let B be a maximal bipartite partial graph of G and let $\{V_1, V_2\}$ be a bipartition of V such that each edge of B has one end in V_1 and the other end in V_2. Let $\overline{V}$ be a "copy" of V, that is, $\overline{V} \cap V = \emptyset$ and $|\overline{V}| = |V|$. If v is a vertex of G, then by $\bar{v}$ we denote the copy of v. The vertex set of G' is taken to be $V' = V \cup \overline{V}$, and the edge set of G' is taken to be

$$E' = \bigcup_{e \in E} f(e),$$

where f is function defined on E as follows:

— if e is a self-loop, say (v, v), then $f(e) = \{(v, \bar{v})\}$;
— if $e = (u, v)$ is an edge of B, then $f(e) = \{(u, v), (\bar{u}, \bar{v})\}$;
— if $e = (u, v)$ is neither a self-loop nor an edge of B, then $f(e) = \{(u, \bar{v}), (\bar{u}, v)\}$.

The set $f(e)$ will be referred to as the *image* of e in G'. Let $V_1' = V_1 \cup \overline{V}_2$ and $V_2' = \overline{V}_1 \cup V_2$, where $\overline{V}_1 = \{\bar{v} : v \in V_i\}$, $i = 1, 2$. The graph G' is bipartite and the partition $\{V_1', V_2'\}$ of V' is such that each edge of G' has one end in V_1' and the other end in V_2'. Furthermore, G' is connected if and only if G is not bipartite. Finally, to each edge e' of G' we assign the weight $w'(e') = w(e)$, where e is the edge of G for which $e' \in f(e)$. Let $\mathbf{A}'$ be the incidence matrix of G' and let $\boldsymbol{b}' = (b'(v'))_{v' \in V'}$, where $b'(v')$ equals the sum of the weights $w'(e')$ of the edges of G' incident to v'. Consider the equation system

$$\mathbf{A}'\boldsymbol{x}' = \boldsymbol{b}'. \tag{4}$$

For every edge e' of G', let $L[x'(e')]$ and $U[x'(e')]$ denote the minimum and the maximum of the variable $x'(e')$ over the nonnegative solutions of equation system (4), respectively. Moreover, for every edge e of G, let $L[f(e)]$ and $U[f(e)]$ denote the minimum and the maximum of the expression $\sum_{e' \in f(e)} x'(e')$ over the nonnegative solutions of equation system (4), respectively. First, observe that if $\boldsymbol{x}'$ is a (nonnegative) solution of equation system (4), then a (nonnegative) solution $\boldsymbol{x}''$ of equation system (4) can be obtained by setting for each edge e' of G'

$$x''(e') = x'(e') \text{ if } \{e'\} \text{ is the image of a self-loop of } G$$

and

$$x''(e') = x'(e'') \text{ if } \{e', e''\} \text{ is the image of an edge of } G \text{ that is not a self-loop.}$$

It follows that, if $\{e', e''\}$ is the image of an edge of G that is not a self-loop, then

$$L[x'(e')] = L[x'(e'')] \quad \text{and} \quad U[x'(e')] = U[x'(e'')]. \tag{5}$$

Second, if $\boldsymbol{x}$ is a (nonnegative) solution of equation system (1), then a (nonnegative) solution $\boldsymbol{x}'$ of equation system (4) can be obtained by setting for each edge e' of G'

$$x'(e') = x(e), \text{ where } e \text{ is the edge of } G \text{ whose image } f(e) \text{ contains } e'.$$

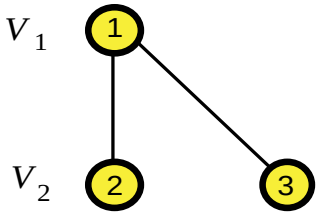


FIG. 7.

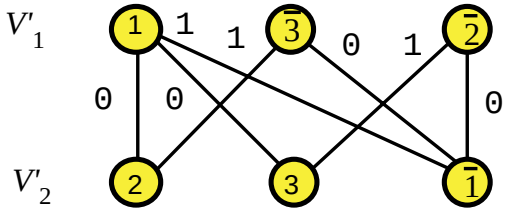


FIG. 8.

On the other hand, if $\boldsymbol{x}'$ is a (nonnegative) solution of equation system (4), then a (nonnegative) solution $\boldsymbol{x}$ of equation system (1) can be obtained by setting for each edge e of G

$$x(e) = \left[\sum_{e' \in f(e)} x'(e')\right] \Big/ |f(e)|.$$

Therefore, one has

$$L[x(e)] = (1/|f(e)|)L[f(e)] \quad \text{and} \quad U[x(e)] = (1/|f(e)|)U[f(e)]. \tag{6}$$

EXAMPLE 2 (continued). *By choosing as the maximal bipartite partial graph of G the graph shown in Figure* 7*, we associate with Γ the bipartite EWG $\Gamma' = (G', \boldsymbol{w}')$ shown in Figure* 8.

The general expression of a nonnegative solution of equation system (4) *is*

$$\begin{aligned} x'(1,2) &= x'(\bar{1},\bar{3}) = \mu, \\ x'(1,3) &= x'(\bar{1},\bar{2}) = \nu, \\ x'(1,\bar{1}) &= 1 - \mu - \nu, \\ x'(2,\bar{3}) &= 1 - \mu, \\ x'(\bar{2},3) &= 1 - \nu, \end{aligned}$$

where μ and ν are bounded as shown in Figure 9*. At this point, it is easy to check formulae* (5) *and* (6). □

We now state some technical results to relate the kernels of Γ and Γ'.

LEMMA 7. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG and let $\Gamma' = (G', \boldsymbol{w}')$ be a bipartite EWG associated with Γ. An edge e of G is an invariant edge of Γ if and only if $L[f(e)] = U[f(e)]$, where $f(e)$ is the image of e in G'.*

Proof. The proof follows from formula (6). □

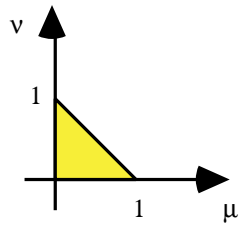


FIG. 9.

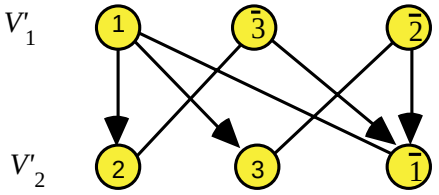


FIG. 10.

LEMMA 8. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG and let $\Gamma' = (G', \boldsymbol{w}')$ be a bipartite EWG associated with Γ. An edge e of G belongs to the kernel of Γ if and only if its image in G' is contained in the kernel of Γ'.*

Proof. If e is a self-loop of G, then the statement immediately follows from formula (6) and Lemma 7. We now prove the statement in the case where e is not a self-loop and $f(e) = \{e', e''\}$.

("if") If both e' and e'' belong to the kernel of Γ', then $x'(e') = x'(e'') = 0$ for every solution $\boldsymbol{x}'$ of equation system (4). Therefore, $L[x'(e') + x'(e'')] = U[x'(e') + x'(e'')] = 0$ and the statement follows from formula (6).

("only if") If e belongs to the kernel of Γ, then, by formula (6), one has

$$x'(e') + x'(e'') = 0$$

for every nonnegative solution $\boldsymbol{x}'$ of equation system (4). By the nonnegativity of $\boldsymbol{x}'$, $x'(e') = x'(e'') = 0$, which proves that both e' and e'' belong to the kernel of Γ'. □

COROLLARY 2. *Let $\Gamma = (G, \boldsymbol{w})$ be an EWG and let $\Gamma' = (G', \boldsymbol{w}')$ be a bipartite EWG associated with Γ'. An edge of G belongs to the kernel of Γ if and only if an element of its image in G' belongs to the kernel of Γ'.*

Proof. The proof follows from Lemma 8 and formula (5). □

EXAMPLE 2 (continued). *The zero set of $\boldsymbol{w}'$ is $Z' = \{(1,2), (1,3), (\bar{1}, \bar{2}), (\bar{1}, \bar{3})\}$. The mixed graph $G'(Z')$ is strongly connected (see Figure 10). So, the kernel of Γ' is empty. By Corollary 2, the kernel of Γ is empty.* □

THEOREM 6. *The kernel of an EWG can be found in time linear in the size of G.*

Proof. Let $\Gamma = (G, \boldsymbol{w})$ be an EWG and let $\Gamma' = (G', \boldsymbol{w}')$ be a bipartite EWG associated with Γ. If G is bipartite, then the statement was proven by Gusfield [7]. Otherwise, since G' is bipartite, the kernel K' of Γ' can be found in time linear in the size of G' and, hence, of G. So, it is sufficient to prove that both constructing G' and determining K from K' take a linear time. In order to construct G', we perform a DFS traversal of G, which allows us to find both a maximal bipartite partial graph B of G and the nontree edges that create odd cycles when added to B. When an edge

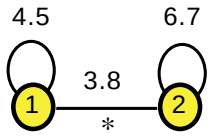


FIG. 11.

e' of G' is created, we get e' to point to the edge e of G for which $e' \in f(e)$. Finally, by Corollary 2, the set K can be obtained as follows. Initially, each edge e of G is unmarked. For each element e' of K', if the edge e of G that e' points to is unmarked, then e is marked and added to K. □

6. Security of statistical data. In the security analysis of statistical data [4, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17], EWGs and, more in general, weighted hypergraphs can be used to control the amount of information that is implicitly released when statistical data are made public, in order to avoid disclosure of confidential data. We now illustrate this application by discussing a typical case. Suppose that we are given a data set $\{b_i : i \in I\}$, where b_i is the value of a confidential variable b of nonnegative real type (e.g., salary) for individual i in the population I. The sum of values of b over a subset I' of I is called *sensitive* if I' contains exactly one individual. Now, given a statistical summary $\sigma = \{b(v) : v \in V\}$ where $b(v)$ is the sum of b over a subset $I(v)$ of I containing at least two people, for each v in V, the problem that naturally arises consists of checking that no sensitive sum is implicitly released. We now present a graph-theoretic approach to this problem. Let $I_\sigma = \cup_{v \in V} I(v)$, which we call the set of individuals *covered* by σ. The *basic partition* of I_σ is the coarsest partition of I_σ such that each $I(v)$ can be obtained as the union of one or more classes of the partition. A class J of the basic partition of I_σ will be indexed by the set $e = \{v \in V : J \subseteq I(v)\}$. If E is the index set of the classes of the basic partition of I_σ, the pair $G = (V, E)$ defines a hypergraph where hyperedge e is incident to vertex v if and only if v belongs to e. Consider the weighted hypergraph $\Gamma = (G, \boldsymbol{w})$, where, for each hyperedge e of G, $w(e)$ is given by the sum of the values of the variable b over the class $J(e)$ of the basic partition of I_σ indexed by e; that is,

$$w(e) = \sum_{i \in J(e)} b_i.$$

Finally, a hyperedge e of G is *marked* if $|J(e)| = 1$. Then, no sensitive sum is implicitly released given σ if and only if no invariant hyperedge of Γ is marked. If this is the case, the statistical summary σ is said to be *safe*. Since the invariant edges of an EWG can be found in linear time, one has that, if G is a graph, then one can decide whether σ is or is not safe in linear time, too.

EXAMPLE 3. *Consider five individuals with salaries* $b_1 = 2.0$, $b_2 = 2.5$, $b_3 = 3.8$, $b_4 = 3.7$, *and* $b_5 = 3.0$. *Suppose that the two sums* $b_1 + b_2 + b_3$ *and* $b_3 + b_4 + b_5$ *are made public. Let* $\sigma_1 = \{b_1 + b_2 + b_3, b_3 + b_4 + b_5\}$. *The set of individuals covered by* σ_1 *is* $\{1, \ldots, 5\}$ *and its basic partition consists of the three classes* $\{1, 2\}$, $\{3\}$, *and* $\{4, 5\}$. *Thus, the weighted hypergraph* Γ_1 *associated with* σ_1 *is the EWG shown in Figure* 11, *where the edge* $(1, 2)$ *is marked. Since the set of invariant edges of* Γ_1 *turns out to be empty,* σ_1 *is safe. Next, suppose that the sum* $b_1 + b_2 + b_4 + b_5$ *is also made public.*

Let $\sigma_2 = \{b_1 + b_2 + b_3, b_3 + b_4 + b_5, b_1 + b_2 + b_4 + b_5\}$. *Again, the set of individuals covered by* σ_2 *is* $\{1, \ldots, 5\}$ *and its basic partition consists of the three classes* $\{1, 2\}$,

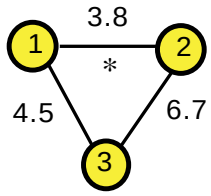


FIG. 12.

$\{3\}$, *and* $\{4,5\}$. *The weighted hypergraph* Γ_2 *associated with* σ_2 *is the EWG shown in Figure* 12, *where the edge* $(1,2)$ *is marked. Since each edge is an invariant edge of* Γ_2, σ_2 *is not safe (and the salary* b_3 *is unprotected).* □

Let $\sigma = \{b(v)\colon v \in V\}$ be a statistical summary of the data set $\{b_i\colon i \in I\}$ and let $\Gamma = (G, \boldsymbol{w})$ be the associated weighted hypergraph. It is worth noting that if v is a "leaf" of G, that is, if v belongs to exactly one hyperedge e of G, then the class of the basic partition of I_σ indexed by e coincides with $I(v)$ so that $w(e) = b(v)$; furthermore, the hyperedge e is definitely an invariant hyperedge of Γ, and, since $|I(v)| > 1$, it is not marked. As we are interested in checking the existence of marked invariant hyperedges of Γ (if any), we can reduce Γ by deleting all leaves of G and their incident hyperedges. Let $\Gamma' = (G', \boldsymbol{w}')$ be the resulting weighted hypergraph. Of course, σ is safe if and only if no invariant hyperedge of Γ' is sensitive. We now show that if σ is a two-dimensional table with suppressions, then G' is always a graph so that one can decide whether σ is or is not safe in linear time. Let σ be obtained from a complete two-dimensional table T by suppressing all sensitive cells ("primary suppressions") as well as additional (internal or marginal) cells to exclude the possibility of arriving at the contents of sensitive cells by indirect methods ("complementary suppressions"). Denote by

— $T(r,c)$ the value of internal cell (r,c), $1 \le r \le m$, and $1 \le c \le n$,
— $T(r,+)$ the rth row total, $1 \le r \le m$, and
— $T(+,c)$ the cth column total, $1 \le c \le n$.

Assume that each $T(r,c)$ is the sum of the values of a confidential variable of nonnegative real type over the set $I(r,c)$ of individuals. So, a cell (r,c) of T is sensitive if $|I(r,c)| = 1$. We first detail the structure of the weighted hypergraph $\Gamma = (G, \boldsymbol{w})$ associated with σ and then show that the reduction of Γ results in an EWG. Let U, R, and C be the set of unsuppressed internal cells, the set of marginal cells corresponding to unsuppressed row totals, and the set of marginal cells corresponding to unsuppressed column totals, respectively. Then the vertex set of G is

$$V = U \cup R \cup C.$$

Let $S = \{(r,c) \notin U\colon r \notin R \text{ and } c \notin C\}$. Moreover, for each $r \in R$, let $C_r = \{c \notin C\colon (r,c) \notin U\}$; analogously, for each $c \in C$, let $R_c = \{r \notin R\colon (r,c) \notin U\}$. Then, the set of individuals covered by σ is $I_\sigma = \cup_{(r,c)\notin S} I(r,c)$ and the basic partition of I_σ contains

— one class $I(r,c)$ for each (r,c) in U and for each (r,c) not in U with $r \in R$ and $c \in C$,
— one class $\cup_{c\in C_r} I(r,c)$ for each $r \in R$ with $C_r \neq \emptyset$, and
— one class $\cup_{r\in R_c} I(r,c)$ for each $c \in C$ with $R_c \neq \emptyset$.

Recall that the hyperedges of G are the indices of these classes. The hyperedge e

indexing a class such as $I(r,c)$ is

$$\begin{aligned}
&e=\{(r,c),(r,+),(+,c)\} && \text{if } (r,c)\in U, r\in R, c\in C,\\
&e=\{(r,c),(r,+)\} && \text{if } (r,c)\in U, r\in R, c\notin C,\\
&e=\{(r,c),(+,c)\} && \text{if } (r,c)\in U, r\notin R, c\in C,\\
&e=\{(r,c)\} && \text{if } (r,c)\in U, r\notin R, c\notin C,\\
&e=\{(r,+),(+,c)\} && \text{if } (r,c)\notin U, r\in R, c\in C,
\end{aligned}$$

and $w(e)$ is always set to $T(r,c)$. For the hyperedge e indexing a class such as $\cup_{c\in C_r} I(r,c)$, one has $e=\{(r,+)\}$ and

$$w(e)=\sum_{c\in C_r} T(r,c),$$

and for the hyperedge e indexing a class such as $\cup_{r\in R_c} I(r,c)$, one has $e=\{(+,c)\}$ and

$$w(e)=\sum_{r\in R_c} T(r,c).$$

At this point, it should be clear that the leaves of G are all and the only vertices of the type (r,c), of the type $(r,+)$ with $r\in R$ and $C_r=\{1,\dots,n\}$, and of the type $(+,c)$ with $c\in C$ and $R_c=\{1,\dots,m\}$. Let L be the set of leaves of G and let $R'=R-L$ and $C'=C-L$. After deleting all the leaves of the hypergraph G, we remain with the hypergraph $G'=(V',E')$ whose hyperedges are incident to at most two vertices. More precisely, one has that $V'=R'\cup C'$ and E' consists of the edges

$$\begin{aligned}
&\{(r,+),(+,c)\} && \text{if } (r,c)\notin U, r\in R, c\in C,\\
&\{(r,+)\} && \text{with } r\in R',\\
&\{(+,c)\} && \text{with } c\in C'.
\end{aligned}$$

To sum up, the reduction of the weighted hypergraph associated with σ is an EWG and, therefore, the safety of σ can be tested in linear time.

EXAMPLE 4. *Consider the table of Figure* 13 *whose entries are assumed to be nonnegative reals. Suppose that the following cells*

$$(1,1),(1,2),(2,1),(2,2),(2,3),(4,4)$$

are all sensitive. The table of Figure 14 *is obtained from the table of Figure* 13 *by suppressing all the six sensitive cells and the additional cells* $(3,3)$, $(3,+)$, $(4,+)$, *and* $(+,4)$.

The reduced weighted hypergraph $\Gamma'=(G',\boldsymbol{w}')$ *associated with the table of Figure* 14 *is the EWG shown in Figure* 15. *The invariant edges of* Γ' *are the edge joining the vertices* $(2,+)$ *and* $(+,3)$ *and the self-loop at vertex* $(+,3)$. *One of these two edges is marked and, therefore, the table of Figure* 14 *is not safe.* ∎

7. Closing remarks. We solved the problem of finding the set of invariant edges of an EWG under the assumption that edge weights are nonnegative reals. The case where edge weights are nonnegative integers is an open problem. However, if the underlying graph of the EWG is bipartite, then Gusfield's algorithm still holds owing to the total unimodularity of the incidence matrix.

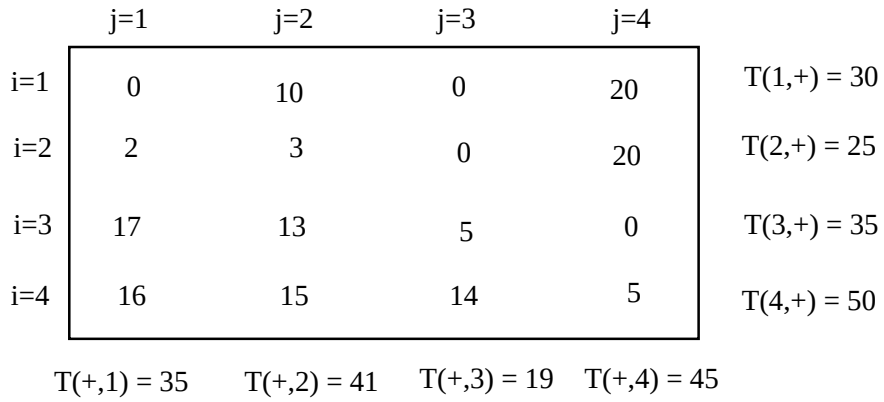


FIG. 13.

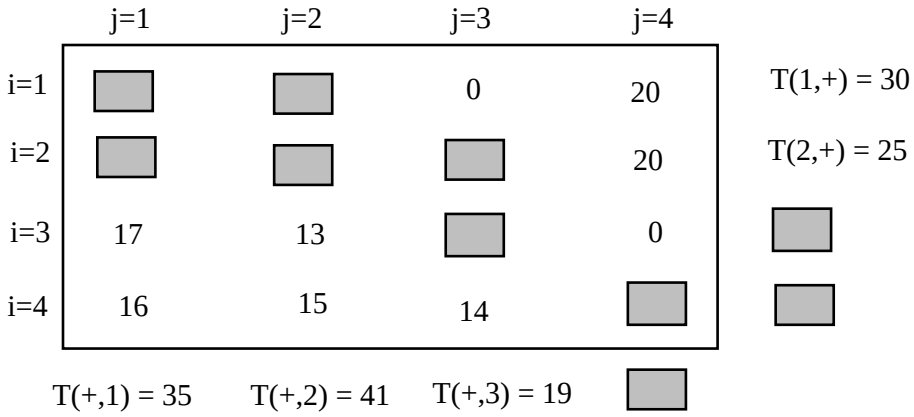


FIG. 14.

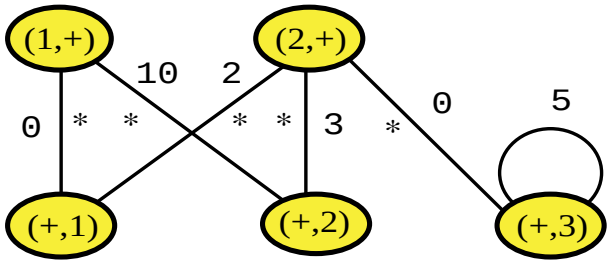


FIG. 15.

A natural generalization of the problem dealt with in this paper is the search of invariant edges of an edge-weighted hypergraph. It should be noted that mutatis mutandis Theorem 3 (see section 3) applies to edge-weighted hypergraphs, too. So, in order to find the invariant edges of an edge-weighted hypergraph $(G, \boldsymbol{w})$, we have to devise a procedure for computing its kernel, say K, and the coloops of the matroid $\boldsymbol{\mathcal{M}}(G-K)$. It should be clear that, in order to find to coloops of $\boldsymbol{\mathcal{M}}(G-K)$, we need a formula for the rank of the incidence matrix of G. At present, such a formula is

known only for special classes of hypergraphs, e.g., for the class of connected uniform hypergraphs [2].

Acknowledgments. The authors wish to thank M. Moscarini for her valuable suggestions and the two referees for their comments.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft, and J.F. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1987.
[2] A. Björner and J. Karlander, *The mod p rank of incidence matrices for connected uniform hypergraphs*, European J. Combin., 14 (1993), pp. 151–155.
[3] A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler, *Oriented Matroids*, Cambridge University Press, Cambridge, MA, 1993.
[4] L. Branković, P. Horak, and M. Miller, *An optimization problem in statistical databases*, SIAM J. Disc. Math., 13 (2000), pp. 346–353.
[5] M. Conforti and M.R. Rao, *Some new matroids on graphs*: *Cut sets and the max cut problem*, Math. Oper. Res., 12 (1987), pp. 193–204.
[6] G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.
[7] D. Gusfield, *A graph theoretic approach to statistical data security*, SIAM J. Comput., 17 (1988), pp. 552–571.
[8] P. Horak, L. Brankovic, and M. Miller, *A combinatorial problem in database security*, Discrete Appl. Math., 91 (1999), pp. 119–126.
[9] T.-S. Hsu and M.Y. Kao, *Security problems for statistical databases with general cell suppression*, in Proceedings of the 9th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, 1997, pp. 155–164.
[10] M.-Y. Kao, *Data security equals graph connectivity*, SIAM J. Discrete Math., 9 (1996), pp. 87–100.
[11] M.Y. Kao, *Efficient detection and protection of information in cross-tabulated tables* II: *Minimal linear invariants*, J. Comb. Optim., 1 (1997), pp. 187–202.
[12] M.Y. Kao, *Total protection of analytic-invariant information in cross-tabulated tables*, SIAM J. Comput., 26 (1997), pp. 231–242.
[13] M.-Y. Kao and D. Gusfield, *Efficient detection and protection of information in cross tabulated tables* I: *Linear invariant test*, SIAM J. Discrete Math., 6 (1993), pp. 460–476.
[14] F.M. Malvestuto, *A universal-scheme approach to statistical databases containing homogeneous summary tables*, ACM Trans. Database Systems, 18 (1993), pp. 678–708.
[15] F.M. Malvestuto and M. Moscarini, *Suppressing marginal totals from a two-dimensional table to protect sensitive information*, Statist. Comput., 7 (1997), pp. 101–114.
[16] F.M. Malvestuto and M. Moscarini, *An audit expert for large statistical databases*, in Proceedings of the First Conference on Statistical Data Protection, Lisbon, Eurostat, 1998, pp. 29–43.
[17] F.M. Malvestuto, M. Moscarini, and M. Rafanelli, *Suppressing marginal cells to protect sensitive information in a two-dimensional statistical table*, in Proceedings of the 10th Symposium on Principles of Database Systems, Denver, CO, 1991, pp. 252–258.
[18] E.M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
[19] C. Van Nuffelen, *On the incidence matrix of a graph*, IEEE Trans. Circuits and Systems, 23 (1976), p. 572.
[20] D.J.A. Welsh, *Matroids: Fundamental concepts*, in Handbook of Combinatorics, Vol. 1, R.L. Graham, M. Groötschel, and L. Lovász, eds., North-Holland, Amsterdam, 1995.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1456–1478

CHARACTERIZATIONS OF 1-WAY QUANTUM FINITE AUTOMATA*

ALEX BRODSKY† AND NICHOLAS PIPPENGER†

Abstract. The 2-way quantum finite automaton introduced by Kondacs and Watrous [*Proceedings of the* 38*th Annual Symposium on Foundations of Computer Science*, 1997, IEEE Computer Society, pp. 66–75] can accept nonregular languages with bounded error in polynomial time. If we restrict the head of the automaton to moving classically and to moving only in one direction, the acceptance power of this 1-way quantum finite automaton is reduced to a proper subset of the regular languages.

In this paper we study two different models of 1-way quantum finite automata. The first model, termed measure-once quantum finite automata, was introduced by Moore and Crutchfield [*Theoret. Comput. Sci.*, 237 (2000), pp. 275–306], and the second model, termed measure-many quantum finite automata, was introduced by Kondacs and Watrous [*Proceedings of the* 38*th Annual Symposium on Foundations of Computer Science*, 1997, IEEE Computer Society, pp. 66–75].

We characterize the measure-once model when it is restricted to accepting with bounded error and show that, without that restriction, it can solve the word problem over the free group. We also show that it can be simulated by a probabilistic finite automaton and describe an algorithm that determines if two measure-once automata are equivalent.

We prove several closure properties of the classes of languages accepted by measure-many automata, including inverse homomorphisms, and provide a new necessary condition for a language to be accepted by the measure-many model with bounded error. Finally, we show that piecewise testable sets can be accepted with bounded error by a measure-many quantum finite automaton, introducing new construction techniques for quantum automata in the process.

Key words. quantum finite automata, quantum computation, automata theory

AMS subject classifications. 68Q05, 68Q10, 68Q45, 68Q70

PII. S0097539799353443

1. Introduction.

In 1997 Kondacs and Watrous [9] showed that a 2-way quantum finite automaton (2QFA) could accept the language $L = a^n b^n$ in linear time with bounded error. The ability of the reading head to be in a superposition of locations rather than in a single location at any time during the computation gives the 2QFA its power. Even if we restrict the head of a 2QFA from moving left, we can still construct a 2QFA that can accept the language $L' = \{x \in \{a,b\}^* \mid |x|_a = |x|_b\}$ in linear time with bounded error. However, if we restrict the head of a 2QFA to moving right on each transition, we get the 1-way quantum finite automaton (1QFA) of Kondacs and Watrous [9], which, when accepting with bounded error, can accept only a proper subset of the regular languages.

If the reading head is classical, then quantum mechanical evolution hinders language acceptance, restricting the set of languages accepted by 1QFAs with bounded error to a proper subset of the regular languages [9].

During its computation, a 1QFA performs measurements on its configuration. Since the acceptance capability of a 1QFA depends on the measurements that the QFA may perform during the computation, we investigate two models of 1QFAs that differ only in the type of measurement that they perform during the computation.

*Received by the editors March 12, 1999; accepted for publication (in revised form) March 26, 2002; published electronically August 1, 2002.

http://www.siam.org/journals/sicomp/31-5/35344.html

†Department of Computer Science, University of British Columbia, Canada (abrodsky@cs.ubc.ca, nicholas@cs.ubc.ca).

The first model, termed measure-once quantum finite automata (MO-QFAs), is similar to the one introduced by Moore and Crutchfield [12]. The second model, termed measure-many quantum finite automata (MM-QFAs), is similar to the one introduced by Kondacs and Watrous [9] and is more complex than the MO-QFA. The main difference between the two models is that a MO-QFA performs one measurement at the end of its computation, while a MM-QFA performs a measurement after every transition. This makes the measure-many model more powerful than the measure-once model, where the power of a model refers to the acceptance capability of the corresponding automata.

First, we present results dealing with MO-QFAs. We show that the class of languages accepted by MO-QFAs with bounded error is exactly the class of group languages. Consequently, this class of languages accepted by MO-QFAs is closed under inverse homomorphisms, word quotients, and Boolean operations. We show that MO-QFAs that do not accept with bounded error can accept nonregular languages and, in particular, can solve the word problem over the free group. We also describe an algorithm that determines if two MO-QFAs are equivalent and prove that probabilistic finite automata (PFAs) can simulate MO-QFAs.

Second, we shift our focus to MM-QFAs. We show that the classes of languages accepted by these automata are closed under complement, inverse homomorphisms, and word quotients. We prove by example that the class of languages accepted by MM-QFAs with bounded error is not closed under homomorphisms and prove a necessary condition for membership within this class. We also relate the sufficiency of this condition to the question of whether the class is closed under Boolean operations. Finally, we show, by construction, that MM-QFAs can accept piecewise testable sets with bounded error and introduce novel concepts for constructing MM-QFAs.

The rest of the paper is organized in the following way: section 2 contains the definitions of the quantum automata and background information, section 3 discusses measure-once quantum finite automata, section 4 discusses measure-many quantum finite automata, and section 5 summarizes.

2. Definitions and background.

2.1. Definition of an MO-QFA.

An MO-QFA is defined by a 5-tuple:

$$M = (Q, \Sigma, \delta, q_0, F),$$

where Q is a finite set of states; Σ is a finite input alphabet with an end-marker symbol \$; δ is the transition function

$$\delta : Q \times \Sigma \times Q \to \mathbb{C}$$

that represents the probability density amplitude that flows from state q to state q' upon reading symbol σ; the state q_0 is the initial configuration of the system; and F is the set of accepting states. For all states $q_1, q_2 \in Q$ and symbols $\sigma \in \Sigma$ the function δ must be unitary, thus satisfying the condition

$$\sum_{q' \in Q} \overline{\delta(q_1, \sigma, q')}\delta(q_2, \sigma, q') = \begin{cases} 1, & q_1 = q_2, \\ 0, & q_1 \neq q_2. \end{cases} \tag{2.1}$$

We assume that all input is terminated by the end-marker \$; this is the last symbol read before the computation terminates. At the end of a computation, M measures

its configuration; if it is in an accepting state, then it accepts, otherwise it rejects. This definition is equivalent to that of the QFA defined by Moore and Crutchfield [12].

The configuration of M is a linear superposition of states and is represented by an n-dimensional complex unit vector, where $n = |Q|$. This vector is denoted by

$$|\Psi\rangle = \sum_{i=1}^{n} \alpha_i |q_i\rangle,$$

where $\{|q_i\rangle\}$ is the set of orthonormal basis vectors corresponding to the states of M. The coefficient α_i is the probability density amplitude of M being in state q_i. Since $|\Psi\rangle$ is a unit vector, it follows that $\sum_{i=1}^{n} |\alpha_i|^2 = 1$.

The transition function δ is represented by a set of unitary matrices $\{U_\sigma\}_{\sigma\in\Sigma}$ where U_σ represents the unitary transitions of M upon reading symbol σ. If M is in configuration $|\Psi\rangle$ and reads symbol σ, then the new configuration of M is denoted by

$$|\Psi'\rangle = U_\sigma |\Psi\rangle = \sum_{q_i,q_j\in Q} \alpha_i \delta(q_i, \sigma, q_j) |q_j\rangle.$$

Measurement is represented by a diagonal zero-one projection matrix P where $P_{ii} = [q_i \in F]$. The probability of M accepting string x is defined by

$$p_M(x) = \langle \Psi_x | P | \Psi_x \rangle = \| P|\Psi_x\rangle \|^2,$$

where $|\Psi_x\rangle = U(x)|q_0\rangle = U_{x_n} U_{x_{n-1}} \dots U_{x_1} |q_0\rangle$.

2.2. Definition of an MM-QFA. An MM-QFA is defined by a 6-tuple:

$$M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej}),$$

where Q is a finite set of states, Σ is a finite input alphabet with an end-marker symbol \$, δ is a unitary transition function of the same form as for an MO-QFA, and the state q_0 is the initial configuration of M. The set Q is partitioned into three subsets: Q_{acc} is the set of halting accepting states, Q_{rej} is the set of halting rejecting states, and Q_{non} is the set of nonhalting states.

The operation of an MM-QFA is similar to that of an MO-QFA except that after every transition, M measures its configuration with respect to the three subspaces that correspond to the three subsets Q_{non}, Q_{acc}, and Q_{rej}: $E_{non} = \mathit{Span}(\{|q\rangle \mid q \in Q_{non}\})$, $E_{acc} = \mathit{Span}(\{|q\rangle \mid q \in Q_{acc}\})$, and $E_{rej} = \mathit{Span}(\{|q\rangle \mid q \in Q_{rej}\})$. If the configuration of M is in E_{non}, then the computation continues; if the configuration is in E_{acc}, then M accepts, otherwise it rejects. After every measurement the superposition collapses into the measured subspace and is renormalized.

Just like MO-QFAs, the configuration of an MM-QFA is represented by a complex n-dimensional vector, the transition function is represented by unitary matrices, and measurement is represented by diagonal zero-one projection matrices that project the vector onto the respective subspaces.

The definition of an MM-QFA is almost identical to the definition by Kondacs and Watrous in [9]. The only difference is that we require only one end-marker at the end of the tape, rather than two end-markers at the start and end of the tape; this does not affect the acceptance power of the automaton; see Appendix A for further details.

Since M can have a nonzero probability of halting partway through the computation, it is useful to keep track of the cumulative accepting and rejecting probabilities.

Therefore, in some cases we use the representation of Kondacs and Watrous [9] that represents the state of M as a triple $(|\Psi\rangle, p_{acc}, p_{rej})$, where p_{acc} and p_{rej} are the cumulative probabilities of accepting and rejecting. The evolution of M on reading symbol σ is denoted by

$$(P_{non}|\Psi'\rangle,\ p_{acc} + \|P_{acc}|\Psi'\rangle\|^2,\ p_{rej} + \|P_{rej}|\Psi'\rangle\|^2),$$

where $|\Psi'\rangle = U_\sigma|\Psi\rangle$, and P_{acc}, P_{rej}, and P_{non} are the diagonal zero-one projection matrices that project the configuration onto the nonhalting, accepting, and rejecting subspaces.

2.3. Language acceptance. A QFA M is said to accept a language L with cut-point λ if for all $x \in L$, the probability of M accepting x is greater than λ, and if for all $x \notin L$, the probability of M accepting x is at most λ. A QFA M accepts L with bounded error if there exists an $\epsilon > 0$ such that for all $x \in L$ the probability of M accepting x is greater than $\lambda + \epsilon$ and for all $x \notin L$ the probability of M accepting x is less than $\lambda - \epsilon$. We call ϵ the margin.

We partition the languages accepted by QFAs into several natural classes. Let the class $\mathbf{RMO}_\epsilon$ be the set of languages accepted by an MO-QFA with a margin of at least ϵ. Let the restricted class of languages, $\mathbf{RMO} = \cup_{\epsilon>0}\mathbf{RMO}_\epsilon$, be the set of languages accepted by an MO-QFA with bounded error, and let the unrestricted class of languages, $\mathbf{UMO} = \mathbf{RMO}_0$, be the set of languages accepted by an MO-QFA with unbounded error. We define the languages classes $\mathbf{RMM}_\epsilon$, $\mathbf{RMM}$, and $\mathbf{UMM}$ accepted by an MM-QFA in a similar fashion.

Since the cut-point of a QFA can be arbitrarily raised or lowered, we could without loss of generality fix the cut-point to be $\frac{1}{2}$. However, for the purposes of presentation we use the general cut-point definition stated above.

2.4. Reversible finite automata. Unitary operations are reversible, thus QFAs bear a strong resemblance to various variants of reversible finite automata. A group finite automaton (GFA) is a deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ with the restriction that for every state $q \in Q$ and every input symbol $\sigma \in \Sigma$ there exists exactly one state $q' \in Q$ such that $\delta(q', \sigma) = q$, i.e., δ is a complete one-to-one function and the automaton derived from M by reversing all transitions is deterministic.

A reversible finite automata (RFA) is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that for every state $q \in Q$ and for every symbol $\sigma \in \Sigma$ there is at most one state $q' \in Q$ such that $\delta(q', \sigma) = q$, or, if there exist distinct states $q_1, q_2 \in Q$ and symbol $\sigma \in \Sigma$ such that $\delta(q_1, \sigma) = q = \delta(q_2, \sigma)$, then $\delta(q, \Sigma) = \{q\}$. The latter type of state is called a spin state because once an RFA enters it, it will never leave it. This definition is equivalent to the one used by Ambainis and Freivalds [3] and is an extension of Pin's [17] definition.

2.5. Previous work. Moore and Crutchfield [12] introduced a variant of the MO-QFA model and investigated the model in terms of quantum regular languages (QRLs). They showed several closure properties including closure under inverse homomorphisms and derived a method for bilinearizing the representation of an MO-QFA that transforms it into a generalized stochastic system.

Kondacs and Watrous [9] introduced a variant of the MM-QFA that was derived from their 2QFA model. Using a technique similar to Rabin's [18], Kondacs and Watrous proved that 1QFAs that accept with bounded error are restricted to accepting a proper subset of the regular languages and that the language $L = \{a, b\}^*b$ is not a member of that subset.

Ambainis and Freivalds [3] showed that MM-QFAs could accept languages with probability higher than $\frac{7}{9}$ if and only if the language could be accepted by an RFA, which is equivalent to being accepted with certainty by an MM-QFA. In [2] Ambainis, Bonner, Freivalds, and Kikusts construct a hierarchy of languages such that the ith language in the hierarchy can be accepted by an MM-QFA with at most probability p_i, where the series (p_i) converges to $\frac{1}{2}$ and is strictly decreasing.

Ambainis, Nayak, Ta-Shma, and Vazirani [5], and Nayak [13], investigated how efficiently MM-QFAs can be constructed compared to DFAs. They showed that for some languages the accepting MM-QFA is exponentially larger than the corresponding DFA.

In [1] Amano and Iwama studied a restricted version of the 2QFA model where the head was not allowed to move right. They showed that the emptiness problem for this model is not decidable. This is another instance where quantum mechanics provides computational power that is not achievable in the classical case.

3. MO-QFAs.

3.1. Bounded error acceptance. Restricting MO-QFAs to accept with bounded error greatly reduces their accepting power; this is not surprising since PFAs suffer a similar fate [18]. Since MM-QFAs can accept only a proper subset of the regular languages if they are required to accept with bounded error and since every MO-QFA can be simulated exactly by an MM-QFA, the class **RMO** is a proper subset of the regular languages. The class **RMO** is exactly the class of languages accepted by GFAs, otherwise known as group languages, and whose syntactic semigroups are groups; see Eilenberg [7]. This result is implied by Theorem 7 in [12] but is not stated in the paper. To prove this result we first need Lemma 3.1.

LEMMA 3.1. *Let U be a unitary matrix. For any $\epsilon > 0$ there exists an integer $n > 0$ such that for all vectors x, where $\|x\|^2 \leq 1$, it is true that $\|(I - U^n)x\|^2 < \epsilon$.*

Proof. Let $m = \dim(U)$. Since U is a normal matrix, U^n can be written as

$$U^n = PD^nP^{-1},$$

where P is a unitary matrix and D is the diagonal matrix of eigenvalues with the jth eigenvalue having the form $e^{i\pi r_j}$ [14]. If all eigenvalues in D are rotations through rational fractions of π, i.e., r_j is rational, then let $n = 2\prod_{j=1}^{m} q_j$, where q_j is the denominator of r_j. Thus $D^n = I$ and we are done.

Otherwise, at least one eigenvalue is a rotation of unity through an irrational fraction of π. Let $l \leq m$ be the number of these eigenvalues. For the other $m - l$ eigenvalues compute n, just as above, and let $D' = D^n$. The value of the jth element on the diagonal of D' is either 1 or $e^{i\pi n r_j}$, where r_j is some irrational real number. Consider taking D' to some power $k \in \mathbb{Z}^+$. The values that are 1 do not change, but the other l values that are of the form $e^{i\theta_j k}$, where $\theta_j = \pi n r_j$, form a vector that varies through a dense subset in an l-dimensional torus. Hence, there exists k such that the l-dimensional vector is arbitrarily close to $\vec{1} = (1, 1, \ldots, 1)$. Thus, for any $\epsilon' > 0$ there exists a $k > 0$ such that $\|(I - D'^k)\vec{1}\|^2 < \epsilon'$. Hence

$$\begin{aligned}
\|(I - U^{nk})x\|^2 &= \|(I - PD^{nk}P^{-1})x\|^2 \\
&= \|P(I - D'^k)P^{-1}x\|^2 \\
&\leq \|(I - D'^k)m\vec{1}\|^2 \\
&= m^2\|(I - D'^k)\vec{1}\|^2 \\
&\leq m^2\epsilon'.
\end{aligned}$$

Select ϵ' such that $\epsilon' < \frac{\epsilon}{m^2}$ to complete the proof. □

Lemma 3.2, due to Bernstein and Vazirani [6], states that if two configurations are close, then the differences in probability distributions of the configurations are small. This lemma relates the closeness of configurations to the variation distance between their probability distributions and allows us to partition the set of reachable configurations into equivalence classes. The variation distance between two probability distributions is the maximum difference in the probabilities of the same event occurring with respect to both distributions.

LEMMA 3.2 (Bernstein and Vazirani [6]). *Let $|\psi\rangle$ and $|\varphi\rangle$ be two complex vectors such that $\| |\psi\rangle\|^2 = \| |\varphi\rangle\|^2 = 1$ and $\| |\psi\rangle - |\varphi\rangle\|^2 < \epsilon$. The total variation distance between the probability distributions resulting from the measurement of $|\psi\rangle$ and $|\varphi\rangle$ is at most 4ϵ.*

Theorem 3.3 follows from these two lemmas.

THEOREM 3.3. *A language L can be accepted by an MO-QFA with bounded error if and only if it can be accepted by a GFA.*

Proof. The "if" direction follows from the fact that the transition function for a GFA is also a valid transition function for an MO-QFA that can accept the same language with certainty.

For the "only if" direction, by contradiction, assume that there exists a language L that can be accepted by an MO-QFA with bounded error but cannot be accepted by a GFA. Since the class **RMO** is a subset of the regular languages, L must be regular. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an MO-QFA that accepts L with bounded error. If two strings x and y take M into the same reachable configuration, then for any string z the probability of M accepting xz is equal to the probability of M accepting yz, which means that $xz \in L$ if and only if $yz \in L$. Therefore, the space of reachable configurations of M's computation can be partitioned into a finite number of equivalence classes defined by the corresponding minimal DFA for L.

Let $|\psi\rangle$ and $|\varphi\rangle$ denote reachable configurations of M and let $\sim_L$ be the right invariant equivalence relation induced by L. Since L cannot be accepted by a GFA, there must exist two distinct equivalence classes $[y]$ and $[y']$, an equivalence class $[x]$, and a symbol $\sigma \in \Sigma$ such that $[y\sigma] \sim_L [y'\sigma] \sim_L [x]$. If U_σ is the transition matrix for symbol σ, $|\psi\rangle \in [y]$ and $|\varphi\rangle \in [y']$, then $U_\sigma|\psi\rangle \in [x]$ and $U_\sigma|\varphi\rangle \in [x]$.

Since M accepts L with bounded error, let ϵ be the margin. By Lemma 3.1 there exists an integer $k > 0$ such that $\|(I - U_\sigma^k)|\psi\rangle\|^2 < \frac{\epsilon}{4}$ and $\|(I - U_\sigma^k)|\varphi\rangle\|^2 < \frac{\epsilon}{4}$. Hence, $U_\sigma^k|\psi\rangle \in [y]$ because if

$$\begin{aligned}\|(I - U_\sigma^k)|\psi\rangle\|^2 &= \| |\psi\rangle - U_\sigma^k|\psi\rangle\|^2 \\ &= \|V(|\psi\rangle - U_\sigma^k|\psi\rangle)\|^2 \\ &< \frac{\epsilon}{4},\end{aligned}$$

where V is an arbitrary unitary matrix, then by Lemma 3.2 the probability of $VU_\sigma^k|\psi\rangle$ being measured in a particular state is within ϵ of $V|\psi\rangle$ being measured in the same state; this probability is less than the margin. Similarly $U_\sigma^k|\varphi\rangle \in [y']$. Hence $[y] \sim_L [y\sigma^k]$ and $[y'] \sim_L [y'\sigma^k]$.

We assumed that $[x] \sim_L [y\sigma] \sim_L [y'\sigma]$ and showed that $[y] \sim_L [y\sigma^k]$ and $[y'] \sim_L [y'\sigma^k]$; therefore, $[y] \sim_L [x\sigma^{k-1}] \sim_L [y']$. Let z be the string that distinguishes $[y]$ and $[y']$. Then the string $\sigma^{k-1}z$ partitions $[x]$ into at least two distinct equivalence classes, but this is a contradiction. Therefore, there cannot exist a language L that can be accepted by an MO-QFA with bounded error but not by a GFA. □

Theorem 3.3 implies that $\mathbf{RMO}_\epsilon = \mathbf{RMO}_{\epsilon'}$ for all $\epsilon, \epsilon' > 0$; hence there are most two distinct classes of languages accepted by MO-QFAs: the restricted class **RMO**, which is equivalent to the class of languages accepted by a GFA, and the unrestricted class **UMO**.

It follows immediately from Theorem 3.3 that the class **RMO** is closed under Boolean operations, inverse homomorphisms, and word quotients and is not closed under homomorphisms.

3.2. Nonregular languages. Unlike the class **RMO**, the class **UMO** contains languages that are nonregular. This is not surprising given that Rabin [18] proved a similar result for PFAs. In fact our proof closely mimics Rabin's [18] technique.

LEMMA 3.4. *Let $L = \{x \in \{a,b\}^* \mid |x|_a \neq |x|_b\}$; there exists a 2-state MO-QFA M that accepts L with cut-point 0.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $F = \{q_1\}$, and δ is defined by the transition matrices

$$U_a = U_b^{-1} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix},$$

where α is an irrational fraction of π. Since U_a is a rotation matrix and α is an irrational fraction of π, the orbit formed by applying U_a to $|q_0\rangle$ is dense in the circle, and there exists only one k, such that $U_a^k|q_0\rangle = |q_0\rangle$, namely $k = 0$. This also holds for $U_b = U_a^{-1}$. Thus, $U(x)|q_0\rangle = |q_0\rangle$ if and only if the number of U_a rotations applied to $|q_0\rangle$ is equal to the number of U_b rotations, which is true if and only if the $|x|_a = |x|_b$. Otherwise, M has a nonzero probability of halting in state q_1. ∎

Lemma 3.4 implies that the class **RMO** is properly contained within the class **UMO**, and therefore the two classes are distinct.

The MO-QFA in Lemma 3.4 solves the word problem for the infinite cyclic group: is the input word equal to the identity element in the group, where the group has only one generator element, say a, and its inverse $b = a^{-1}$? We can generalize this result to the general word problem for the free group. The word problem for a free group is to decide whether or not a product of a sequence of elements of the free group reduces to the identity [10].

LEMMA 3.5. *The word problem for the free group language can be accepted by an MO-QFA.*

Proof. Construct a free group of rotation matrices drawn from the group SO_3 as discussed by Wagon [20]. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a 3-state MO-QFA where $\Sigma = \{a, a^{-1}, b, b^{-1}, \dots\}$ such that $|\Sigma|$ is equal to the sum of the number of rotation matrices and their inverses, δ is defined by the rotation matrices and their inverses, and $F = \{q_0\}$. The MO-QFA will accept identity words with certainty and reject non-identity words with a strictly nonzero probability, hence solving the word problem for the free group. ∎

3.3. Equivalence of MO-QFAs. In classical automata theory there is an algorithm to determine if two automata are equivalent. We say that QFAs M and M' are equivalent if their probability distributions over Σ^* are the same: for every word $x \in \Sigma$, the probability of M accepting x is equal to the probability of M' accepting x. In order to determine if two MO-QFAs are equivalent we first bilinearize them using the method detailed by Moore and Crutchfield [12]; this yields two generalized stochastic systems. We then apply Paz's [15, p. 21, p. 140] method for testing stochastic system equivalence to the generalized stochastic systems to determine if they have the same distribution.

3.4. Simulation of MO-QFAs by PFAs. Most classical computation is either deterministic or probabilistic; hence it is useful to ask how probabilistic automata compare to their quantum analogues. In the case of MO-QFAs, any language accepted by an MO-QFA can also be accepted by a PFA. If L can be accepted by an MO-QFA with bounded error, then it can also be accepted by a PFA with bounded error.

THEOREM 3.6. *Let M be an MO-QFA that accepts L with cut-point λ; then*

1. *there exists a PFA that accepts L with some cut-point λ',*
2. *if M accepts L with bounded error, then there exists a PFA that accepts L with bounded error.*

Proof. The second result follows from Theorem 3.3 because every GFA is also a PFA.

Since we can bilinearize M, L is a generalized cut-point event (GCE) [15, p. 153]. Since the class of GCEs is equal to the class of probabilistic cut-point events (PCEs) [15, p. 153], which are accepted by PFAs, there exists a PFA that can accept L with some cut-point λ'. □

Combining Theorem 3.6 with Lemma 3.5 yields a new insight into the languages accepted by PFAs.

COROLLARY 3.7. *The word problem for the free group language can be solved by a PFA.*

4. MM-QFAs. MM-QFAs are more powerful than MO-QFAs because a measurement is performed after every transition. This allows the machine to terminate before reading the entire string and simulate the spin states of RFAs.

As mentioned before, an MM-QFA uses one end-marker while the Kondacs and Watrous [9] 1QFA uses two end-markers. The second marker does not add any more power to the model (see Appendix A) but makes constructing an MM-QFA easier because the MM-QFA can start in an arbitrary configuration. Hence, for the sake of conciseness and clarity we shall assume that some of the MM-QFAs constructed in the following proofs have two end-markers.

4.1. Closure properties. Unlike the closure properties of the classes **RMO** and **UMO**, which can be derived easily, the closure properties of the classes **RMM** and **UMM** are not as evident and in one important case are unknown. We show that the classes **RMM** and **UMM** are closed under complement, inverse homomorphism, and word quotient. Similar to the class **RMO**, the class **RMM** is not closed under homomorphisms. It remains an open problem to determine whether the classes **RMM** and **UMM** are closed under Boolean operations.

Theorem 4.1 proves that both classes are closed under complement and inverse homomorphisms by showing that each class $\mathbf{RMM}_\epsilon$ is closed under complement and inverse homomorphisms; closure under word quotient follows directly from the latter, given the presence of end-markers.

THEOREM 4.1. *The class $\mathbf{RMM}_\epsilon$ is closed under complement, inverse homomorphisms, and word quotient.*

Proof. Closure under complement follows from the fact that we can exchange the accept and reject states of the MM-QFA. This exchanges the probabilities of acceptance and rejection but does not affect the margin.

Given an MM-QFA M and a homomorphism h we construct an MM-QFA M' that accepts $h^{-1}(L)$. Let $M = (Q, \Sigma, \delta, Q_{acc}, Q_{rej})$ and $M' = (Q', \Sigma, \delta', Q'_{acc}, Q'_{rej})$. Assume that δ and δ' are defined in terms of matrices $\{U_\sigma\}_{\sigma\in\Sigma}$ and $\{U'_\sigma\}_{\sigma\in\Sigma}$. Unlike the proof for MO-QFAs in [12], the direct construction of

$$U'_\sigma = U(h(\sigma))$$

will not work because a measurement occurs between transitions, and combining transitions without taking this into account could produce incorrect configurations. After every transition some amount of probability amplitude is placed in the halting states and should not be allowed to interact with the nonhalting states in the following transitions. This is achieved by storing the amplitude in additional states; this technique is also used in [5]. Assume without loss of generality that

$$Q_{non} = \{q_i \in Q \mid 0 \le i < n_{non}\},$$
$$Q_{halt} = \{q_i \in Q \mid n_{non} \le i < n\},$$

where $n = |Q|$ and $n_{non} = |Q_{non}|$. Let $m = \max_{\sigma\in\Sigma}\{|h(\sigma)|\}$ and let

$$Q' = Q \cup Q'_{halt},$$

where

$$Q'_{halt} = \{q_i\}_{i=n+1}^{n+m(n-n_{non})},$$
$$Q'_{acc} = Q_{acc} \cup \{q_{n+j(i-n_{non})} \in Q'_{halt} \mid q_i \in Q_{acc}, 1 \le j \le m\},$$
$$Q'_{rej} = Q_{rej} \cup \{q_{n+j(i-n_{non})} \in Q'_{halt} \mid q_i \in Q_{rej}, 1 \le j \le m\}.$$

Intuitively, we replicate the halting states m times; each replication is termed a halting state set.

We construct δ' from the matrices of δ. Let V_σ be a unitary block matrix

$$V_\sigma = U_{shift} \begin{bmatrix} U_\sigma & \\ & I_{m(n-n_{non})} \end{bmatrix},$$

where

$$U_{shift} = \begin{bmatrix} I_{n_{non}} & & \\ & & I_{n-n_{non}} \\ & I_{m(n-n_{non})} & \end{bmatrix}.$$

The matrix U_{shift} is a unitary matrix that shifts the amplitudes in the halting set i to the halting set $i+1$ and the amplitude in halting set m to halting set 0. Analogous to the MO-QFA case where $U'_\sigma = U(h(\sigma))$, for MM-QFAs let

$$U'_\sigma = V(h(\sigma)) = V_{x_k} V_{x_{k-1}} \dots V_{x_1},$$

where $h(\sigma) = x = x_1 x_2 \dots x_k$ and $k \le m$.

After every x_i subtransition the halting amplitude is shifted and stored in the $m+1$ halting sets of states. When the subtransition is done, the amplitude in halt state set 0 is zero, which is what is required to prevent unwanted interactions. A minimum of m subtransitions must occur before halting set m contains nonzero amplitude, but no more than m subtransitions will ever occur; therefore halting set 0 will never receive nonzero amplitude from halting set m. Since M' has the same distribution as M, the margin will not decrease.

Closure under word quotient follows from closure under inverse homomorphism and the presence of both end-markers. □

Just like the class **RMO**, the class **RMM** is not closed under homomorphisms.

THEOREM 4.2. *The class* **RMM** *is not closed under homomorphisms.*

Proof. Let $L = \{a, b\}^* c$ and define a homomorphism h to be $h(a) = a$, $h(b) = b$, and $h(c) = b$. Since L can be accepted by an RFA, $L \in \mathbf{RMM}$ [3], but $h(L) = \{a, b\}^* b \notin \mathbf{RMM}$, the result follows. □

A more interesting question is whether the classes **RMM** and **UMM** are closed under Boolean operations. Unlike MO-QFAs that have two types of states, accept and reject, MM-QFAs have three types of states: accept, reject, and nonhalt. Consequently, the standard procedure of taking the tensor product of two automata to obtain their intersection or union does not work. A general method of intersecting two MM-QFAs is not known. Thus, it is not known whether **RMM** and **UMM** are closed under Boolean operations.

4.2. Bounded error acceptance. The restriction of bounded error acceptance reduces the class of languages that an MM-QFA can accept to a proper subclass of the regular languages [9]. To study the languages in class **RMM**, we look at their corresponding minimal automata. Ambainis and Freivalds [3] showed that if the minimal DFA $M(L) = (Q, \Sigma, \delta, q_0, F)$ contains an irreversible construction, defined by two distinct states $q_1, q_2 \in Q$ and strings $x, y, z \in \Sigma^*$ such that $\delta(q_1, x) = \delta(q_2, x) = q_2$, $\delta(q_2, y) \in F$, and $\delta(q_2, z) \notin F$, then an RFA cannot accept L and an MM-QFA cannot accept it with a probability greater than $\frac{7}{9}$; this condition is both sufficient and necessary.

We derive a similar necessary condition for a language L to be a member of the class **RMM**. This condition, called the partial order condition, is a relaxed version of a condition defined by Meyer and Thompson [11]. A language L is said to satisfy the partial order condition if the minimal DFA for L satisfies the partial order condition. A DFA satisfies the partial order condition if it does not contain two distinguishable states $q_1, q_2 \in Q$ such that there exists strings $x, y \in \Sigma^+$, where $\delta(q_1, x) = \delta(q_2, x) = q_2$, and $\delta(q_2, y) = q_1$. States q_1 and q_2 are said to be distinguishable if there exists a string $z \in \Sigma^*$ such that $\delta(q_1, z) \in F$ and $\delta(q_2, z) \notin F$ or vice versa [8]. Using a result in [9], Theorem 4.3 proves that the partial order condition is necessary for an MM-QFA to accept L with bounded error.

THEOREM 4.3. *If $M = (Q, \Sigma, \delta, q_0, F)$ is a minimal DFA for language L that does not satisfy the partial order condition, then $L \notin \mathbf{RMM}$.*

Proof. By contradiction, assume that $L \in \mathbf{RMM}$. Let $L_b = \{a, b\}^* b$. Since the minimal DFA for L does not satisfy the partial order condition, there exist states $q_1, q_2 \in Q$ and strings $x, y \in \Sigma^+$ as defined above and a distinguishing string $z \in \Sigma^*$ such that $\delta(q_1, z) \notin F$ if and only if $\delta(q_2, z) \in F$. Without loss of generality assume that $\delta(q_1, z) \notin F$ and $\delta(q_2, z) \in F$.

Let s be the shortest string such that $\delta(q_0, s) = q_1$. Let $L' = s^{-1} L z^{-1}$. By Theorem 4.1, $L' \in \mathbf{RMM}$. Define the homomorphism h as

$$\begin{aligned} h(a) &= xy, \\ h(b) &= x, \\ h(\Sigma - \{a, b\}) &= xy, \end{aligned}$$

where the last definition is for completeness. Let $L'' = h^{-1}(L')$. By Theorem 4.1 $L'' \in \mathbf{RMM}$. But $L'' = L_b \notin \mathbf{RMM}$, a contradiction. □

The partial order condition is so named because once the state q_2 is visited, there is no path back to state q_1. Thus, there exists a partial order on the states of the DFA. We do not know whether this condition is also sufficient for MM-QFA acceptance with bounded error. While we do not know whether the class **RMM** is

closed under Boolean operations, Theorem 4.6 relates closure under intersection to the partial order condition.

LEMMA 4.4. *Let M be a DFA that satisfies the partial order condition. The minimal DFA M' that accepts $L(M)$ satisfies the partial order condition.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and $M' = (Q', \Sigma, \delta', q_0', F')$ be the corresponding minimal DFA. Assume by contradiction that M' does not satisfy the partial order condition. Hence, M' has two states that correspond to the equivalence classes $[q_1']$ and $[q_2']$ such that $[q_1']x \sim_L [q_2']x \sim_L [q_2']$ and $[q_2']y \sim_L [q_1']$. By the Myhill–Nerode theorem [8], the equivalence classes partition the set of reachable states in Q. Hence, for each equivalence class $[q_i']$ there is a corresponding subset of Q. Let Q_1 and Q_2 denote the subsets of Q corresponding to the equivalence classes $[q_1']$ and $[q_2']$ and assign an arbitrary order on each subset. Select the first state, say $p_1 \in Q_1$, and define the set $R = \{q \in Q_2 \mid \exists n, m \in \mathbb{Z}^+,\ \delta(p_1, x^m) = \delta(q, x^n) = q\}$. If there exists a state $r \in R$ and string $y \in \Sigma^+$ such that $\delta(r, y) = p_1$, then M does not satisfy the partial order condition, and this is a contradiction. Otherwise, there does not exist a $y \in \Sigma^+$ such that $\delta(r, y) = p_1$ for all $r \in R$. In this case there is a partial order on p_1 and on $Q_1 \backslash \{p_1\}$ because p_1 will never be visited again if M reads a sufficient number of x's. Remove p_1 from Q_1 and repeat the procedure on $p_2 \in Q_1$. After a finite number of iterations either we will find a p_i that satisfies our requirements, which means that M does not satisfy the partial order condition and is a contradiction, or none of the states in Q_1 will have the required characteristics, in which case M' satisfies the partial order condition. Therefore, if M satisfies the partial order condition, so will its minimal equivalent M'. □

LEMMA 4.5. *Let L' and L'' be languages that satisfy the partial order condition. Then $L = L' \cap L''$ also satisfies the partial order condition.*

Proof. Let $M' = (Q', \Sigma, \delta', q_0', F')$ be the minimal DFA accepting the language L' and let $M'' = (Q'', \Sigma, \delta'', q_0'', F'')$ be the minimal DFA accepting the language L''. We first construct an automaton M that accepts $L' \cap L''$ by combining M' and M'' using a direct product. Define $M = (Q, \Sigma, \delta, q_{00}, F)$, where $Q = Q' \times Q''$, $q_{00} = (q_0', q_0'')$, $F = \{(q', q'') \in Q \mid q' \in F' \ \wedge \ q'' \in F''\}$, and $\delta((q', q''), \sigma) = (\delta'(q', \sigma), \delta''(q'', \sigma))$.

We argue that if M' and M'' satisfy the partial order condition, then so will M. Assume, by contradiction, that M does not satisfy the partial order condition. Then there exist two states $q_{ij} = (q_i', q_j'')$ and $q_{kl} = (q_k', q_l'')$ and strings $x, y, z \in \Sigma^+$ such that $\delta(q_{ij}, x) = \delta(q_{kl}, x) = q_{kl}$, $\delta(q_{kl}, y) = q_{ij}$, and $\delta(q_{ij}, z) \in F$ if and only if $\delta(q_{kl}, z) \notin F$. In the first case assume that either $i \neq k$ or $j \neq l$, and without loss of generality, assume the former. Then there exists state $q_i' \in Q'$ and state $q_k' \in Q'$ such that $\delta'(q_i', x) = \delta'(q_k', x) = q_k'$, $\delta_1(q_k', y) = q_i'$. But this means that M' does not satisfy the partial order condition, a contradiction. In the second case assume that $i = k$ and $j = l$. This implies that $q_{ij} = q_{kl}$ and hence there cannot exist a string z that distinguishes the two states, also a contradiction. Therefore M must satisfy the condition.

Since M satisfies the partial order condition and accepts L, by Lemma 4.4 the minimal automaton that accepts L satisfies the partial order condition, and hence L itself satisfies the partial order condition. □

THEOREM 4.6. *If the partial order condition is sufficient for acceptance with bounded error by MM-QFAs, then the class* **RMM** *is closed under intersection.*

Proof. By Lemma 4.5 the intersection of two languages that satisfy the partial order condition is a language that satisfies the partial order condition. □

One method for proving that the class **RMM** is not closed under intersection

involves intersecting two languages in **RMM** and showing that the resulting language is not in **RMM**. By Theorem 4.6 this method will not work unless the partial order condition is insufficient. To study whether the partial order condition is sufficient, as well as necessary, we show that a well-known class of languages can be accepted by an MM-QFA with bounded error.

4.3. Piecewise testable sets. A piecewise testable set is a Boolean combination of sets of the form

$$L_z = \Sigma^* z_1 \Sigma^* z_2 \Sigma^* \dots \Sigma^* z_n \Sigma^*,$$

where $z_i \in \Sigma$ [16]. Intuitively, L_z is the language of strings that contains the successive symbols of z as a subsequence; we call such a language a partial piecewise testable set.

Piecewise testable sets, introduced by Simon in [19], form a natural family of star-free languages. Such sets define a class of computations that wait for a partially ordered sequence of trigger events (input symbols); if a trigger event (symbol) is read that is not next in the sequence, it is simply ignored. Another natural interpretation of piecewise testable sets is subsequence searching. Consider a language where a word is said to be in the language if it contains a finite Boolean combination of subsequences. Such a language is a piecewise testable set and word acceptance corresponds to searching the words for the required subsequences. Finally, such languages belong to a class of languages whose MM-QFAs have an arbitrarily large, but finite, set of ordered states.

We show, by construction, that MM-QFAs can accept partial piecewise testable sets with bounded error. The MM-QFAs we construct accept with one-sided error and are what we call "end-decisive." We say that an MM-QFA accepts with positive one-sided error if it accepts strings in the language with nonzero probability and rejects strings not in the language with certainty. We say that an MM-QFA accepts with negative one-sided error if it accepts strings in the language with certainty and rejects strings not in the language with nonzero probability.

We say that an MM-QFA is end-decisive if it will not be observed in an accept state until the end-marker \$ is read. An MM-QFA is co-end-decisive if it will not be observed in a reject state until the end-marker is read.

Classes of languages that are accepted by end-decisive MM-QFAs with the same one-sided error, i.e., all positive or all negative, are closed under intersection and union. Furthermore, if language L can be accepted by an end-decisive MM-QFA with bounded error, and language L' can be accepted by an end-decisive MM-QFA with bounded one-sided error, then the union or intersection of L and L' can be accepted by an end-decisive MM-QFA with bounded error. To construct these MM-QFAs we introduce two useful concepts: junk states and trigger chains.

A junk state is a halting state of an end-decisive or co-end-decisive MM-QFA. If the MM-QFA is end-decisive, then all its junk states are reject states. If the MM-QFA is co-end-decisive, then all its junk states are accept states. An end-decisive or co-end-decisive MM-QFA may be observed in a junk state at any point of the computation. While junk states are either accept or reject states, we treat the junk state as a separate halting state. Any accept or reject state that is not a junk state is called a decisive state. Intuitively, a junk state signals a failed computation.

Each end-decisive or co-end-decisive automata that accepts with bounded error has probability, bounded by some constant $\tau < 1$, of ending up in a junk state and a probability $1 - \tau$ of ending up in a decisive state. If $\tau \not< 1$, then the amount

of probability amplitude ending up in a decisive state can become arbitrarily small, dropping below any fixed margin. Thus, τ must be strictly less than one for the MM-QFA to accept with bounded error; τ is independent of the input string x.

A trigger chain is a construction of junk states and transition matrices that causes a reduction in amplitude of a particular state only if the amplitude of another state is decreased, presumably by some previous transition. Trigger chains correspond directly to partial piecewise testable sets. Consider the matrix

$$X = \begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{2}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{2} & -\frac{1}{\sqrt{2}} & \frac{1}{2} \end{bmatrix}.$$

This matrix is a special case of a transition matrix introduced by Ambainis and Freivalds [3]. This matrix operates on three states and is a triggering mechanism of the chain. Consider the vectors

$$|\psi\rangle = (\alpha, 0, \beta)^T$$

and

$$X|\psi\rangle = \left(\frac{\alpha}{2} + \frac{\beta}{2}, \frac{\alpha}{\sqrt{2}} - \frac{\beta}{\sqrt{2}}, \frac{\alpha}{2} + \frac{\beta}{2}\right)^T.$$

The vectors $|\psi\rangle$ and $X|\psi\rangle$ are equal if and only if $\alpha = \beta$. If $\alpha \neq \beta$, then the amplitudes of the first and third state are averaged, with the remainder of the amplitude going into the second state. We define a generalized version of X by embedding it into a larger identity block matrix. Define X_i to be

$$X_i = \begin{bmatrix} I_i & & \\ & X & \\ & & I_{s-i-3} \end{bmatrix},$$

where I_m is an $m \times m$ identity matrix, X is defined as above, and s is the number of states, i.e., the size of X_i. The matrix X_i operates on a triple of states, q_i through to q_{i+2}. We assume that state q_{i+1}, the second state, is a junk state unless otherwise noted.

THEOREM 4.7. *Let L_z be a partial piecewise testable set. There exists an end-decisive MM-QFA that accepts L_z with bounded positive one-sided error.*

Proof. We construct an MM-QFA M with $m+1$ states that accepts L_z where $z = z_0 z_1 \dots z_n$ and $m = 2n + 4$.

For each link in the trigger chain we require a junk state and a nonhalting state. We order the states to correspond with the description of the X_i matrices. Specifically, the first $2n+1$ states are the nonhalting states, interleaved with junk states. Each triple of states $(q_{2i}, q_{2i+1}, q_{2i+2})$ corresponds to a link of the trigger chain, of which there are $n+1$. State q_{2n+1} is the decisive accept state and state q_{2n+3} is the decisive reject state. The junk states are rejecting states.

Let $m = 2n + 4$ and $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$, where

$$\begin{aligned} Q &= \{q_0, \dots, q_m\}, \\ Q_{junk} &= \{q_i \in Q \mid 0 < i < 2n \ \wedge \ i \equiv 1 \bmod 2\} \cup \{q_{2n+4}\}, \\ Q_{acc} &= \{q_{2n+1}\}, \\ Q_{rej} &= \{q_{2n+3}\}. \end{aligned}$$

Define δ by the transition matrices $\{U_\sigma\}_{\sigma\in\Sigma}$. Each transition matrix U_σ consists of a product of matrices:

$$U_\sigma = U_{\sigma,0}U_{\sigma,1}\dots U_{\sigma,n},$$

where the matrices $U_{\sigma,i}$ implement the triggers.

Define $U_{\sigma,i}$ to be

$$U_{\sigma,i} = \begin{cases} S, & i = 0 \ \wedge \ z_0 = \sigma, \\ X_{2i-2}, & 1 \le i \le n \ \wedge \ z_i = \sigma, \\ I_{m+1}, & \text{otherwise}, \end{cases}$$

where

$$S = \begin{bmatrix} 0 & 1 & & \\ 1 & 0 & & \\ & & I_{m-1} & \end{bmatrix}.$$

The matrix S shifts the amplitude of q_0 to the junk state q_1. This is the first trigger that is activated when z_0 is read.

Finally, let the transition matrix for the end-marker \$ be

$$U_\$ = FX_{2n},$$

where

$$F = \begin{bmatrix} R & & & & & & & \\ & \ddots & & & & & & \\ & & R & & & & & \\ & & & 0 & 0 & 0 & 0 & 1 \\ & & & 0 & 1 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 & 1 & 0 \\ & & & 0 & 0 & 1 & 0 & 0 \\ & & & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the matrix

$$R = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

The matrix F sends all amplitude into the junk states. The matrix X_{2n} sends some minimum amount of amplitude into an accept state if the amplitudes of states q_{2n} and q_{2n+2} differ.

The initial configuration of the machine is $|\psi_{init}\rangle = (\alpha_0, \alpha_1, \dots, \alpha_m)^T$, where

$$\alpha_i = \begin{cases} \frac{1}{\sqrt{n+2}}, & 0 \le i \le 2n+2 \ \wedge \ i \equiv 0 \pmod 2, \\ 0, & \text{otherwise}, \end{cases}$$

i.e., the amplitude is evenly distributed among all nonhalting states.

The only decisive accepting state in the machine is q_{2n+1}, and amplitude flows into it only when the end-marker is read. In order for it to get a nonzero amplitude, the amplitudes of states q_{2n} and q_{2n+2} must differ. Since all nonhalting states start

with the same amplitude, and since the amplitude of state q_{2n+2} will not change during the execution of the machine until the end-marker is read, the amplitude of state q_{2n-2} must change in order for the amplitude of state q_{2n} to change. Following the same argument, state q_{2i} will not change in amplitude until state q_{2i-2} changes in amplitude. Furthermore, the change in amplitude of state q_{2i} is governed by the matrix components X_{2i-2} and X_{2i}. Hence, the initial change of amplitude of state q_{2i} depends exclusively on a change in amplitude of state q_{2i-2} and is governed by component X_{2i-2} that is located in the transition matrix U_{z_i}. If any other transition matrix is applied, then the amplitude of state q_{2i} will not change. Hence, M can read $(\Sigma - \{z_i\})^*$ without changing the amplitude of state q_{2i}, but, as soon as z_i is read, component X_{2i-2} will be applied and q_{2i} will have a decreased amplitude, provided state q_{2i-2} already had a decrease of its amplitude. Finally, the amplitude of any state q_{2i} will never increase beyond its initial value, and once the amplitude of state q_{2i} decreases, it will never increase beyond $\frac{1}{\sqrt{n+2}}(1 - (\frac{1}{2})^{n+1})$. For the case of symbol z_0, the amplitude of state q_0 is changed by matrix S to 0 and is the starting trigger. When the end-marker is read, a minimum of $\frac{1}{\sqrt{2(n+2)}}(\frac{1}{2})^{n+1}$ of amplitude is placed into the accepting state only if the amplitude of state q_{2n} has decreased. The amplitude from q_{2n+2} is channeled into a decisive reject state. The rest of the amplitude from the remaining $n+1$ nonhalting states is channeled into junk states. If the amplitudes of q_{2n} and q_{2n+2} do not differ, then all amplitude is channeled into junk and decisive reject states.

The probability of M accepting a string not in the language is 0, while the probability of M accepting a string in the language is at least $\frac{1}{n+2}(\frac{1}{2})^{2n+3}$. We select the cut-point to be strictly between the two values. □

Any Boolean combination of partial piecewise testable sets may be expressed as a union of intersections of partial piecewise testable sets and complements of partial piecewise testable sets, i.e.,

$$\bigcup_{i=0}^{s}\left(\bigcap_{j=0}^{t}\tilde{L}_{ij}\right), \tag{4.1}$$

where $\tilde{L}_{ij}$ is a partial piecewise testable set or the complement thereof.

We first show how to construct the implicants of the above expression, i.e., $\cap_{j=0}^{t}\tilde{L}_{ij}$, and then how to take the union of the implicants. An implicant can be written in the form

$$\bigcap_{j=0}^{t}\tilde{L}_{ij} = \left(\bigcap_{j=0}^{r}L_{ij}\right)\bigcap\left(\bigcap_{j=r}^{t}\overline{L}_{ij}\right),$$

where the L_{ij}'s are partial piecewise testable sets. By De Morgan's rule, the latter part of this expression can be rewritten as $\overline{\cup_{j=r}^{t}L_{ij}}$. Let $L^{\cap}{}_i = \cap_{j=0}^{r}L_{ij}$, let $L^{\cup}{}_i = \cup_{j=r}^{t}L_{ij}$, and let $L_i = L^{\cap}{}_i \cap \overline{L^{\cup}{}_i}$.

First, we show that $L^{\cap}{}_i$ can be accepted by an end-decisive MM-QFA with bounded positive one-sided error. Second, we show that $\overline{L^{\cup}{}_i}$ can be accepted by an end-decisive MM-QFA with bounded error. Third, we show that L_i can be accepted by an end-decisive MM-QFA with bounded error. Finally, we show that $\cup_{i=0}^{s}L_i$ can be accepted by an end-decisive MM-QFA with bounded error. We first need two composition lemmas.

We say that an MM-QFA M accepts L with cut-point λ and maximum margin η if for all $x \in \Sigma^*$,

$$\lambda - \eta < \Pr[M(x) = \text{accept}] < \lambda + \eta.$$

Usually, the maximum margin will be exponentially greater than the margin ϵ; this creates problems when we compose automata.

LEMMA 4.8. *Let M and M' be end-decisive MM-QFAs that accept L and L', respectively, with cut-points λ and λ', margins ϵ and ϵ', and maximum margins η and η'. There exists an end-decisive MM-QFA M'' such that the inequalities*

$$(4.2)\quad (\lambda+\epsilon)\cdot(\lambda'+\epsilon') \le \Pr[M''(x)=\text{accept}] \le (\lambda+\eta)\cdot(\lambda'+\eta') \quad \forall x \in L \cap L',$$
$$(4.3)\quad (\lambda-\eta)\cdot(\lambda'+\epsilon') \le \Pr[M''(x)=\text{accept}] \le (\lambda-\epsilon)\cdot(\lambda'+\eta') \quad \forall x \in \overline{L} \cap L',$$
$$(4.4)\quad (\lambda+\epsilon)\cdot(\lambda'-\eta') \le \Pr[M''(x)=\text{accept}] \le (\lambda+\eta)\cdot(\lambda'-\epsilon') \quad \forall x \in L \cap \overline{L'},$$
$$(4.5)\quad (\lambda-\eta)\cdot(\lambda'-\eta') \le \Pr[M''(x)=\text{accept}] \le (\lambda-\epsilon)\cdot(\lambda'-\epsilon') \quad \forall x \in \overline{L} \cap \overline{L'}$$

are satisfied.

Proof. Let $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ and $M' = (Q', \Sigma, \delta', q_0', Q'_{acc}, Q'_{rej})$ be end-decisive MM-QFAs that accept L and L'. Using these two MM-QFAs we construct an MM-QFA $M'' = (Q'', \Sigma, \delta'', q_0'', Q''_{acc}, Q''_{rej})$ that satisfies the above inequalities.

Let $Q'' = Q \times Q'$ and $q_0'' = (q_0, q_0')$. The sets of halting states are defined as

$$Q''_{acc} = \{(q_i, q_j') \in Q'' \mid q_i \in Q_{acc} \ \wedge \ q_j' \in Q'_{acc}\},$$
$$Q''_{rej} = \{(q_i, q_j') \in Q'' \mid (q_i \in Q_{rej} \ \vee \ q_j' \in Q'_{rej})\},$$

and the transition function δ'' is defined as

$$\delta''((q,q'),\sigma,(r,r')) = \delta(q,\sigma,r)\cdot\delta'(q',\sigma,r'),$$

which is a tensor product of the transition functions δ and δ'.

Since M and M' are end-decisive, i.e., the accepting states will have only nonzero amplitude when the end-marker is read, thus the MM-QFA M'' will be end-decisive.

By the tensor product construction, the probability of M'' accepting x is

$$\Pr[M''(x) = \text{accept}] = \Pr[M(x) = \text{accept}] \cdot \Pr[M'(x) = \text{accept}].$$

Since

$$\begin{aligned}
\lambda + \epsilon \le \Pr[M(x) = \text{accept}] \le \lambda + \eta \quad & \forall x \in L,\\
\lambda - \eta \le \Pr[M(x) = \text{accept}] \le \lambda - \epsilon \quad & \forall x \notin L,\\
\lambda' + \epsilon' \le \Pr[M'(x) = \text{accept}] \le \lambda' + \eta' \quad & \forall x \in L',\\
\lambda' - \eta' \le \Pr[M'(x) = \text{accept}] \le \lambda' - \epsilon' \quad & \forall x \notin L',
\end{aligned}$$

multiplying out the probabilities yields the inequalities (4.2), (4.3), (4.4), and (4.5). ∎

COROLLARY 4.9. *Let M and M' be end-decisive MM-QFAs that accept L and L', respectively, with bounded positive one-sided error. There exists an end-decisive MM-QFA that accepts $L \cap L'$ with bounded positive one-sided error.*

Proof. Let λ, λ', ϵ, and ϵ' be the respective cut-points and margins of MM-QFAs M and M'. Since $\lambda - \epsilon = \lambda' - \epsilon' = 0$, $\lambda + \epsilon > 0$, and $\lambda' + \epsilon' > 0$, the result follows from Lemma 4.8. ∎

We mentioned before that the maximum margin of an MM-QFA that accepts language L could be exponentially greater than the margin. This prevents us from directly constructing intersections or unions of languages that are accepted by end-decisive MM-QFAs with bounded error. To get around this problem we use a tensor power technique to magnify the ratio of the probability of a true positive to the probability of a false positive.

LEMMA 4.10. *Let M be an end-decisive MM-QFA that accepts words in L with probability at least $\lambda + \epsilon$ and accepts words not in L with probability at most $\lambda - \epsilon$. For any positive integer n there exists an MM-QFA M' that accepts words in L with probability at least $(\lambda + \epsilon)^n$ and accepts words not in L with probability at most $(\lambda - \epsilon)^n$.*

Proof. Using Lemma 4.8 to compose n copies of M yields the result. □

We first use Lemma 4.10 to construct finite unions of languages that are accepted by end-decisive MM-QFAs with bounded error.

LEMMA 4.11. *Let M be an end-decisive MM-QFA that accepts L with bounded error and let M' be an end-decisive MM-QFA that accept L' with bounded error. There exists an end-decisive MM-QFA M'' that accepts $L'' = L \cup L'$ with bounded error.*

Proof. Assume that M accepts words in L with probability at least $\lambda + \epsilon$ and accepts words not in L with probability at most $\lambda - \epsilon$. Similarly, assume that M' accepts words in L' with probability at least $\lambda' + \epsilon'$ and accepts words not in L' with probability at most $\lambda' - \epsilon'$.

Using Lemma 4.10, let M_s be the sth tensor power of M and let M'_t be the tth tensor power of M'.

Let $M_s = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ and $M'_t = (Q', \Sigma, \delta', q'_0, Q'_{acc}, Q'_{rej})$, where $Q = \{q_0, \ldots, q_{n-1}\}$ and $Q' = \{q'_0, \ldots, q'_{m-1}\}$. Let δ and δ' be represented by the unitary matrices U_σ and U'_σ, respectively.

Let $M'' = (Q'', \Sigma, \delta'', Q''_{acc}, Q''_{rej})$ where $Q'' = \{q''_0, \ldots, q''_{n+m-1}\}$; δ'' is represented by the matrices

$$U''_\sigma = \begin{bmatrix} U_\sigma & 0 \\ 0 & U'_\sigma \end{bmatrix},$$

$Q''_{acc} = \{q''_i \in Q'' \mid q_i \in Q_{acc} \vee q'_{i-n} \in Q'_{acc}\}$, and $Q''_{rej} = \{q''_i \in Q'' \mid q_i \in Q_{rej} \vee q'_{i-n} \in Q'_{rej}\}$. The automata are initialized with the amplitude evenly divided between the states q''_0 and q''_n, i.e., each state contains $\frac{1}{\sqrt{2}}$ amplitude. Intuitively, M and M' run in parallel, not interacting unless one of the two crashes. In that case the computation is over.

If $x \in L \cap L'$, then

$$\Pr[M''(x) = \text{accept}] \geq \frac{(\lambda + \epsilon)^s + (\lambda' + \epsilon')^t}{2};$$

if $x \in L \cap \overline{L'}$, then

$$\Pr[M''(x) = \text{accept}] \geq \frac{(\lambda + \epsilon)^s}{2};$$

if $x \in \overline{L} \cap L'$, then

$$\Pr[M''(x) = \text{accept}] \geq \frac{(\lambda' + \epsilon')^t}{2};$$

and if $x \in \overline{L} \cap \overline{L'}$, then

$$\Pr[M''(x) = \text{accept}] \leq \frac{(\lambda - \epsilon)^s + (\lambda' - \epsilon')^t}{2}.$$

The last case corresponds to $x \notin L''$. By setting s and t appropriately, we can ensure that

$$(\lambda - \epsilon)^s + (\lambda' - \epsilon')^t \ll \min\{(\lambda + \epsilon)^s, (\lambda' + \epsilon')^t\}.$$

Hence, the MM-QFA M'' accepts $L \cup L'$ with bounded error. Furthermore, M'' is end-decisive because both M_s and M_t are end-decisive. □

COROLLARY 4.12. *Let M and M' be end-decisive MM-QFAs that accept L and L', respectively, with bounded positive one-sided error. There exists an end-decisive MM-QFA that accepts $L \cup L'$ with bounded positive one-sided error.*

Proof. Since $\lambda - \epsilon = \lambda' - \epsilon' = 0$, the same argument as in Corollary 4.9 applies. □

One useful property of languages that are accepted by end-decisive MM-QFAs with bounded positive one-sided error is that we can usually construct end-decisive MM-QFAs that can accept the complement of such languages with bounded error. We say that an end-decisive MM-QFA accepts with positive amplitude if the amplitude in its accept states is always nonnegative.

LEMMA 4.13. *Let L be a language that is accepted by an end-decisive MM-QFA with bounded positive one-sided error and positive amplitude. There exists an end-decisive MM-QFA that accepts $\overline{L}$ with bounded error.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ be an end-decisive MM-QFA that accepts L with bounded positive one-sided error. Since M rejects all strings not in L with certainty, for every computation of M on $x \notin L$ zero amplitude is placed into the accepting states of M. Let $n = |Q|$, let $a = |Q_{acc}|$, and assume that $Q_{acc} = \{q_{n-1}, q_{n-2}, \ldots, q_{n-a}\}$.

We use M to construct an end-decisive MM-QFA M' to accept $\overline{L}$ with bounded error. Let $M' = (Q', \Sigma, \delta', q_0, Q'_{acc}, Q'_{rej})$, where

$$\begin{aligned} Q' &= Q \cup \{q_n, q_{n+1}, \ldots, q_{n+3a}\}, \\ Q'_{rej} &= Q_{rej} \cup Q_{acc} \cup \{q_{n+i} \in Q' \mid i \equiv 2 \bmod 3\}, \\ Q'_{acc} &= \{q_{n+i} \in Q' \mid i \equiv 0 \bmod 3\}, \end{aligned}$$

and the transition function δ' is extended in the following manner. For all symbols except the end-marker, the transition function for M' is defined by the matrices

$$U'_\sigma = \begin{bmatrix} U_\sigma & \\ & I_{3a} \end{bmatrix}.$$

The end-marker transition is defined by the matrix

$$U'_\$ = \begin{bmatrix} U_\$ & \\ & I_{3a} \end{bmatrix} X,$$

where matrix X performs an averaging and cleanup operation. We define X in terms of 4×4 submatrices. Every accept state $q_{n-a+i} \in Q_{acc}$ in M becomes a reject state in M'. Additionally, for each such state, three additional states were added to M', q_{n+3i}, q_{n+3i+1}, and q_{n+3i+2}, which are an accepting, a nonhalting,

and a rejecting state, respectively. The matrix X operates on the 4-tuples of states $(q_{n-a+i}, q_{n+3i}, q_{n+3i+1}, q_{n+3i+2})$. Each operation is localized to the 4-tuple of states and hence can be described by a 4×4 matrix X_i. Assume that the order of rows and columns of the matrix correspond to the order in the 4-tuple. Let

$$X_i = \overbrace{\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix}}^{\text{cleanup}} \overbrace{\begin{bmatrix} \frac{1}{2} & \frac{1}{\sqrt{2}} & \frac{1}{2} & \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & \\ \frac{1}{2} & -\frac{1}{\sqrt{2}} & \frac{1}{2} & \\ & & & 1 \end{bmatrix}}^{\text{averaging}}.$$

Since M accepts with positive amplitude, the amplitude in state q_{n-a+i} will be nonnegative. If the nonhalting state q_{n+3i+1} contains a fixed amount of amplitude α, and the old accept state q_{n-a+i} contains β amplitude. Then, the averaging operation places $\frac{\alpha-\beta}{\sqrt{2}}$ amplitude in the accept state q_{n+3i}. Then, the cleanup operation places any amplitude remaining in the nonhalting state q_{n+3i+1} into the reject state q_{n+3i+2}.

We initialize M' in the same way as M except that a fraction of the amplitude is placed in the new nonhalting states. These states behave as reservoirs until the end-marker is read. The amount of amplitude placed in the states is greater than the maximum amount of amplitude that any accepting state may ever contain, i.e., $\alpha > \beta \geq 0$.

If $x \in L$, then at least one of the accept states of M will contain a minimum amount of positive amplitude. Hence, the amount of amplitude in at least one of the accept states of M' will be strictly less than $\frac{\alpha}{\sqrt{2}}$ by some fixed amount. If $x \notin L$, then all accept states of M' will have exactly $\frac{\alpha}{\sqrt{2}}$ amplitude in them.

Hence, if $x \in L$, the probability of M' accepting x will be strictly less than that if $x \notin L$. Hence, M' accepts $\overline{L}$ with bounded error. Since the accept states are only observed after the end-marker is read, M' is end-decisive. □

If L is a language that can be accepted by an end-decisive MM-QFA with bounded error and L' is a language that can be accepted by an end-decisive MM-QFA with bounded positive one-sided error, then we use Lemma 4.10 to construct an MM-QFA that accepts the intersection of the two languages.

LEMMA 4.14. *Let M be an end-decisive MM-QFA that accepts L with bounded error and let M' be an end-decisive MM-QFA that accept L' with bounded positive one-sided error. There exists an MM-QFA M'' that accepts $L'' = L \cap L'$ with bounded error.*

Proof. Let MM-QFA M accept L with cut-point λ, margin ϵ, and maximum margin η, and let MM-QFA M' accept M' with cut-point λ', margin ϵ', and maximum margin η'.

First, consider the inequalities in Lemma 4.8 that occur when we compose the MM-QFAs M and M' using the tensor technique. Since MM-QFA M' accepts with bounded positive one-sided error, the inequalities are

$$\begin{aligned}
(\lambda+\epsilon)\cdot(\lambda'+\epsilon') \leq \Pr[N(x) = \text{accept}] \leq (\lambda+\eta)\cdot(\lambda'+\eta') \quad & \forall x \in L \cap L', \\
(\lambda-\eta)\cdot(\lambda'+\epsilon') \leq \Pr[N(x) = \text{accept}] \leq (\lambda-\epsilon)\cdot(\lambda'+\eta') \quad & \forall x \in \overline{L} \cap L', \\
(\lambda+\epsilon)\cdot 0 \leq \Pr[N(x) = \text{accept}] \leq (\lambda+\eta)\cdot 0 \quad & \forall x \in L \cap \overline{L'}, \\
(\lambda-\eta)\cdot 0 \leq \Pr[N(x) = \text{accept}] \leq (\lambda-\epsilon)\cdot 0 \quad & \forall x \in \overline{L} \cap \overline{L'}.
\end{aligned}$$

These reduce to three cases:

$$\Pr[M''(x) = \text{accept}] \geq (\lambda + \epsilon) \cdot (\lambda' + \epsilon') \quad \forall x \in L \cap L', \tag{4.6}$$

$$\Pr[M''(x) = \text{accept}] \leq (\lambda - \epsilon) \cdot (\lambda' + \eta') \quad \forall x \in \overline{L} \cap L', \tag{4.7}$$

$$\Pr[M''(x) = \text{accept}] = 0 \quad \forall x \in \overline{L'}. \tag{4.8}$$

If we can guarantee that

$$(\lambda - \epsilon) \cdot (\lambda' + \eta') < (\lambda + \epsilon) \cdot (\lambda' + \epsilon'),$$

then the tensor technique is sufficient to construct the intersection. Let M_n be the nth tensor composition of M. By Lemma 4.10 M_n accepts words in L with probability at least $(\lambda+\epsilon)^n$ and accepts words not in L with probability at most $(\lambda-\epsilon)^n$. Construct MM-QFA M'' by composing M_n with M' using the tensor technique; for sufficiently large constant n the inequality

$$(\lambda - \epsilon)^n \cdot (\lambda' + \eta') < (\lambda + \epsilon)^n \cdot (\lambda' + \epsilon')$$

will be satisfied. Thus, MM-QFA M'' accepts L'' end-decisively with bounded error. □

We now assemble our array of tools to construct an arbitrary Boolean combination of partial piecewise testable sets.

THEOREM 4.15. *Piecewise testable sets can be accepted by end-decisive MM-QFAs with bounded error.*

Proof. Let L be a piecewise testable set. We first rewrite it in canonical form:

$$\begin{aligned}
L &= \bigcup_{i=0}^{s} \bigcap_{j=0}^{t} \tilde{L}_{ij} \\
&= \bigcup_{i=0}^{s} \left(\cap_{j=0}^{r} L_{ij} \bigcap \cap_{j=r}^{t} \overline{L}_{ij} \right) \\
&= \bigcup_{i=0}^{s} \left(\cap_{j=0}^{r} L_{ij} \bigcap \overline{\cup_{j=r}^{t} L_{ij}} \right) \\
&= \bigcup_{i=0}^{s} \left(L^{\cap}{}_{i} \bigcap \overline{L^{\cup}{}_{i}} \right) \\
&= \bigcup_{i=0}^{s} L_i.
\end{aligned}$$

By Theorem 4.7 we can construct end-decisive MM-QFAs that accept partial piecewise testable sets, L_{ij}, with bounded positive one-sided error. Using these constructions and Corollaries 4.9 and 4.12, we can construct end-decisive MM-QFAs that accept languages $L^{\cap}{}_{i}$ and $L^{\cup}{}_{i}$ with bounded positive one-sided error.

The constructions in Theorem 4.7 channel only nonnegative amplitude into their accept states; furthermore, the constructions in Lemmas 4.8 and 4.11 do not negate amplitude. Consequently, the constructions for languages $L^{\cap}{}_{i}$ and $L^{\cup}{}_{i}$ channel only positive amplitude into their accept states. Hence, said constructions accept with positive amplitude. Since $L^{\cup}{}_{i}$ is also accepted with bounded positive one-sided error, by Lemma 4.13, we can construct an end-decisive MM-QFA that can accept $\overline{L^{\cup}{}_{i}}$ with bounded error.

Since $L^{\cap}{}_{i}$ is accepted by an end-decisive MM-QFA with bounded positive one-sided error and $\overline{L^{\cup}{}_{i}}$ is accepted by an end-decisive MM-QFA with bounded error, by Lemma 4.14, we can construct an end-decisive MM-QFA that accepts $L_i = L^{\cap}{}_{i} \cap \overline{L^{\cup}{}_{i}}$

with bounded error.

Since the languages L_i can be accepted by end-decisive MM-QFAs with bounded error, by Lemma 4.11, we can construct an end-decisive MM-QFA that accepts $L = \cup_i L_i$ with bounded error. □

5. Conclusions. We defined two models of 1QFA: the measure-once model that performs one measurement at the end of the computation, and the measure-many model that performs a measurement after every transition. The measure-many model is strictly more powerful than the measure-once but is more difficult to characterize.

When restricted to accepting with bounded error, measure-once automata can accept only group languages, while unrestricted measure-once automata can accept irregular sets and, in particular, can solve the word problem on the free group. Any language accepted by an MO-QFA can also be accepted by a PFA; therefore PFAs can also solve the word problem on the free group. We also sketched an algorithm for determining equivalence of two MO-QFAs.

The measure-many automaton is difficult to characterize. We have shown that the two classes of languages, those accepted with and without bounded error, are closed under complement and inverse homomorphisms; it is still an open question if these classes are closed under Boolean operations. We defined the partial order condition for languages and proved that it is a necessary condition for a language to be accepted by an MM-QFA with bounded error. We also showed that piecewise testable sets can be accepted with bounded error by MM-QFAs, and in the process detailed several novel construction techniques.

We do not know if the partial order condition is also a sufficient condition for bounded acceptance. If it is, then the two classes of languages accepted by an MM-QFA are closed under intersection.[1]

Appendix A. End-marker theorems.

THEOREM A.1. *Let M be an MO-QFA that has both left and right end-markers. There exists an MO-QFA M' that uses only one end-marker and is equivalent to M.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an MO-QFA with left and right end-markers, effectively allowing M to start in any possible configuration. Let the symbol $\not\in$ denote the left end-marker. Define $M' = (Q, \Sigma, \delta', q_0, F)$ from M. Let δ be defined in terms of the transition matrices $\{U_\sigma\}_{\sigma\in\Sigma}$. We define δ' from δ in the following way: for every $\sigma \in \Sigma$ let

$$U'_\sigma = U_{\not\in}^{-1} U_\sigma U_{\not\in},$$

and let

$$U'_\$ = U_\$ U_{\not\in}.$$

Now consider what happens when M and M' read a string $x = x_1 \dots x_n$. Since

$$\begin{aligned} U'(x\$) &= U'_\$ U'_{x_n} \dots U'_{x_1} \\ &= U_\$ U_{\not\in} U_{\not\in}^{-1} U_{x_n} U_{\not\in} \dots U_{\not\in}^{-1} U_{x_1} U_{\not\in} \\ &= U_\$ U_{x_n} \dots U_{x_1} U_{\not\in} \\ &= U(\not\in x\$), \end{aligned}$$

[1] After this paper was submitted, Ambainis, Kikusts, and Valdats [4] showed that this condition is not sufficient and that the class **RMM** is not closed under intersection.

the probability of M accepting x is equal to the probability of M' accepting x. Thus one end-marker on the right suffices, and by symmetry one left end-marker would also suffice. Therefore, an MO-QFA starting in configuration $|q_0\rangle$ can simulate an MO-QFA starting in any arbitrary configuration. □

THEOREM A.2. *Let M be an MM-QFA that has both left and right end-markers. There exists an MM-QFA M' that uses only a right end-marker and is equivalent to M.*

Proof. Let $M = (Q, \Sigma, \delta, Q_{acc}, Q_{rej})$ be an MM-QFA that uses two end-markers and accepts L. Assume without loss of generality that

$$\begin{aligned} Q_{non} &= \{q_i \in Q \mid 0 \le i < n_{non}\}, \\ Q_{acc} &= \{q_i \in Q \mid n_{non} \le i < n_{acc}\}, \\ Q_{rej} &= \{q_i \in Q \mid n_{acc} \le i < n_{rej} = n = |Q|\}, \end{aligned}$$

which facilitates a simpler description of M'. We construct $M' = (Q', \Sigma, \delta', Q'_{acc}, Q'_{rej})$ that accepts L with only the right end-marker. Let $Q' = Q \cup \{q_n, q_{n+1}, \ldots, q_{2n-n_{non}}\}$, $Q'_{acc} = \{q_{n+i-n_{non}} \in Q' \mid q_i \in Q_{acc}\}$, and $Q'_{rej} = \{q_{n+i-n_{non}} \in Q' \mid q_i \in Q_{rej}\}$. Assume that δ is defined in terms of transition matrices $\{U_\sigma\}_{\sigma\in\Sigma}$. The construction of $\{U'_\sigma\}_{\sigma\in\Sigma}$ is similar to that in the proof of Theorem A.1. Let I_l represent an identity matrix of size l and $m = n - n_{non}$. We define δ' in terms of its unitary block matrices. For all $\sigma \in \Sigma$ let

$$U'_\sigma = \begin{bmatrix} U_{\not\sigma}^{-1} & \\ & I_m \end{bmatrix} S \begin{bmatrix} U_\sigma & \\ & I_m \end{bmatrix} \begin{bmatrix} U_{\not\sigma} & \\ & I_m \end{bmatrix},$$

$$U'_\$ = S \begin{bmatrix} U_\$ & \\ & I_m \end{bmatrix} \begin{bmatrix} U_{\not\$} & \\ & I_m \end{bmatrix},$$

where

$$S = \begin{bmatrix} I_{n_{non}} & & \\ & & I_m \\ & I_m & \end{bmatrix}$$

transfers (sweeps) all probability amplitude from states in the old halting states to the new halting states. The old halting states, those in Q_{acc} and Q_{rej}, are no longer halting states in M'. The operation of M' is similar to the operation of the QFA constructed in Theorem A.1. The "sweeping" operation saves the amplitude that was in the old states, while it performs the $U_{\not\sigma}^{-1}$ operation in the new halting states (since otherwise the $U_{\not\sigma}^{-1}$ would corrupt the amplitude stored in the original halting states). □

REFERENCES

[1] M. AMANO AND K. IWAMA, *Undecideability of quantum finite automata*, in Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, 1999, pp. 368–375.

[2] A. AMBAINIS, R. BONNER, R. FREIVALDS, AND A. KIKUSTS, *Probabilities to accept languages by quantum finite automata*, in Computation and Combinatorics, Lecture Notes in Comput. Sci. 1627, Springer, New York, 1999.

[3] A. AMBAINIS AND R. FREIVALDS, 1-*way quantum finite automata: Strengths, weaknesses and generalizations*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, November 1998, IEEE Computer Society, pp. 332–342.

[4] A. Ambainis, A. Ķikusts, and M. Valdats, *On the class of languages recognizable by* 1*-way quantum finite automata*, Lecture Notes in Comput. Sci. 2010, Springer, New York, 2001, pp. 75–86.
[5] A. Ambainis, A. Nayak, A. Ta-Shma, and U. Vazirani, *Dense quantum coding and a lower bound for* 1*-way quantum automata*, in Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, 1999, pp. 376–383.
[6] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.
[7] S. Eilenberg, *Automata, Languages and Machines*, Vol. B, Academic Press, New York, 1976.
[8] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
[9] A. Kondacs and J. Watrous, *On the power of quantum finite state automata*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, 1997, IEEE Computer Society, pp. 66–75.
[10] R. Lipton and Y. Zalcstein, *Word problem solvable in logspace*, J. ACM, 24 (1977), pp. 523–526.
[11] A. Meyer and C. Thompson, *Remarks on algebraic decomposition of automata*, Math. Systems Theory, 3 (1969), pp. 110–118.
[12] C. Moore and J. Crutchfield, *Quantum automata and quantum grammars*, Theoret. Comput. Sci., 237 (2000), pp. 275–306.
[13] A. Nayak, *Optimal lower bounds for quantum automata and random access codes*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, IEEE Computer Society.
[14] J. Ortega, *Matrix Theory*, Plenum Press, New York, 1987.
[15] A. Paz, *Introduction to Probabilistic Automata*, Academic Press, New York, 1971.
[16] D. Perrin, *Finite automata*, in Handbook of Theoretical Computer Science, Vol. B, J. van Leeuwen, ed., Elsevier Science, 1994, Chap. 1.
[17] J. Pin, *On languages accepted by finite reversible automata*, in Proceedings of the 14th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 267, Springer, New York, 1987, pp. 237–249.
[18] M. Rabin, *Probabilistic automata*, Inform. and Control (Shenyang), 6 (1963), pp. 230–245.
[19] I. Simon, *Piecewise testable events*, in Proceedings of the 2nd GI Conference, Lecture Notes in Comput. Sci. 33, Springer, New York, 1975.
[20] S. Wagon, *The Banach-Tarski Paradox*, Cambridge University Press, New York, 1985.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1479–1500

FAST GREEDY ALGORITHMS FOR CONSTRUCTING SPARSE GEOMETRIC SPANNERS*

JOACHIM GUDMUNDSSON†, CHRISTOS LEVCOPOULOS‡, AND GIRI NARASIMHAN§

Abstract. Given a set V of n points in $\mathbb{R}^d$ and a real constant $t > 1$, we present the first $O(n \log n)$-time algorithm to compute a geometric t-spanner on V. A geometric t-spanner on V is a connected graph $G = (V, E)$ with edge weights equal to the Euclidean distances between the endpoints, and with the property that, for all $u, v \in V$, the distance between u and v in G is at most t times the Euclidean distance between u and v. The spanner output by the algorithm has $O(n)$ edges and weight $O(1) \cdot wt(MST)$, and its degree is bounded by a constant.

Key words. computational geometry, sparse geometric spanners, cluster graph

AMS subject classifications. 68U05, 65D18

PII. S0097539700382947

1. Introduction.

Complete graphs represent ideal communication networks, but they are expensive to build; sparse spanners represent low-cost alternatives. The weight of the spanner network is a measure of its sparseness; other sparseness measures include the number of edges, the maximum degree, and the number of Steiner points. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and many other areas and have been a subject of considerable research [1, 2, 4, 8, 11].

Consider a set V of n points in $\mathbb{R}^d$, where the dimension d is a constant. A network on V can be modeled as an undirected graph G with vertex set V and with edges $e = (u, v)$ of weight $wt(e)$. A Euclidean network is a geometric network where the weight of the edge $e = (u, v)$ is equal to the Euclidean distance $d(u, v)$ between its two endpoints u and v. Let $t > 1$ be a real number. We say that G' is a *t-spanner* for V if, for each pair of points $u, v \in V$, there exists a path in G' of weight at most t times the Euclidean distance between u and v. A *sparse t-spanner* is defined to be a t-spanner of size (number of edges) $O(n)$ and weight (sum of edge weights) $O(1) \cdot wt(MST)$, where $wt(MST)$ is the total weight of a minimal spanning tree. Given a geometric network $G = (V, E)$, a (generic) weight function wt defined on its edges, and two vertices $u, v \in V$, we let $D_{\{G,wt\}}(u, v)$ denote the weight of the shortest path from u to v in G for the weight function wt.

The problem of constructing spanners has been investigated by many researchers. Levcopoulos and Lingas [10] presented an $O(n \log n)$-time algorithm that produced a sparse t-spanner for the two-dimensional case. It works by taking any t-spanner which has the form of a (possibly partial) triangulation and achieving almost the same t as that triangulation. However, the problem gets much more difficult in higher

*Received by the editors December 20, 2000; accepted for publication (in revised form) February 18, 2002; published electronically August 1, 2002.
http://www.siam.org/journals/sicomp/31-5/38294.html

†Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands (Joachim@cs.uu.nl). This author's research was funded by the Swedish Foundation for International Cooperation in Research and Higher Education.

‡Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden (Christos@cs.lth.se).

§School of Computer Science, Florida International University, Miami, FL 33199 (giri@cs.fiu.edu). Part of this author's work was done at the University of Memphis and at Lund University.

dimensions. There are several algorithms that run in time $O(n \log n)$ [3, 9, 15, 17]. However, they only guarantee a linear number of spanner edges, and they do not guarantee low weight. Das and Narasimhan [8] gave an $O(n \log^2 n)$-time algorithm that constructs, for any set V of n points in $\mathbb{R}^d$ and any constant $t > 1$, a sparse t-spanner for V in which the degree of every point is bounded by a constant. Chen, Das, and Smid [5] showed that the lower bound for computing any t-spanner for a given set of points V in R^d is $\Omega(n \log n)$ in the algebraic computation tree model.

Mount [12] showed that a significant result claimed in Arya et al. [2] of an $O(n \log n)$-time algorithm to compute a low-weight Euclidean spanner is incorrect. Thus the problem of devising an $O(n \log n)$-time algorithm to produce low-weight spanners remained unsolved.

Before a correct $O(n \log n)$-time algorithm was presented, sparse spanners were used in designing efficient approximation schemes for geometric problems. Rao and Smith [14] made a breakthrough by showing an optimal $O(n \log n)$-time approximation scheme for the well-known Euclidean *traveling salesperson problem*, assuming that it is possible to compute sparse spanners in time $O(n \log n)$. Also, Czumaj and Lingas [6] showed approximation schemes for minimum-cost multiconnectivity problems in geometric graphs that also depended on the assumption that sparse spanners could be computed in $O(n \log n)$ time. Thus the existence of an $O(n \log n)$-time algorithm to construct sparse spanners became a critical open problem. Note that the most efficient algorithm to construct sparse spanners is due to Das and Narasimhan [8] and runs in $O(n \log^2 n)$ time. In this paper we show the following theorem.

THEOREM 1. *Given a set V of n points in d-dimensional space and any real constant $t > 1$, in the algebraic decision tree model of computation, a sparse t-spanner of the complete Euclidean graph can be constructed in $O(\frac{n \log^2 n}{\log \log n})$ time. If the model is extended with indirect addressing, a sparse t-spanner of the complete Euclidean graph can be constructed in $O(n \log n)$ time. The constants implicit in the O-notation depend on t and d.*

It was shown in [8] that the greedy algorithm produces spanners with $O(n)$ edges and weight $O(wt(MST))$. However, a naive implementation of the greedy algorithm (shown in Figure 1) takes $O(n^3 \log n)$ time, mainly due to the fact that a quadratic number of shortest path queries are needed to be answered in a "dynamic" graph with $O(n)$ edges. Each of the queries takes $O(n \log n)$ time.

Our algorithm is inspired by the algorithm due to Das and Narasimhan [8]. They showed how to use clustering in order to speed up shortest path queries; i.e., they showed that approximate shortest path queries sufficed to produce sparse spanners. However, their algorithm was not efficient enough because they were unable to main-

---

Algorithm STANDARD-GREEDY(G, t)
1. sort the edges in E by increasing weight
2. $E' := \emptyset$
3. $G' := (V, E')$
4. **for** each edge $(u, v) \in E$ **do**
5. **if** SHORTESTPATH$(G', u, v) > t \cdot d(u, v)$ **then**
6. $E' := E' \cup \{(u, v)\}$
7. $G' := (V, E')$
8. output G'

FIG. 1. *The naive $O(n^3 \log n)$-time greedy spanner algorithm.*

tain the clusters efficiently, and the algorithm had to frequently rebuild the clusters. For convenience, we will refer to the $O(n \log^2 n)$-time algorithm from [8] as the *DN-clustering* spanner algorithm. We retain the general framework of that algorithm. Our main contribution is in developing techniques to efficiently perform clustering. We believe that the techniques that we have developed are likely to be useful in designing other greedy-style "dynamic algorithms," i.e., in situations where only insertions take place and particularly in increasing order of length. What we prove in this paper is that, after some preprocessing, given a linear-sized edge-weighted graph with integral edge weights in the range $[0, N]$, and given a set of cluster centers, one can perform clustering very efficiently in only $O(n + N)$ time.

The terms length and weight are used interchangeably throughout the paper.

2. An improved spanner algorithm. We first describe the previous cluster-based spanner algorithm due to Das and Narasimhan [8]. Then, in section 2.2, we present a simple modification to the DN-clustering algorithm to construct sparse t-spanners.

2.1. The DN-clustering spanner algorithm. The algorithm by Das and Narasimhan [8] can roughly be described as follows.

The algorithm starts with an empty spanner G'. A preprocessing step helps to eliminate all but a linear number of edges from further consideration. For a given value of t, this step is performed by a call to an $O(n \log n)$-time algorithm presented by Salowe [15] or Arya et al. [2] to compute a spanner with stretch factor $\sqrt{t/t'}$ (for some $t > t' > 1$). Among the edges not eliminated, very short edges (i.e., those of length at most D/n, where D is the distance between the farthest pair of points) are simply added to G' since their contribution to the overall weight of the spanner cannot be more than the weight of a minimum spanning tree, $wt(MST)$. For the remaining edges, the greedy algorithm is simulated by sorting the edges (by increasing weight) and then processing them in $\log n$ phases. Greedy processing of an edge $e = (u, v)$ entails a shortest path query, i.e., checking whether $D_{\{G', wt\}}(u, v) \leq \sqrt{tt'} \cdot wt(e)$. If the answer to the query is no, then edge e is added to the spanner G', or else it is discarded. Whenever shortest path queries are required to be answered, these are not solved on the spanner G' being constructed. Instead, they are solved on a cluster graph H, which is simultaneously maintained. A set of points $C \subseteq V$ is a cluster of radius r with cluster center $v \in C$ if, for every point $u \in C$, there is a path in G' between v and u of length at most r. A set of clusters $C_1, \ldots, C_k$ is a cluster cover of G' if every point in V belongs to at least one cluster. A cluster graph H can be constructed from a cluster cover by adding two types of edges: intracluster edges (edges connecting the cluster center of a cluster C to all other vertices in C) and intercluster edges (edges connecting two cluster centers). The cluster graph H from [8] has the following properties:

1. distances in H "approximate" distances in the current spanner graph G',
2. every vertex in H has bounded degree, and
3. "specialized" shortest path queries in H can be answered in $O(1)$ time.

Item 1 was demonstrated by showing that corresponding distances in H and G' differ by only a small constant factor. The shortest path query when processing edge $e = (u, v)$ is "specialized" in the sense that, at the instant that this query is processed, the cluster graph H has only edges between clusters (the so-called intercluster edges) whose lengths are within a constant factor of $wt(e)$.

In order to understand how shortest path queries can be answered efficiently, we note that, in an unweighted graph with bounded degree, checking whether the

distance between two vertices u and v is at most a constant $\sqrt{tt'}$ can be achieved in $O(1)$ time (since there are only $O(1)$ vertices at the distance t from u). Thus, for all practical purposes, the cluster graph H behaves like an unweighted graph of bounded degree for which a bounded radius subgraph around vertex u needs to be searched for the presence of vertex v. It is thus easily shown that specialized shortest path queries can be answered in $O(1)$ time.

Since the edges considered have weights in the range $(D/n, D]$ and they are processed in $\log n$ phases, the edges can be sorted into $\log n$ bins, where the ith bin has edges of weight in the range $(2^{i-1} \cdot D/n, 2^i \cdot D/n]$. In order for shortest path queries to be answered quickly, the cluster graph has to be carefully maintained. At the end of each phase, the cluster graph is recomputed from scratch using the graph G'. This was deemed necessary since, in order to answer specialized shortest path queries about edge $e{=}(u, v)$ in constant time, all intercluster edges in H need to be of length within a constant factor of $d(u, v)$.

The time complexity analysis is straightforward. Preprocessing steps ran in $O(n \log n)$ time. The $O(n)$ shortest path queries were processed in $O(n)$ time, since each query took only $O(1)$ time. The cluster graph computation at the start of each phase took $O(n \log n)$ time (since it involved running Dijkstra's shortest path algorithm on linear-sized graphs starting from sequentially selected cluster centers). Since there were $\log n$ phases, the cluster graph computations took a total of $O(n \log^2 n)$ time. The crucial observation made in [8] was that shortest path queries need not be answered precisely. Instead, approximate shortest path queries suffice to produce low-weight spanners. The second observation was that shortest path queries are expensive if the shortest path involves a number of short edges and that clustering can help to eliminate all short edges. This, of course, meant that the greedy algorithm, too, was only approximately simulated by the algorithm.

2.2. A faster spanner algorithm. In this section, we present a simple modification to the DN-clustering algorithm to construct sparse t-spanners. This algorithm improves on the time complexity of the DN-clustering algorithm and runs in time $O(\frac{n \log^2 n}{\log \log n})$ in the algebraic decision tree model of computation.

First we observe that there is wide disparity in the overall time spent by the DN-clustering algorithm on shortest path queries ($O(n)$) and the time spent on the cluster graph computations ($O(n \log^2 n)$). In order to balance the two costs, it is necessary to do fewer than $O(\log n)$ cluster graph computations. This in turn would make the shortest path queries more expensive because it increases the ratio between the lengths of the longest and shortest edges in the cluster graph, which implies that the number of edges along the shortest path between two cluster centers will also increase, and therefore the query time will also increase. Instead of processing the edges in $\log n$ phases, we process them in $\frac{4 \cdot d \cdot \log n}{\log \log n}$ batches. We use the term batches to distinguish from the word phases used by the earlier DN-clustering algorithm.

If the clustering is recomputed after processing every batch of edges, the total time for cluster graph computations will be $O(\frac{n \log^2 n}{\log \log n})$, since each call to the clustering algorithm takes $O(n \log n)$ time. We carefully analyze the cost of the $O(n)$ shortest path queries and show that it can now be answered in a total of $O(n \log n)$ time. In phase i of the DN-clustering algorithm, edges from the ith bin were processed. These edges had weights in the range $(W, 2W]$, where $W = 2^{i-1}(D/n)$. During phase i, the cluster graph H could have intercluster edges whose weights were in the range $(\delta W, 2W(1+2\delta)]$, where $\delta < \frac{1}{2}$ is a positive constant. This meant that, for edge (u, v) of weight $l \in (W, 2W]$, checking whether there is a path from u to v of length at most

$t \cdot l$ could be done in $O(1)$ time. More precisely, it was observed in [8] that, if there exists a path from u to v of length at most $t \cdot l$, then the number of edges on this path can be at most $\frac{2t}{\delta}$. It was further observed that, since the vertices of H had a constant degree bound (say, c), and since there are at most $O(c^{\frac{2t}{\delta}})$ vertices that lie $\frac{2t}{\delta}$ edges away from vertex u, this shortest path query could be done in $O(c^{\frac{2t}{\delta}} \log c^{\frac{2t}{\delta}})$ time. A tighter analysis was unnecessary in the DN-clustering algorithm of [8] since c, t, and δ were all constants; below we show an improved analysis of this cost.

Recall that our algorithm works in $\frac{4 \cdot d \cdot \log n}{\log \log n}$ batches. Batch i of our algorithm can be described as follows. For $W = 2^{\frac{(i-1) \cdot \log \log n}{4 \cdot d}}(D/n)$, the edges processed in batch i have weights in the range $(W, W2^{\frac{\log \log n}{4 \cdot d}}]$; i.e., they are in the range $(W, W(\log n)^{\frac{1}{4 \cdot d}}]$. Thus, for edge (u, v) of weight $l \in (W, W(\log n)^{\frac{1}{4 \cdot d}}]$, we need to check whether there is a path from u to v of length at most $\sqrt{tt'} \cdot l$. During batch i, the cluster graph H can have intercluster edges with weights in the range $(\delta W, (1 + 2\delta)W(\log n)^{\frac{1}{4 \cdot d}}]$. Thus, if there does exist such a path from u to v, then the number of edges on this path can be at most $\frac{\sqrt{tt'}(\log n)^{\frac{1}{4 \cdot d}}}{\delta}$. The crucial observation we make is that the vertices of the cluster graph correspond to clusters of radius δW. These clusters may overlap, but their centers can lie in only one cluster. In other words, if these clusters are shrunk in half, they do not intersect. Thus the vertices correspond to disjoint clusters of radius $\delta \cdot W/2$. Now it is possible to bound the number of vertices within distance at most $\sqrt{tt'} \cdot l = \sqrt{tt'}W(\log n)^{\frac{1}{4 \cdot d}}$. Packing arguments [8] show that, in $\mathbb{R}^d$, the number of balls of radius r that can be packed in a ball of radius R is bounded by $O((R/r)^d)$. Thus the number of balls of radius $r = \frac{\delta W}{2}$ that can be packed in a ball of radius $R = \sqrt{tt'}W(\log n)^{\frac{1}{4 \cdot d}}$ is at most $O((\frac{t \cdot (\log n)^{\frac{1}{4 \cdot d}}}{\delta})^d)$. Due to the constant degree, the maximum number of vertices and edges that can be reached when performing Dijkstra's algorithm starting from vertex u is $O((\frac{t \cdot (\log n)^{\frac{1}{4 \cdot d}}}{\delta})^d) = O((\log n)^{\frac{1}{4}})$ (since t, d, and δ are constants). We conclude that Dijkstra's algorithm for a shortest path query has a time complexity of $O((\log n)^{\frac{1}{4}} \cdot (\log((\log n)^{1/4}))) = O(\log n)$. Thus all $O(n)$ shortest path queries can be answered in $O(n \log n)$ time. Note that, even though the clustering and shortest path costs are not precisely balanced, it is possible to prove (using lengthy but straightforward algebraic calculations) that (asymptotically) it cannot be improved. The obtained spanner satisfies the leapfrog property [8] (also defined in section 5), which implies that the weight of the spanner is $O(wt(MST)$.

THEOREM 2. *In the algebraic decision tree model of computation, given a set V of n points in d-dimensional space and any real constant $t > 1$, a sparse t-spanner of the complete Euclidean graph can be constructed in $O(\frac{n \log^2 n}{\log \log n})$ time. The constants implicit in the O-notation depend on t and d.*

3. A fast spanner algorithm that uses indirect addressing. In the rest of the paper, we describe an efficient algorithm to construct sparse spanners with a running time of $O(n \log n)$. This algorithm is also inspired by the DN-clustering algorithm in [8]. As explained in section 2.1, the reason their algorithm runs in time $O(n \log^2 n)$ is that the clustering step takes $O(n \log n)$ time per phase. The running time for our algorithm is achieved by designing a linear time algorithm for an "approximate" version of the clustering step, thus executing all the clustering steps in $O(n \log n)$ total time.

One crucial idea that we employ to speed up the clustering is to replace the real-valued edge weights by integral values. As observed in [8], the shortest path

queries required by the algorithm need not be answered precisely; approximately correct answers suffice. A convenient way to achieve the *integralization* is to use the *floor/ceiling* function. However, this assumes a more powerful model of computation. In order to get around this problem, we reduce the dependence of the algorithm on the floor/ceiling function and compute the floor/ceiling function by using operations allowed under the algebraic computation tree model extended with indirect addressing. The second crucial component of our algorithm is an implementation of the clustering algorithm in $O(n)$ time assuming small integral edge weights for the edges. We also prove that the integralization introduces only a bounded amount of error and that this error bound helps to prove the correctness of the other required operations. The third and final crucial component in our algorithm is that we show how it is possible to select the cluster centers for each stage of the algorithm in linear time.

The improved spanner algorithm can be roughly described as follows; see Figure 2. It is important to note that the skeleton of the algorithm is similar to the DN-clustering algorithm from [8]. In particular, this improved algorithm also runs in $O(\log n)$ phases. If a fewer number of phases are used, then the error due to integralization could be too large. Even if a fewer number of phases can be used, the running time of the overall algorithm will remain as $O(n \log n)$, since it is dominated by other steps in the algorithm. In particular, the integralization itself has an initial cost of $O(n \log n)$.

The algorithm starts with an empty spanner G' and employs the same first (preprocessing) step to eliminate all but a linear number of edges. For a given value of t, this step is performed by a call to an $O(n \log n)$-time algorithm presented by Arya et al. [2] to compute a spanner with stretch factor $\sqrt{t/t'}$ (for some $t > t' > 1$) and with bounded degree. As in [8], in the next step, short edges of length at most D/n are simply added to G'; their contribution to the overall weight of the spanner is bounded by $O(wt(MST))$. The greedy algorithm is then simulated on the remaining edges of the initial spanner.

The edges of the graph have real-valued weights that are equal to the Euclidean distance between their endpoints. The edges are sorted by increasing weight and then processed in $\log n$ phases. Each of the edges in the spanner graph also have corresponding integer-valued weights that are sufficiently close approximations of the real-valued weights; these integer-valued weights change through the course of the algorithm, becoming coarser and coarser approximations as the algorithm progresses. In order to distinguish between the real- and integer-valued weights, we assume that there are two different weight functions defined on the edges of G'. For edge $e = (u, v)$, the real-valued weight function $wt(e)$, as mentioned before, is defined as the Euclidean distance $d(u, v)$ between u and v. The integer-valued weight function, denoted by $Iwt_i(e)$, is a function of $wt(e)$ and the phase number i. It is maintained during the execution of the algorithm, as will be described later. Whenever the phase number is clear from the context, we use the simpler notation $Iwt(e)$ instead of $Iwt_i(e)$. Also, unless specified otherwise, we assume that, when we refer to the weight of an edge, we are referring to the real-valued weight of the edge.

At the start of each phase, the integer-valued weight function $Iwt(e)$ is recomputed for this phase. Then a set of vertices of G' are selected as cluster centers, and a cluster graph H is constructed from the current spanner graph G', using the weight function Iwt. This cluster graph H is a simpler graph than the graph G', and distances between vertices in H are reasonably close to distances between the same pair of vertices in G'. Clustering is made more precise in section 3.2. The difference

between this and the one in [8] lies in the fact that the cluster centers have to be selected before the clustering is done, and the clustering is done with the weight function Iwt. As mentioned before, we improve on the time complexity of this clustering step and show how it can be implemented to run in $O(n)$ time. Once the cluster graph H is constructed, the algorithm processes the set of edges for that phase. Greedy processing of an edge $e = (u, v)$ entails, as before, a shortest path query, i.e., checking whether $D_{\{G',wt\}}(u, v) \leq t \cdot wt(e)$. We answer an approximate version of this query, i.e., performing a shortest path query on the simpler graph H and not on the partial spanner graph G'. If the answer to the approximate query is "no," edge e is added to the graph G'; otherwise, it is discarded. Each of the steps is described in more detail in the rest of the paper.

In section 3.1, we describe the integralization process and analyze the error due to it. The clustering algorithm is described in section 3.2, and, finally, in section 3.3, we describe how to compute the shortest paths in the cluster graph and prove that the total running time of the algorithm is $O(n \log n)$.

The detailed algorithm is given in Figure 2. The inputs are V, which is a set of n points in d-dimensional space, and two constants t and t' such that $1 < t' \leq t$. As one can see, it is similar to the DN-clustering algorithm except for the integralization steps (steps 10, 11, 14, and 22) and the computation of the cluster centers (steps 12 and 21). In sections 4 and 5, we will show that the output G' indeed is a t-spanner and that a suitable selection of the input parameter t' will guarantee that G' has small weight. Recall that the truly time-critical step of this algorithm is the clustering step (step 15) and selecting the new cluster centers for the next phase (step 21). Both of these steps will be closely described in section 3.2. Note that the two values bounding δ are decided in Lemmas 14 and 17.

3.1. Integralization. As mentioned before, in order to speed up the cluster graph computation, we replace the real-valued edge weights by integral values. The integralization changes in every phase. It is done in such a way that the edge weights and distances encountered in that phase are always in the range $[0, N]$, where $N = c \cdot n$ for some constant integer c. The choice of c will dictate the errors introduced in the distance computations; this will be discussed later.

A closer inspection of a phase leads to the following simple observations. At the start of phase i, the spanner graph constructed so far has edges of weight at most W_i. During phase i, the edges considered for inclusion by the greedy algorithm are in the range $(W_i, 2W_i]$. The shortest path query for an edge of length l involves checking whether the distance between a given pair of vertices is at most $t \cdot l$. Hence the longest paths that need to be dealt with during phase i are of weight $t \cdot 2W_i$. The idea is to make the largest distance we consider in phase i correspond to the integer $c \cdot n$. To be on the safe side, since there are small errors in the distance computations, we set $2(t \cdot 2W_i)$ to correspond to $c \cdot n$. Thus, in phase i, the unit integer length will correspond to the real length of $U_i = \frac{4 \cdot t \cdot W_i}{c \cdot n}$.

Although a constant-time floor/ceiling function is not used in the algorithm, a convenient way to describe the integralization is as follows:

$$Iwt_i(e) := \left\lceil \frac{wt(e)}{U_i} \right\rceil.$$

We will describe below how the integralization step is performed.

3.1.1. Error bounds. As defined above, we observe that the integralization function Iwt always involves a rounding up ($Iwt_i(e) \cdot U_i \geq wt(e)$). Thus, in phase i, the

Algorithm IMPROVED-GREEDY(V, t, t')

Compute a $(\sqrt{t/t'})$-spanner $G = (V, E)$ using the algorithm from [2]
$\delta := \min\left(\frac{\sqrt{tt'}-(1+\epsilon)t'}{2(1+\epsilon)(\sqrt{tt'+3t'})}, \frac{\sqrt{tt'}-(1+\epsilon)}{2(\sqrt{tt'}(1+\epsilon)+5+7\epsilon+2\epsilon^2)}\right)$
$D :=$ length of longest edge in E
$E' = \{e \in E \mid wt(e) < D/n\}$
$G' := (V, E')$
$W_i := 2^{(i-1)}D/n$ for $i = 1, 2, \ldots, \log n$
$r := \lceil \frac{n}{\epsilon} \rceil$; $R = \lceil \frac{n}{\epsilon\delta} \rceil$; COMMENT: R & r are integral versions of W_i & δW_i.
for $i := 1$ to $\log n - 1$ **do**
    $E_i :=$ set of (sorted) edges of E with weights in $(W_i, W_{i+1}]$
Build Integer tree with values $\{1, \ldots, cn\}$
INTEGRALIZE$(E', 1)$
$C_1 :=$ NAIVE-CENTERS$(G', \delta W_1)$;
for $i := 1$ to $\log n$ **do**
    INTEGRALIZE(E_i, i)
    $H :=$ CLUSTER-GRAPH(G', Iwt, C_i, r, R)
    for each edge $e = (u, v) \in E_i$ in increasing order **do**
        if not SHORT-PATH$\left(H, u, v, \frac{n\sqrt{tt'}d(u,v)}{\epsilon\delta W_i}\right)$ **then**
            $E' := E' \cup \{e\}$
            $G' := (V, E')$
            INTEREDGESTYPE2(u, v)
    $C_{i+1} :=$ UPDATE-CENTERS(H, i, C_i, r)
    REINTEGRALIZE(E')
output G'

FIG. 2. *The $O(n \log n)$-time spanner algorithm.*

error in the length of any single edge is at most U_i. In other words, $Iwt_i(e) \cdot U_i - wt(e) \leq U_i$. Note that this error is an additive or an absolute error. Since any simple path can use at most $n-1$ edges, the error in the length of any simple path of the spanner graph is less than nU_i. Another consequence is that, given two simple paths P_1 and P_2, if $Iwt(P_1) = Iwt(P_2)$, then $|wt(P_1) - wt(P_2)| < nU_i$. It follows that nU_i is also a bound on the error that can be introduced when running Dijkstra's single-source shortest path algorithm using the integral weights instead of the real weights. The following lemma formalizes this statement.

LEMMA 3. *In phase i, if $D_{\{G', wt\}}(u, v) > W_i$ for some $u, v \in G'$, then*

$$D_{\{G', wt\}}(u, v) \leq D_{\{G', Iwt\}}(u, v) \cdot U_i < \left(1 + \frac{4t}{c}\right) \cdot D_{\{G', wt\}}(u, v).$$

Proof. Since $wt(e) \leq Iwt_i \cdot U_i = \lceil \frac{wt(e)}{U_i} \rceil \cdot U_i < wt(e) + U_i$, we have

$$\begin{aligned} D_{\{G', wt\}}(u, v) &\leq D_{\{G', Iwt\}}(u, v) \cdot U_i \\ &< D_{\{G', Iwt\}}(u, v) + nU_i \\ &= D_{\{G', wt\}}(u, v) + \frac{4tW_i}{c} \\ &< \left(1 + \frac{4t}{c}\right) D_{\{G', wt\}}(u, v). \qquad \square \end{aligned}$$

As a direct consequence, we obtain the following important corollary.

COROLLARY 4. *For a path P in G' with $wt(P) \geq \delta W_i$, the absolute error in computing its weight is at most nU_i, and the relative error is at most $\frac{nU_i}{\delta W_i} = \frac{4t}{c\delta}$.*

3.1.2. Computing the integralization. Here we show how to compute the integer values of the weights of the edges over all phases in $O(n \log n)$ total time without using the floor/ceiling function.

We first observe that the spanner graph has at most $O(n)$ edges at the start of any phase. Consider a specific phase i. In this phase, for a specific edge e, since its integer value is in the range $[0, N]$ (where $N = c \cdot n$), $Iwt(e)$ can be computed in $O(\log n)$ time without the use of the floor/ceiling function by performing a binary search on the set of real values $j \cdot U_i$, for $j = 0, \ldots, N$. We assume that the function INTEGRALIZE(E_i, i) performs this operation for each edge in the set E_i in $O(\log n)$ time per edge.

If the above observations are used in a naive fashion for all edges, then the cost of integralization is $O(n \log n)$ just for one phase. Since the number of phases is not constant, the integralization would turn out to be too expensive. We have to show that the algorithm spends $O(\log n)$ time for computing the integralization of an edge weight over all the phases. The idea is to compute the integral value in $O(\log n)$ time when the edge is encountered for the first time. Integralizations of an edge for subsequent phases is done by calling REINTEGRALIZE, and are computed in constant time from the integer weights of the edge computed in the previous phase. If the integral weight of an edge is I in phase i, then the integral weight of the edge in phase $i+1$ will be $I/2$ if I is even and $(I+1)/2$ if it is odd. This is correct since $U_{i+1} = 2U_i$; i.e., the integralization in phase $i+1$ is twice as coarse as that in phase i. Checking if an integer is odd or even cannot be done in constant time in our model but can be easily accomplished by using $O(n)$ preprocessing. One way to accomplish this would be to build a balanced leaf-oriented binary tree including $c \cdot n$ leaves with the values $1, \ldots, cn$. Every element in the tree, with value val, also contains a pointer to the element in the tree containing the value $\lceil \frac{val}{2} \rceil$. Assume for simplicity that $c \cdot n = 2^{c'}$. The tree can be built top-down in linear time. The root will have value 1. Consider a node v with value ℓ. The left child of v will have the value $2\ell - 1$, and the right child will have value 2ℓ. This step is repeated until all of the leaves are at level c'. Hence, by using $O(n)$-time preprocessing, the integral weight of an edge for the next phase can be computed in constant time. Another way to handle this problem would be to extend the model of computation with trigonometric functions, i.e., the sine function.

Note also that the relative error for an edge with newly computed weight is less than U_{i+1}; hence Lemma 3 still holds. It is clear that REINTEGRALIZE(E') performs its operation for each edge in the edge set E' in $O(1)$ time per edge.

The above explanation proves that the integralization is computed in $O(n \log n)$ time for all edges over all phases. The integer weights are then used directly in the clustering algorithms described below.

3.2. Clustering the graph. Now we turn our attention to the main contribution of this paper, namely, how to construct a cluster graph in linear time. First we have some definitions. Here we assume that $G = (V, E)$ is a metric graph with a weight function w defined on its edges E. The following definition of a cluster is modified from the one in [8] to allow for arbitrary weight functions. The definition of a cluster cover is also modified and is defined for a given set of cluster centers. Figure 3 illustrates a cluster and a cluster cover.

DEFINITION 5 (cluster, cluster center, and radius). *Given a vertex $v \in V$ and a nonnegative real value r,* CLUSTER(G, v, r, w) *is defined as the set of all vertices $U \subseteq V$ such that $D_{\{G,w\}}(v, u) \leq r$ for all $u \in U$. The vertex v is called the* cluster center *of this cluster, and r is called the* radius *of the cluster.*

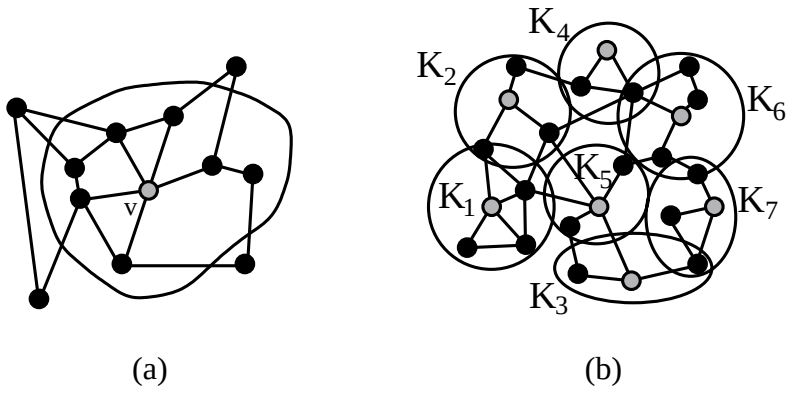


FIG. 3. (a) *A cluster with center at v and radius r.* (b) *The clusters $K_1, \ldots, K_7$ form a cluster cover.*

DEFINITION 6 (cluster-cover). *Given a set of cluster centers $C = \{v_1, \ldots, v_m\} \subseteq V$ and a radius r, the* CLUSTER-COVER(G, C, r, w) *(if it exists) is a set of clusters $K = \{K_1, \ldots, K_m\}$ such that K_i, for $1 \leq i \leq m$, is a cluster with radius r and cluster center v_i and such that $K_1 \cup K_2 \cup \cdots \cup K_m = V$.*

The set C and the radius r will be chosen in such a way that the cluster cover always exists. In general, clusters in a cluster cover may overlap. In our algorithm, the cluster centers will be reasonably far apart so that the amount of overlap is limited. We also modify the definition from [8] of a cluster graph so that it is a bit more general and is defined for a given set of clusters and for an arbitrary weight function.

DEFINITION 7 (cluster graph). *Assume that $C = \{v_1, v_2, \ldots, v_m\} \subseteq V$ is a given set of cluster centers. For a given radius r, we assume that $K = \{K_1, K_2, \ldots, K_m\}$ is equal to* CLUSTER-COVER(G, C, r, w). *Given a second radius $R > r$,* CLUSTER-GRAPH(G, w, C, r, R) *is defined as a graph $H = (V, E_H)$ with a weight function w defined on its edges E_H. The weight of an edge $[u, v]$ in E_H is defined to be equal to $D_{\{G,w\}}(u, v)$. (We use square brackets to distinguish cluster graph edges from the edges of G.) The edges of H are defined as follows.*

Intracluster edges. *For all K_i and for all $u \in K_i$, $[u, v_i] \in E_H$.*

Intercluster edges. *For all $v_i, v_j \in C$, $[v_i, v_j]$ is an intercluster edge if either*

1. *$v_i \notin K_j$ and $v_j \notin K_i$ and $D_{\{G,w\}}(v_i, v_j) \leq R$ (type 1), or*
2. *there exists $e = (u_i, u_j) \in E$ such that $u_i \in K_i$ and $u_j \in K_j$ (type 2).*

3.2.1. Computing the cluster cover. Here we describe how the cluster cover is computed efficiently under some assumptions. Once a cluster cover is computed, we show later that it is straightforward to construct the cluster graph.

Note that the input to the cluster cover computation is a weighted graph $G = (V, E)$ with a weight function w defined on its edges, a set $C \subseteq V$ of cluster centers, and a radius R. We will assume that $|V| = n$, $|E| = O(n)$, the weight function w is an integral, and the radius R is an integer. Since we do not have to deal with distances greater than R, we can safely assume that the weight of any edge is an integer value in the range $[0, R]$. We will further assume that the cluster centers are chosen in such a way that a cluster cover exists, which will be shown in section 3.3. The obvious way to implement this algorithm is as it was done in [8], i.e., to run Dijkstra's

single-source shortest path algorithm from all the cluster centers and to compute the clusters in the cluster cover. However, this has a running time of $O(n \log n)$. In order to speed it up, we run Dijkstra's algorithm in *parallel* from all the cluster centers and use a simple and fast priority queue, which we denote by PQ. The priority queue we use is an array of size R, indexed from 1 to R, as shown in Figure 4. This is sufficient for our purposes because of the following reasons. First, the weight function is integral, and the array contains all possible distance values from the cluster centers to vertices in the clusters. Second, it is well known that, in Dijkstra's algorithm, once a vertex has been extracted from the priority queue, its distance from the source will never be updated again, and the distance from the source at the time of the extraction is the correct distance from the source. In other words, the minimum value of the items in the priority queue is monotonic. Since the priority queue is an array, EXTRACT-MIN can be implemented as a scan through the array for the "next" largest item.

One problem is that clusters can overlap and that vertices may have entries in the priority queue with distances from several cluster centers. Let σ denote the maximal number of clusters that a vertex may belong to. The problem can be taken care of by augmenting the priority queue entries to be a pointer to a linked list where every entry in the list also stores information about the vertex as well as the corresponding cluster center. Since a vertex belongs to at most σ clusters, the space complexity of the priority queue will be $O(n \cdot \sigma + R)$. Also, every vertex contains a list of the clusters it belongs to.

It should be noted that this version of Dijkstra's algorithm, as shown in Figure 5, needs to perform a number of RELAX steps and that in each such step the priority queue may need to be updated. The process of RELAXing an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating the value for v, i.e., adding a new entry and removing an old entry. Since every vertex contains information about which clusters it belongs to and the distance to each cluster center, each update is performed in time $O(\sigma)$. It should be pointed out that this is the only place where we are unable to eliminate the use of indirect addressing since it is critical that this update be performed efficiently, i.e.,

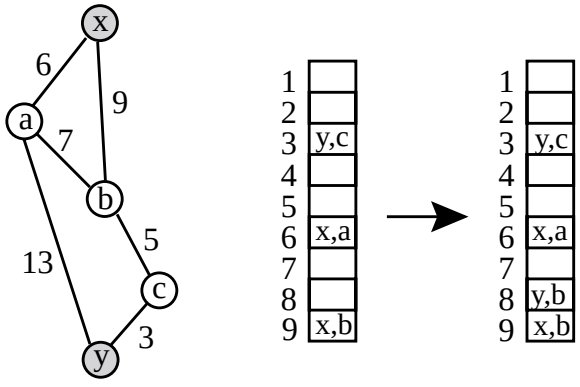


FIG. 4. *An example of how the cluster cover is computed with x and y as cluster centers and radius $R = 9$. The array to the left is the initial priority queue after all edges leading out of x and y have been processed. The array to the right shows the final priority queue. Note that, after vertex c was processed, the edge (c, b) was relaxed, and (y, b) was inserted into the priority queue with length 8.*

Algorithm ParallelDijkstra

```
1.  Q := Initialize(G', ClusterCenters)                    O(n)
2.  while Q ≠ ∅ do                                         O(n · σ)
3.      u := ExtractMin(Q)
4.      for each vertex v adjacent to u do                 O(1)
5.          Relax(u, v, Iwt)                               O(σ)
```

Fig. 5. *The parallel Dijkstra algorithm.*

in constant time. Also note that an edge (u, v) may be Relaxed several times ($O(\sigma)$ times), each time with respect to a different cluster center.

Thus the time and space complexity of the algorithm is affected by the amount of overlap of the clusters in the cluster cover. The space complexity of the data structure is $O(n \cdot \sigma + R)$. Furthermore, the Relax operation has a running time of $O(\sigma)$, and the total number ($O(n \cdot \sigma)$) of ExtractMin operations can be performed in total time $O(n \cdot \sigma + R)$. A careful implementation of cluster cover can be made to run in time $O(n \cdot \sigma^2 + R)$. The value of R in our applications will be $O(n)$; hence the time complexity will be $O(n \cdot \sigma^2)$.

3.2.2. Computing the cluster graph. Now we are ready to describe how to compute the cluster graph. The input is a weighted graph G with a weight function w, a set of cluster centers $C = \{v_1, \ldots, v_m\}$, and two different radii r and R, where $R > r$. In order to compute the cluster graph, the algorithm computes a cluster cover from the same set of cluster centers but with the two radii r and R. Let the cluster covers with radii, r and R be denoted by K_r and K_R, respectively. We augment the cluster cover procedure to also produce a data structure that supports the following queries for both the cluster covers.

- FindCenters(v, K): *Given $v \in V$, it returns all cluster centers v_i such that v is in a cluster from K centered at v_i; i.e., $D_{\{G,w\}}(v, v_i)$ is at most the radius of the clusters in K. It also returns $D_{\{G,w\}}(v, v_i)$ for these cluster centers.*
- ComputeDistance(v_i, v): *Given $v \in V$ and a cluster center v_i, it returns the quantity $D_{\{G,w\}}(v, v_i)$ if $D_{\{G,w\}}(v, v_i) \leq R$; otherwise, it returns the value ∞.*

Now the cluster graph $H = (V, E_H)$ is computed easily as follows. The intracluster edges of H are computed by performing FindCenters queries for each vertex $v \in V$ in the cluster cover K_r and adding the corresponding edges. Recall that FindCenters returns a set T of 2-tuples, where each tuple t consists of a vertex $t.u$ and its distance $t.d$ to v. The algorithm is described in pseudocode in Figure 6 and has a running time of $O(n \cdot \sigma)$.

From Definition 7, we have that each intercluster edge can be one of two types—either type 1 or type 2. The type 2 edges are only added after the initial construction

Algorithm IntraEdges

```
1. for every vertex v ∈ V do
2.     T := FindCenters(v, K_r)
3.     for every element t ∈ T do
4.         AddEdge(H, v, t.u, t.d); Comment: add edge (v, t.u) of weight t.d to H
```

Fig. 6. *Algorithm to add intracluster edges.*

```
Algorithm InterEdgesType1
1.  for every cluster center v in K_r do
2.      T := FindCenters(v, K_R)
3.      for every element t ∈ T do
4.          if (ComputeDistance(v, t.u) ≥ r) then
5.              AddEdge(H, v, t.u, t.d)
```

FIG. 7. *Algorithm to add intercluster edges of type* 1.

to maintain the cluster graph and will be considered in the next paragraph. An edge $[v_i, v_j]$ of type 1 is added if $v_i \notin K_j$, $v_j \notin K_i$, and $D_{\{G,w\}}(v_i, v_j) \leq R$, where $K_i, K_j \in K_r$ are clusters with centers at v_i and v_j. For every cluster center v_i, we use the FindCenters query to list all the clusters from K_R that it is contained in. The centers v_j of these clusters satisfy the condition that $D_{\{G,w\}}(v_i, v_j) \leq R$. Now we use the ComputeDistance queries to make sure that $v_i \notin K_j$ and $v_j \notin K_i$. Adding the intercluster edges of type 1 is done in time $O(n \cdot \sigma^2)$, as shown in the algorithm in Figure 7.

The time complexity of computing the cluster graph is $O(n \cdot \sigma^2)$. Having the cluster centers selected before performing the clustering enables clusters to be grown in "parallel," and thus the above algorithm is able to use one common priority queue to grow all the clusters and is consequently able to perform the clustering efficiently.

3.2.3. Maintaining the cluster graph during a phase. An edge $[v_i, v_j]$ of type 2 is added if there exists an edge $e = (u_i, u_j) \in E$ such that $u_i \in K_i$ and $u_j \in K_j$. During the computation of the cluster graph H at the start of a phase, only intracluster edges and intercluster edges of type 1 are added. Additional edges may be added during a phase of the greedy algorithm. Every time the greedy algorithm decides to add an edge $e = (u, v)$ to the partial spanner graph, several intercluster edges of type 2 may be added to H. This is achieved as follows: for every edge $e = (u_i, u_j)$ that is to be added to G', perform FindCenters queries for u_i and u_j from K_r, and join the corresponding cluster centers by intercluster edges in H. The weight of such edges is computed by performing two ComputeDistance queries for u_i and u_j with the corresponding cluster centers and adding it to the weight of (u_i, u_j). Note that this gives a safe upper bound on the true distance between u_i and u_j. The ratio of R to r determines the maximum error represented by the bound. It is clear that the function shown in Figure 8 runs in $O(\sigma^2)$ time, and it is performed $O(n)$ times.

3.2.4. Selecting the cluster centers for a phase. In order for the ClusterGraph function to be implemented efficiently, it needs to have the set of cluster

```
Algorithm InterEdgesType2(u_i, u_j)
1.  T_1 := FindCenters(u_i, K_r)
2.  T_2 := FindCenters(u_j, K_r)
3.  for every t_1 ∈ T_1 do
4.      for every t_2 ∈ T_2 do
5.          AddEdge(H, t_1.u, t_2.u, t_1.d + w(u_i, u_j) + t_2.d)
```

FIG. 8. *Algorithm to add intercluster edges of type* 2.

centers as input. For the first phase, the cluster centers C_1 are identified in a greedy fashion using the weighted graph $G' = (V, E')$ with real-valued edge weights, and using a radius of $r = \delta W_1$. That is, select any point $v \in V$ not belonging to any clusters already computed, as a cluster center, and compute the cluster with center at v. Continue this procedure until all points in V belong to a cluster. This is referred to as NAIVE-CENTERS in the algorithm given in Figure 2. NAIVE-CENTERS runs in $O(\sigma n \log n)$ time, since this can be implemented using the standard Dijkstra's algorithm.

For subsequent phases, cluster centers are identified (using UPDATECENTERS) in a different way. As a first approximation, the set of cluster centers is always chosen as a subset of the cluster centers used in the previous phase. It may turn out that this choice of cluster centers does not give us a cluster cover. Later we describe how to augment the set of cluster centers to ensure that a cluster cover is obtained.

At the end of each phase, the algorithm selects a set of cluster centers for the next phase. These centers are guaranteed to be sufficiently far apart from each other. More specifically, the cluster centers C_i used in phase i are guaranteed to be at a distance of at least $r/2$.

At the end of phase i, the set of cluster centers for phase $i + 1$ is computed. Initially we set $C_{i+1} := C_i \backslash M_i$; i.e., a subset M_i of the cluster centers is deleted from the list of cluster centers. Later C_{i+1} is augmented appropriately. We now describe how the set M_i is chosen. M_1 is the empty set, implying that C_2 is identical to C_1. For $i > 1$, the algorithm iteratively picks a cluster center from C_i and marks all cluster centers that are within distance r from it. The cluster centers that are marked are inserted into M_i and hence deleted in the next phase. We will refer to this process as a "thinning" of centers in C_i. This is easily implemented by calling the FINDCENTERS after the cluster cover for phase i has been computed. The next cluster center is then picked, and the process continues until all centers have been processed. Now set $C_{i+1} := C_i \backslash M_i$. Clearly this process runs in time $O(m \cdot \sigma)$. It is important to note that, since the integralization changes in every iteration, vertices that are at distance r' in one iteration are at distance at least $r'/2$ in the next iteration. Since the integers are rounded up, it is possible that clustering from centers C_{i+1} with radius r in phase $i + 1$ may not give us a cluster cover.

Steps 1 through 3 (described below) are repeated until a clustering from cluster centers C_{i+1} with radius r gives us a cluster cover in phase $i + 1$.

- *Step* 1. Run PARALLELDIJKSTRA from all vertices of C_{i+1} using the integralization of phase $i+1$. Let K_i be the set of vertices not covered by the clusters centered at C_{i+1}.
- *Step* 2. Greedily pick a subset $K' \subseteq K_i$ such that no two vertices in K' belong to the same cluster from the cluster cover of phase i.
- *Step* 3. If K_i is empty, then **Stop**. Else let K be the result of "thinning" out of the centers in K', and set $C_{i+1} := C_{i+1} \cup K$.

It is easy to see that, after the above process, the new centers are guaranteed to remain at a distance of at least $r/2$. Second, because the process continues until all vertices are covered, it produces a set of cluster centers that will produce a cluster cover in the next phase. Finally, we argue that the number of times steps 1 through 3 are executed is $O(1)$ per phase. Let $\mathcal{C}$ be a cluster from the cluster cover of phase i. Let $\mathcal{P}$ be the region composed of the circular discs of radius r centered at vertices of $\mathcal{C}$. In each iteration of steps 1 through 3, at least one vertex of $\mathcal{C}$ is picked for K_i. This vertex is either chosen for K, or else a large constant fraction of its area

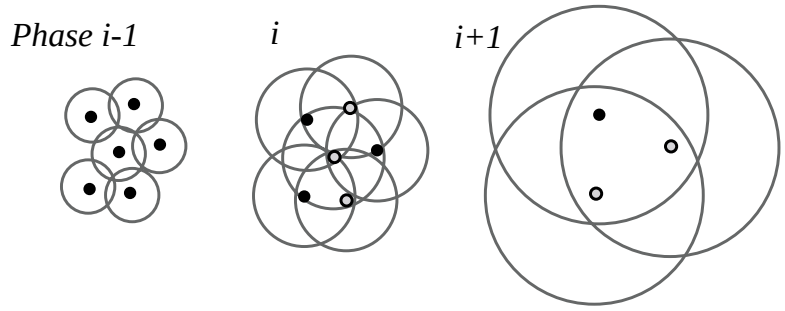


FIG. 9. *If a cluster contains another cluster center, then this cluster center is marked for deletion in the next phase. The figure shows an example of a set of cluster centers, where the cluster centers that are marked for deletion for the next phase are marked in grey.*

is covered by some circular discs of radius r centered at vertices of K. Clearly the region $\mathcal{P}$ will be covered by at most a constant number (exponentially proportional to the dimension of the space d) of circular discs of radius r. Thus steps 1 through 3 will only be executed a constant number of times, each of which takes $O(n)$ time.

We now show that, in phase i, the cluster centers are guaranteed to be at a distance of at least $r/2$ from each other. In phase 1, since cluster centers are identified by using a radius of r, all cluster centers are at a distance of at least r from each other. In phase $i-1$, if two cluster centers are at a distance of r or less, then one of them will get marked and will subsequently be deleted from the list C_i for phase i, as shown in Figure 9. Lemma 8 specifies conditions under which vertices belong to at most a constant number of clusters.

It follows from this lemma that no vertex of H is in more than a constant number of clusters of radius r or of radius R (since $\frac{R}{r} = \frac{1}{\delta}$).

LEMMA 8. *Let $C = \{v_1, \ldots, v_m\} \subseteq V$ be a set of vertices such that, for any pair of vertices $v_i, v_j \in C$, $D_{\{G,w\}}(v_i, v_j) > r'$. If $K = \{K_1, \ldots, K_m\}$ is returned by* CLUSTER-COVER$(G, C, c' \cdot r')$*, where c' is a constant, then each vertex $v \in V$ is contained in at most a constant (which depends on the dimension d and c') number of clusters from K.*

The conditions of the lemma are true for the cluster graph as constructed above with $r' = r/2$ and $c' = 2$ or $c' = 4t/\delta$. Hence any vertex in H is part of at most a constant number of clusters in K_r or K_R. The proof follows from standard packing arguments; see also section 2.2. Similar arguments also show that the number of intercluster edges incident to a cluster center is also a constant (although it might have a large number of intracluster edges). It follows that the degree of any vertex in H that is not a cluster center must be a constant, and the size of H is $O(n)$. Thus σ is also bounded by a constant. Note that IMPROVED-GREEDY uses integralized weights so the resulting clusters are approximate clusters; they are a little bit larger (since integers are always rounded up) than the exact clusters. It is clear that this does not affect the correctness of Lemma 8.

3.3. Answering shortest path queries. When the algorithm IMPROVED-GREEDY considers an edge $e = (u, v)$ for inclusion in the spanner graph, it needs to answer a shortest path query. It needs to check if $D_{\{G',wt\}}(u, v) \leq t \cdot d(u, v)$, where G' is the spanner graph constructed so far. As noted in [8], it is sufficient for this query to be answered approximately. So it is sufficient to devise a procedure to

efficiently check if $D_{\{G',wt\}}(u,v) \leq t(1+\epsilon') \cdot d(u,v)$ for some small $\epsilon' > 0$. In other words, it is sufficient to check if $D_{\{G',Iwt\}}(u,v) \leq t(1+\epsilon'') \cdot d(u,v)/U_i$ for some small $\epsilon'' > 0$. In fact, the algorithm will check if $D_{\{H,Iwt\}}(u,v) \leq \sqrt{tt'} \cdot d(u,v)/U_i$. The time complexity of this test is constant if $D_{\{H,Iwt\}}(u,v) < c' \cdot r$ for some constant c'. Hence we conclude this section with the following theorem, which follows from the above arguments.

THEOREM 9. IMPROVED-GREEDY *runs in time* $O(n \log n)$.

Proof. Following the steps of the IMPROVED-GREEDY algorithm shown in Figure 2, we have that step 1 in the initialization and steps 8 and 11 take time $O(n \log n)$; step 2 takes constant time; step 6 runs in $O(\log n)$ time, while steps 3 and 10 take linear time. The integralization of the weight of the edges in steps 11, 14, and 22 takes a total of $O(\log n)$ time per edge. Since each edge is considered exactly once, the total time spent on integralizing and reintegralizing the weight of the edges is $O(n \log n)$ according to section 3.1. Step 15 requires linear time since σ is bounded by a constant. On line 16, every edge in the input spanner is considered once. For each edge, the algorithm performs one shortest path query in the cluster graph. As mentioned above, each query takes constant time. Hence the total time complexity for computing a linear number of shortest path queries is $O(n)$. Finally, updating the centers is easily done in linear time. From this it follows that IMPROVED-GREEDY runs in time $O(n \log n)$. ∎

In 1999, Thorup [16] showed that single-source shortest path queries could be answered in linear time for undirected graphs with integer edge weights. However, this algorithm was not used in this paper since it does not visit the vertices in order of increasing distance, which is crucial for our algorithm. Also, it uses bit-shift for computing the floor function in constant time, which is not allowed in the computational model used in our algorithm.

4. The graph produced by IMPROVED-GREEDY is a *t*-spanner. In order to show that the produced spanner graph G' is a t-spanner, we need two main results. First, we need to show that the cluster graph H approximates the spanner graph G'; i.e., $D_{\{G',wt\}}(v,u) \leq D_{\{H,Iwt\}}(v,u) \cdot U_i \leq \alpha D_{\{G',wt\}}(v,u)$ for some constant α close to 1. This is done in Lemmas 12 and 13. Second, we need to show, in Lemma 14, that H is always a valid cluster graph of G'. From these results, we easily obtain Theorem 15, which says that the produced spanner G' is a t-spanner of the complete Euclidean graph.

Since the clusters are computed using the function $Iwt(\cdot)$ instead of $wt(\cdot)$, clusters are not as precise as they were in [8]. In this section, we will assume that the smaller radii r_i is δW_i and the larger radii R_i is W_i, where δ is a positive constant decided in Lemmas 14 and 17. Finally, we set $\epsilon = \frac{nU_i}{\delta W_i}$. Some of the results in this section and the next are modified versions of analogous results in [8].

LEMMA 10. *Let* K *be equal to* CLUSTER$(G', v, \delta W_i, Iwt_i)$*, i.e., a cluster with cluster center* v *and radius* δW_i *computed in iteration* i *of the algorithm. If* u *is a vertex in* K*, then* $D_{\{G',wt\}}(v,u) \leq (1+\epsilon)\delta W_i U_i$*. Otherwise, if* $u \notin K$*, then* $D_{\{G',wt\}}(v,u) > \delta W_i U_i$*.*

Proof. The lemma follows from Corollary 4 and the fact that a cluster from K with center at v consists of all vertices within integer distance $\delta W_i/U_i$ from v. ∎

Consider the cluster graph H that results from the clustering performed on G' at the start of phase i. The following results apply to edges and paths in H.

LEMMA 11. *If u is a cluster center and $[u, v]$ is an intracluster edge in H, then*

$$D_{\{G',wt\}}(u, v) \leq (1 + \epsilon)\delta W_i U_i. \tag{1}$$

If $[v_j, v_k]$ is an intercluster edge in H, then

$$\delta W_i U_i < D_{\{G',wt\}}(u, v) \leq (1 + \epsilon) \cdot (W_i + 2\delta W_i)U_i. \tag{2}$$

Proof. The first statement is a direct consequence of Lemma 10; the same holds for the left inequality in (2). The right inequality in the second statement follows from Definition 7 since an intercluster edge of type 2 may be constructed to connect two cluster centers v_j and v_k since there exists an edge $(x, y) \in G'$ such that $x \in K_j$, $y \in K_k$, and $Iwt(x, y) < W_i/U_i$. □

For simplicity, in the rest of this section, we will leave out the unit length U_i. The following lemma is straightforward since H is an approximation of G'.

LEMMA 12. *If there exists a path P_H in H between vertices u and v such that $Iwt(P_H) = L$, then there exists a path $P_{G'}$ in G' between vertices u and v such that $Iwt(P_{G'}) \leq L$.*

We first introduce some definitions. A vertex u is defined to be *sufficiently far* from a vertex v if (1) no single cluster contains both u and v and (2) $D_{\{G',wt\}} \geq W_i$. Define a *cluster path* in H to be a path where the first and last edges may be intracluster edges but all intermediate edges are intercluster edges.

The next lemma is the approximate converse of Lemma 12.

LEMMA 13. *Let u be sufficiently far from v. Let $P_{G'}$ be a path between u and v in G' such that $wt(P_{G'}) = L_1$. Then there exists a cluster path P_H between u and v in H such that*

$$Iwt(P_H) = L_2 < L_1 \cdot \frac{(1 + \epsilon)(1 + 6\delta)}{1 - 2\delta(1 + \epsilon)}.$$

Proof. The proof is similar to the proof of Lemma 4 in [8]. Let the path from u to v having weight L_1 in G' be P. We shall use the notation $P(y, x)$ to denote the vertices of P between vertices y and x, not including y. We construct a cluster path Q from u to v in H with weight L_2 as follows. Let C_0 be any cluster, with center v_0, containing u. The first edge of Q is the intracluster edge $[u, v_0]$. Next, among all clusters with centers adjacent to v_0 in H, let C_1, with center v_1, intersect the furthest vertex along $P(u, v)$, say, w_1. Add the intercluster edge $[v_0, v_1]$ to Q. Next, among all clusters with centers adjacent to v_1 in H, let C_2, with center v_2, intersect the furthest vertex along $P(w_1, v)$, say, w_2. Add the intercluster edge $[v_1, v_2]$ to Q. This process continues until we reach a cluster center, v_m, whose cluster contains v. At this stage, complete Q by adding the intracluster edge $[v_m, v]$, as shown in Figure 10. Three cases arise, and the lemma is proved in each of these cases.

Case 1 ($m = 1$). In this case, there is only one intercluster edge along Q. Since u is sufficiently far from v, we know that $L_1 > W_i - 2(1+\epsilon)\delta W_i$. Now $L_2 = Iwt([u, v_0]) + Iwt([v_0, v_1]) + Iwt([v_1, v])$. However, $Iwt([u, v_0]) \leq (1 + \epsilon)\delta W_i$ and $Iwt([v, v_1]) \leq (1+\epsilon)\delta W_i$, while $Iwt([v_0, v_1]) \leq 2(1+\epsilon)\delta W_i + D_{\{G',Iwt\}}(v, u) \leq 2(1+\epsilon)\delta W_i + (1+\epsilon)L_1$. This result follows from the procedure ADDINTEREDGESTYPE2, since $wt([v_0, v_1])$ is at most $2(1 + \epsilon)\delta W_i$ plus the length of the shortest edge connecting vertices of the two clusters to which u and v belong. So $L_2 \leq (1 + \epsilon)L_1 + 4(1 + \epsilon)\delta W_i$, and we have that $W_i < \frac{(1+\epsilon)L_1}{1-2(1+\epsilon)\delta}$. Combining these inequalities, we get

$$L_2 \leq (1 + \epsilon)L_1 + \frac{4(1 + \epsilon)\delta}{1 - 2(1 + \epsilon)\delta} \cdot L_1 < \frac{(1 + \epsilon)(1 + 2\delta)}{1 - 2\delta(1 + \epsilon)} \cdot L_1.$$

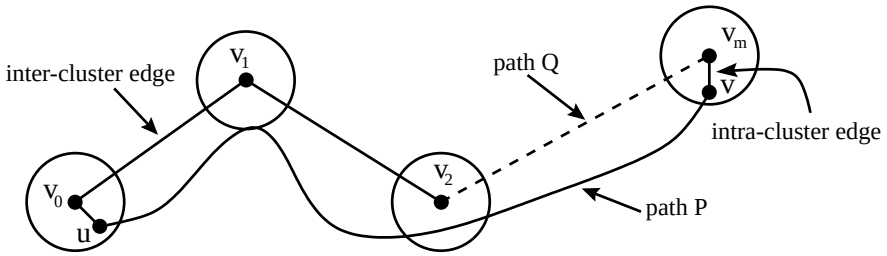


FIG. 10. *Paths in H approximate paths in G'.*

Case 2 ($m \geq 2$ and m even). Suppose $[v_i, v_{i+1}]$ and $[v_{i+1}, v_{i+2}]$ are any two consecutive intercluster edges on Q. Observe that the sum of their weights is greater than W_i. If this were not so, then the edge $[v_i, v_{i+2}]$ would instead have been added to Q while Q was being constructed. Divide Q into portions $Q_0, Q_1, \ldots$, where Q_{2i} is the portion between v_{2i} and v_{2i+2}. Similarly, divide P into portions $P_0, P_1, \ldots$, where P_{2i} is the portion between the last vertex intersecting C_{2i} and the first vertex intersecting C_{2i+2}. We shall first prove that, for any even i, the weight of Q_{2i} is no more than a constant times the weight of P_{2i}.

Let the weight of P_{2i} be p_{2i} and that of Q_{2i} be q_{2i}. Since there cannot be an intercluster edge between v_{2i} and v_{2i+2}, we have that $p_{2i} > W_i - 2\delta(1+\epsilon)W_i$. Thus $W_i < \frac{p_{2i}}{1-2(1+\epsilon)\delta}$. Select r to be any vertex of P_{2i} within the intermediate cluster C_{2i+1}. The vertex r splits P_{2i} into two portions. Let p'_{2i} (respectively, p''_{2i}) be the initial (respectively, final) portions; thus $p_{2i} = p'_{2i} + p''_{2i}$. From the procedure InterEdgesType2, we have $Iwt([v_{2i}, v_{2i+1}]) \leq (1+\epsilon)p'_{2i} + 2\delta(1+\epsilon)W_i$, and, similarly, $Iwt([v_{2i+1}, v_{2i+2}]) \leq (1+\epsilon)p''_{2i} + 2\delta(1+\epsilon)W_i$. Adding the two, we get $q_{2i} \leq (1+\epsilon)p_{2i} + 4\delta(1+\epsilon)W_i$. We now have two inequalities relating p_{2i}, q_{2i} and W_i. Thus

$$q_{2i} < (1+\epsilon)p_{2i} + 4\delta(1+\epsilon) \cdot \frac{p_{2i}}{1-2\delta(1+\epsilon)} < p_{2i} \cdot \frac{(1+\epsilon)(1+2\delta)}{1-2\delta(1+\epsilon)}.$$

Summing over all even values of i and taking into account the two intracluster edges at either end of Q, we get

$$L_2 < L_1 \cdot \frac{(1+\epsilon)(1+2\delta)}{1-2\delta(1+\epsilon)} + 2\delta(1+\epsilon)W_i.$$

Since u is sufficiently far from v, we know that $L_1 > W_i - 2\delta(1+\epsilon)W_i$. That is, $\frac{L_1}{1-2\delta(1+\epsilon)} > W_i$. Substituting this in the above inequality, we obtain

$$L_2 < L_1 \cdot \frac{(1+\epsilon)(1+4\delta)}{1-2\delta(1+\epsilon)}.$$

Case 3 ($m \geq 3$ and m odd). The analysis will be exactly the same as in the previous case, except that we have to account for the last intercluster edge along Q and, correspondingly, the portion of P between the last two clusters. Let q_{m-1} be the integer weight of $[v_{m-1}, v_m]$, and let p_{m-1} be the weight of the portion of P between the last vertex intersecting C_{m-1} and the first vertex intersecting C_m. Clearly $q_{m-1} \leq (1+\epsilon)p_{m-1} + 2\delta(1+\epsilon)W_i$. This inequality can be rewritten as $q_{m-1} < (\frac{(1+\epsilon)(1+2\delta)}{1-2\delta(1+\epsilon)})p_{m-1} + 2\delta(1+\epsilon)W_i$. We then sum up as above to get

$$L_2 < L_1 \cdot \frac{(1+\epsilon)(1+2\delta)}{1-2\delta(1+\epsilon)} + 4\delta(1+\epsilon)W_i.$$

Since $L_1 > W_i - 2\delta(1+\epsilon)W_i$, we have that $L_1 \cdot \frac{4\delta(1+\epsilon)}{1-2\delta(1+\epsilon)} > 4\delta(1+\epsilon)W_i$. Substituting this in the above inequality, we obtain

$$L_2 < L_1 . \frac{(1+\epsilon)(1+6\delta)}{1-2\delta(1+\epsilon)}.$$

That completes the proof of the lemma. □

Before processing group E_i (which contains edges with weights in the range $(W_i, 2W_i]$), the algorithm constructs a fresh cluster graph H using a radius of δW_i.

LEMMA 14. *During the processing of any group E_i, the graph H always represents a valid cluster graph of G'.*

Proof. Let the edges in E_i be ordered by increasing weight as $e_{i1}, \ldots, e_{il}$. The proof is by induction. In the base case, when none of the edges have been processed, the lemma is obviously true. Now assume that the lemma is true just before the algorithm decides to examine edge e_{ij}=(u, v). If this edge is not added to G', then the lemma still holds. Now suppose this edge is added to G'. Since $wt(u, v) > W_i$ and $\delta < 1/2$, the distance between any two previous cluster centers in the new G' will remain greater than δW_i, and thus the previous cluster cover will remain valid. Also, the previous intracluster edges and the intercluster edges of type 1 (see the definition of intercluster edges) will remain the same. We have only to make sure that we add new intercluster edges of type 2, and it is easily seen that this is done by the algorithm. It remains to decide what weights are to be assigned to these new intercluster edges in H. Consider one such edge $[x, y]$, where x (respectively, y) is the center of the cluster to which u (respectively, v) belongs. The weight of this edge should be assigned $D_{\{G', Iwt\}}(x, y)$ (the shortest path in the new graph G' between x and y). However, it will be too time consuming to compute this directly. Instead the algorithm assigns the weight as $Iwt([x, u]) + Iwt(u, v) + Iwt([y, v])$ (see section 3.2.3). We now show that our choice of δ makes this acceptable.

Assume the contrary, i.e., that a shorter alternate path P exists between x and y. Let $Iwt(P)$ denote its integral weight in this phase. Since P cannot involve the edge (u, v), it contains only edges of the previous G'. However, we know that (u, v) was selected to be added to G'; thus no cluster path existed between u and v of weight within $\sqrt{tt'} \cdot Iwt(u, v)$ in H. Furthermore, since u is sufficiently far from v, we may use Lemma 13 to get

$$Iwt(P) + 2\delta W_i(1+\epsilon) > \frac{1-2\delta(1+\epsilon)}{(1+\epsilon)(1+6\delta)} \cdot \sqrt{tt'} \cdot Iwt(u, v).$$

We have that $Iwt(P) < Iwt([x, u]) + Iwt(u, v) + Iwt([v, y])$, which is at most $Iwt(u, v) + 2\delta W_i(1+\epsilon)$. Putting these two results together, we obtain

$$4\delta(1+\epsilon)W_i + Iwt(u, v) > \frac{1-2\delta(1+\epsilon)}{(1+\epsilon)(1+6\delta)} \cdot \sqrt{tt'} \cdot Iwt(u, v).$$

Using the fact that $Iwt(u, v) > W_i$, we get

$$4\delta(1+\epsilon) > \frac{1-2\delta(1+\epsilon)}{(1+\epsilon)(1+6\delta)} \cdot \sqrt{tt'} - 1.$$

If we solve this inequality for δ, we see that the only positive solutions are

$$\delta > \frac{\sqrt{tt'} - (1+\epsilon)}{2(\sqrt{tt'}(1+\epsilon) + 5 + 7\epsilon + 2\epsilon^2)}.$$

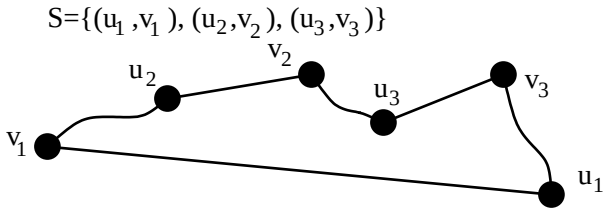


FIG. 11. *Illustration of the definition of the leapfrog property.*

According to our choice of δ in the algorithm IMPROVE-GREEDY, the above inequality will never be satisfied (see Figure 2). Hence we have a contradiction. □

The following theorem now concludes this section.

THEOREM 15. *The graph produced by* IMPROVED-GREEDY *is a t-spanner of the complete Euclidean graph.*

Proof. The proof follows from the fact that G' is a $\sqrt{tt'}$ spanner of G and that G is a $\sqrt{t/t'}$ spanner of the complete graph. □

5. The weight of G' is $O(wt(MST))$. In [4], it was shown that the greedy algorithm produces a spanner that has $O(n)$ edges and a total weight of $O(\log n) \cdot wt(MST(V))$, where $MST(V)$ is the minimum spanning tree of V. The analysis of the greedy algorithm was then improved in [7]. The proof relies on a property known as the *leapfrog property*. This property restricts how a set of line segments may be positioned in space. Here we provide a definition which is technical and nonintuitive.

Let $t \geq t' > 1$. A set of line segments, denoted E', in d-dimensional space satisfy the (t', t)-*leapfrog* property if the following is true for every possible $S = \{(u_1, v_1), \ldots, (u_m, v_m)\}$, which is a subset of E':

$$t' \cdot wt(u_1, v_1) < \sum_{i=2}^{m} wt(u_i, v_i) + t \cdot \left(\sum_{i=1}^{m-1} wt(v_i, u_{i+1}) + wt(v_m, u_1) \right).$$

Informally, this definition says that, if there exists an edge between u_1 and v_1, then any path, not including (u_1, v_1), must have length greater than $t' \cdot wt(u_1, v_1)$, as shown in Figure 11. The following fact was shown by Das and Narasimhan [8].

FACT 16 (Theorem 3 in [8]). *There exists a constant $0 < \phi < 1$ such that the following holds: if a set of line segments E' in d-dimensional space satisfies the (t', t)-leapfrog property, where $t \geq t' \geq \phi t + 1 - \phi > 1$, then $wt(E') = O(wt(MST))$, where MST is a minimum spanning tree connecting the endpoints of E'. The constant implicit in the O-notation depends on t and d.*

Now suppose that we construct a t-spanner such that, for every spanner edge (u, v), the second shortest path is not necessarily longer than $t \cdot wt(u, v)$ but longer than $t' \cdot wt(u, v)$ for some t' such that $t \geq t' > 1$. In this case, the t-spanner satisfies the (t, t')-leapfrog property, as can be proved by using arguments similar to those used in Lemma 2.4 in [7]. Hence the produced spanner will then have total weight $O(wt(MST(V)))$.

So it remains to prove that the weight of the second shortest path between u and v is greater than $t' \cdot wt(u, v)$. First, note that the edges in E_0 do not contribute much because their total length is at most equal to the length of the longest edge ($< n \cdot D/n$), which is less than the weight of the minimum spanning tree. We estimate $wt(E' \backslash E_0)$, where E' is the set of edges produced by the algorithm.

LEMMA 17. *Let $e=(u,v) \in E' \backslash E_0$. The weight of the second shortest path between u and v is greater than $t' \cdot wt(u,v)$.*

Proof. Let C be the shortest simple cycle in G' containing e. We have to estimate $wt(C) - wt(u,v)$. Let $e_1 = (u_1, v_1)$ be the longest edge on the cycle. Then $e_1 \in E' \backslash E_0$, and, among the cycle edges, it is examined last by the algorithm. What happens while the algorithm is examining e_1?

Assume that e_1 is examined in phase i. There is an alternate path in G' from u_1 to v_1 of weight $wt(C) - wt(u_1, v_1)$. However, since the algorithm eventually decides to add e_1 to the spanner, at that moment, the weight of each cluster path from u_1 to v_1 is larger than $\sqrt{tt'} \cdot Iwt(u_1, v_1) \cdot U_i$. Notice that $Iwt(u_1, v_1) \cdot U_i$ and $wt(u_1, v_1)$ are larger than W_i. This implies that u_1 and v_1 are not contained in the same cluster. Thus u_1 is sufficiently far from v_1. Lemma 13 implies that the weight of each path in G' between u_1 and v_1 is large; i.e.,

$$wt(C) - wt(u_1, v_1) > \sqrt{tt'} \cdot Iwt(u_1, v_1) \cdot U_i > \sqrt{tt'} \cdot wt(u_1, v_1) \cdot \frac{1 - 2\delta(1+\epsilon)}{(1+\epsilon)(1+6\delta)}.$$

However, we know, according to the algorithm, that $\delta \leq \frac{\sqrt{tt'} - (1+\epsilon)t'}{2(1+\epsilon)(\sqrt{tt'}+3t')}$. Substituting, we obtain $wt(C) - wt(u_1, v_1) > t' \cdot wt(u_1, v_1)$. ∎

Finally, since Lemma 17 holds, we can use the following observation which, together with Fact 16, concludes the proof of Theorem 1.

OBSERVATION 18. *$E' \backslash E_0$ satisfies the (t', t)-leapfrog property.*

Proof. Consider any subset of the edges $S = \{(u_1, v_1), \ldots, (u_m, v_m)\}$ of E'. By Lemma 17, we know that $t' \cdot d(u_1, v_1)$ is smaller than the weight of the second shortest path between u_1 and v_1 in G'. Consider a path P from v_1 to u_1, composed of the shortest path from v_1 to u_2 (of weight $\leq t \cdot d(v_1, u_2)$), the edge (u_2, v_2), the shortest path from v_2 to u_3 (of weight $\leq t \cdot d(v_2, u_3)$), and so on, until the final portion is the shortest path from v_m to u_1. Clearly $wt(P)$ is at least as large as the weight of the second shortest path between u_1 and v_1. However, $wt(P)$ is also equal to the right-hand side of the definition of the leapfrog property. The observation follows. ∎

This concludes the proof of Theorem 1.

6. Conclusions and open problems. This paper represents an important advancement in the study of spanners; we present the first correct $O(n \log n)$-time algorithm to construct low-weight spanners (weight $O(1) \cdot wt(MST)$) and a small number of edges (only $O(n)$ edges). The implementation of clustering techniques is of independent interest in the design of efficient algorithms.

The main theoretical open problem that remains unsolved is to design an algorithm to construct a sparse t-spanner in time $O(n \log n)$ in the algebraic decision tree model of computation.

Acknowledgments. We are grateful to Michiel Smid for numerous helpful discussions. We are also very grateful to a diligent referee for many good comments and for helping us fix a subtle error in section 3.2.4.

REFERENCES

[1] I. ALTHÖFER, G. DAS, D. P. DOBKIN, D. JOSEPH, AND J. SOARES, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid, *Euclidean spanners: Short, thin, and lanky*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 489–498.

[3] P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.

[4] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, Internat. J, Comput. Geom. Appl., 5 (1995), pp. 124–144.

[5] D. Z. Chen, G. Das, and M. Smid, *Lower bounds for computing geometric spanners and approximate shortest paths*, in Proceedings of the 8th Annual Canadian Conference on Computational Geometry, Ottawa, Canada, 1996, pp. 155–160.

[6] A. Czumaj and A. Lingas, *Linear-time heuristics for minimum weight rectangulation*, in Proceedings of the 27th Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1853, Springer-Verlag, New York, 2000, pp. 856–868.

[7] G. Das, P. Heffernan, and G. Narasimhan, *Optimally sparse spanners in 3-dimensional Euclidean space*, in Proceedings of the 9th Annual ACM Symposium on Computational Geometry, ACM, New York, 1993, pp. 53–62.

[8] G. Das and G. Narasimhan, *A fast algorithm for constructing sparse Euclidean spanners*, Internat. J. Comput. Geom. Appl., 7 (1997), pp. 297–315.

[9] J. M. Keil and C. A. Gutwin, *Classes of graphs which approximate the complete Euclidean graph*, Discrete Comput. Geom., 7 (1992), pp. 13–28.

[10] C. Levcopoulos and A. Lingas, *There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees*, Algorithmica, 8 (1992), pp. 251–256.

[11] C. Levcopoulos, G. Narasimhan, and M. Smid, *Efficient algorithms for constructing fault-tolerant geometric spanners*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 186–195.

[12] D. Mount, Private communication, Department of Computer Science, University of Maryland, College Park, MD, 1998.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[14] S. B. Rao and W. D. Smith, *Improved approximation schemes for traveling salesman tours*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 540–550.

[15] J. S. Salowe, *Construction of multidimensional spanner graphs with applications to minimum spanning trees*, in Proceedings of the 7th Annual ACM Symposium on Computational Geometry, ACM, New York, 1991, pp. 256–261.

[16] M. Thorup, *Undirected single-source shortest path with positive integer weights in linear time*, J. ACM, 46 (1999), pp. 362–394.

[17] P. M. Vaidya, *A sparse graph almost as good as the complete graph on points in K dimensions*, Discrete Comput. Geom., 6 (1991), pp. 369–381.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1501–1526

GRAPH NONISOMORPHISM HAS SUBEXPONENTIAL SIZE PROOFS UNLESS THE POLYNOMIAL-TIME HIERARCHY COLLAPSES*

ADAM R. KLIVANS† AND DIETER VAN MELKEBEEK‡

Abstract. Traditional hardness versus randomness results focus on time-efficient randomized decision procedures. We generalize these trade-offs to a much wider class of randomized processes. We work out various applications, most notably to derandomizing Arthur-Merlin games. We show that every language with a bounded round Arthur-Merlin game has subexponential size membership proofs for infinitely many input lengths unless exponential time coincides with the third level of the polynomial-time hierarchy (and hence the polynomial-time hierarchy collapses). Since the graph nonisomorphism problem has a bounded round Arthur-Merlin game, this provides the first strong evidence that graph nonisomorphism has subexponential size proofs.

We also establish hardness versus randomness trade-offs for space bounded computation.

Key words. derandomization, pseudorandomness, graph isomorphism, interactive proofs, Arthur-Merlin games, hardness versus randomness, universal traversal sequences, unique satisfiability, learning theory, rigid matrices

AMS subject classification. 68Q

PII. S0097539700389652

1. Introduction.

Randomness has proven to be a valuable tool in computer science. It is an indispensable resource in areas like cryptography and distributed computing. The situation is less clear for algorithms. Several computational problems turn out to have simple randomized solutions that require less time or space than the known deterministic algorithms. It is conceivable, however, that efficient deterministic algorithms exist for all of these problems.

In fact, many complexity theorists believe that randomness *cannot* significantly reduce the time or space complexity of a problem. To this end, they try to construct efficient pseudorandom generators—deterministic procedures that stretch seeds containing a few random bits into much longer bit sequences that look random to the randomized process in question. This allows us to reduce the amount of randomness needed and, in some cases, eliminate the need for random bits completely without increasing the need for other resources by much.

Considerable progress has been made in the construction of pseudorandom generators that fool arbitrary time-bounded decision procedures. We now know how to construct them out of certain types of "hard" computational problems. The existence of such hard problems is still open. These constructions are often referred to as *hardness versus randomness trade-offs*; they rule out the possibility that both hard problems exist and randomness speeds up some time-bounded decision problem.

1.1. Traditional hardness versus randomness trade-offs.

Blum and Micali [BM84] and Yao [Yao82] were the first to realize this connection. Nisan and Wigderson

*Received by the editors November 9, 2000; accepted for publication (in revised form) February 14, 2002; published electronically August 1, 2002. A preliminary version of this paper appeared in the *Proceedings of the 31st ACM Symposium on Theory of Computing* [KvM99].

http://www.siam.org/journals/sicomp/31-5/38965.html

†Department of Mathematics, MIT, Cambridge, MA 02139 (klivans@math.mit.edu). This author's research was supported in part by NSF grant CCR-9701304.

‡Department of Computer Sciences, University of Wisconsin, Madison, WI 53706 (dieter@cs.wisc.edu). This author's research was supported in part by NSF grants CCR-9732922 and CCR-0133693.

[NW94] established a whole range of hardness versus randomness trade-offs using an average-case hardness notion. They showed how to use any language in exponential time that requires large circuits to be computed exactly on slightly more than half of the inputs, to construct a pseudorandom generator that fools circuits of polynomial size. They obtained nontrivial derandomizations of polynomial-time randomized decision algorithms under average-case hardness assumptions and even deterministic polynomial-time simulations under the strongest of their hypotheses. Babai, Fortnow, Nisan, and Wigderson [BFNW93] and Impagliazzo and Wigderson [IW97] relaxed the hardness condition from an average-case condition to a worst-case one. For their constructions, they had only to start from a language in exponential time that requires large circuits to be computed exactly on *every* input. As a corollary, they showed how to simulate every polynomial-time randomized decision algorithm deterministically in subexponential time for infinitely many input lengths unless exponential time collapses to the second level of the polynomial-time hierarchy (and, in fact, to the class MA) [BFNW93].

The authors of these papers [NW94, BFNW93, IW97] used their techniques to derandomize traditional models of randomized computation, most notably BPP. We show how to apply the techniques used to derandomize BPP to more general models of randomized computation. The key observation we make is that the reductions used in the proofs are "black-box"; i.e., they relativize. More specifically, the above papers start from a Boolean function f and construct a proposed pseudorandom generator based on it. They then argue that any small circuit that distinguishes the output of the proposed pseudorandom generator from the uniform distribution can be transformed into a small circuit that computes f. We observe that these reductions work for *any* nonuniform model of computation that satisfies certain closure properties. In particular, they work for oracle circuits given any fixed oracle B, i.e., for circuits consisting of AND, OR, and NOT gates and gates that evaluate the oracle B. Thus, in order to build a pseudorandom generator that looks random to any small B-oracle circuit, we need only assume the existence of a function f that cannot be computed by small B-oracle circuits. The same holds for more recent constructions by Sudan, Trevisan, and Vadhan [STV01] that provide alternate proofs for some of the earlier results [NW94, BFNW93, IW97].

1.2. Derandomizing Arthur-Merlin games. The above observations allow us to apply the classical hardness versus randomness results to various settings, in particular to the nondeterministic setting of Arthur-Merlin games [Bab85].

Arthur-Merlin games are verification procedures in which a prover (Merlin) tries to convince a randomized polynomial-time verifier (Arthur) of some fact, say, that a given input belongs to some language. Arthur-Merlin games define an extension of the classical proof paradigm in two ways: the use of randomness by the verifier and the interaction between the prover and the verifier. Of particular interest is the class AM of languages for which there exists an Arthur-Merlin game with a bounded number of rounds of interaction between the verifier and the prover. The class AM forms a randomized extension of NP. The most notable problem in AM not known to be in NP is graph nonisomorphism [GMW91, GS89, BM88].

Derandomizing Arthur-Merlin games requires security against the nondeterministic adversary Merlin. Rudich [Rud97] pointed out that pseudorandom generators in the traditional cryptographic setting, where an adversary has more resources than the generator, cannot hope to have this property. This is because a nondeterministic adversary can guess the seed the verifier will use and run the generator on this seed

TABLE 1
Hardness versus randomness trade-offs for AM.

Hardness assumption	Derandomization consequence
$\exists f \in \mathsf{NEXP} \cap \mathsf{coNEXP} : C_f^{\mathrm{SAT}}(n) = n^{\omega(1)}$	$\mathsf{AM} \subseteq \cap_{\epsilon>0}\mathsf{NTIME}[2^{n^\epsilon}]$
$\exists f \in \mathsf{NEXP} \cap \mathsf{coNEXP} : C_f^{\mathrm{SAT}}(n) = 2^{n^{\Omega(1)}}$	$\mathsf{AM} \subseteq \cup_{c>0}\mathsf{NTIME}[2^{(\log n)^c}]$
$\exists f \in \mathsf{NE} \cap \mathsf{coNE} : C_f^{\mathrm{SAT}}(n) = 2^{\Omega(n)}$	$\mathsf{AM} = \mathsf{NP}$

to obtain the pseudorandom sequence the verifier will use. From a derandomization perspective, however, we can limit the resources the adversary can use. Nisan and Wigderson exploited this fact in the BPP setting. We use it in the context of Arthur-Merlin games. The drawback of cryptographic pseudorandom generators does not apply because the adversaries do not have the resources to run the generator—even if they correctly guess the seed.

We give evidence that AM coincides with NP. More specifically, we show that the existence of an exponential-time decidable language with high worst-case SAT-oracle complexity implies nontrivial derandomizations of AM. Examples of the trade-offs we obtain are presented in Table 1, where we use C^B to denote circuit complexity given access to oracle B. We provide the precise definitions in section 2 and the general trade-off statement in section 3.

For example, the first line of Table 1 states that, if there exists a function computable in NEXP∩coNEXP which requires SAT-oracle circuits of superpolynomial size for almost all input lengths, then AM is contained in nondeterministic subexponential time, i.e., has subexponential size proofs. The third line of Table 1 states that, if we go further and assume that there exists a function in NE ∩ coNE which requires SAT-oracle circuits of size $2^{\epsilon n}$ for some $\epsilon > 0$ and almost all input lengths, then every language in AM has polynomial size proofs.

If the hardness condition on the left-hand side of Table 1 holds for infinitely many input lengths, then the corresponding derandomization on the right-hand side works for infinitely many input lengths. We refer to the *weak* version of Table 1 when we assume the above hardness conditions hold only for infinitely many input lengths. As in the above papers [NW94, BFNW93, IW97], we typically state our theorems assuming the hardness conditions are true for every input length. Both interpretations hold for all of our results.

We can view the assumptions in Table 1 as statements concerning the relationships among computation, nonuniformity, and nondeterminism. For example, the third entry in the table states that, if nonuniformity and nondeterminism cannot significantly speed up computation, then we can derandomize AM. We point out that, if the hardness assumption in the first row of the weak version of Table 1 fails, then exponential time collapses to the third level of the polynomial-time hierarchy.

Arvind and Köbler [AK97] obtained similar results to those in Table 1 using *nondeterministic* circuits but needed average-case hardness assumptions instead of worst-case ones. Nondeterministic circuits consist of AND, OR, and NOT gates and have some additional input bits y. An input x is accepted if there is a setting of the additional input bits y that makes the circuit accept x. Nondeterministic circuits can be efficiently simulated by SAT-oracle circuits but presumably not the other way around. Nondeterministic circuits do not seem to have the closure properties that

allow us to relax the hardness hypothesis from average-case to worst-case, whereas SAT-oracle circuits do. Bridging the gap between average-case and worst-case assumptions is crucial. It allows us to conclude that every language in AM, and graph nonisomorphism, in particular, has subexponential size proofs for infinitely many input lengths unless the polynomial-time hierarchy collapses. See the weak version of the first line in Table 1. Without any unproven hypothesis, the smallest proofs known for the nonisomorphism of two graphs on n vertices are of size $2^{O(\sqrt{n \log n})}$, namely, the transcripts of the deterministic graph isomorphism algorithm by Babai, Kantor, and Luks [BL83, BKL83].

Recently, Miltersen and Vinodchandran [MV99] managed to relax the hardness assumption in the last line of Table 1. They showed that AM collapses to NP if there exists a Boolean function in $\mathsf{NE} \cap \mathsf{coNE}$ with exponential nondeterministic circuit complexity (instead of SAT-oracle circuit complexity). Their technique works only under strong hardness assumptions. In particular, it does not apply to the rest of Table 1. See section 6 for a more detailed discussion.

1.3. Generalizing hardness versus randomness trade-offs. Our simulations of AM are a special case of a general derandomization tool which applies to any randomized process for which we can efficiently check whether a particular choice of random bits results in a successful computation. For example, for a fixed BPP algorithm and input, at least 2/3 of the random strings result in a "successful" computation. Any randomized function computation with this property, regardless of whether it involves a decision problem or not, falls within our framework. We formally define the notion of a success predicate in section 4. If we can decide the success predicate of a randomized process with polynomial size B-oracle circuits, then assuming the existence of a function with high B-oracle circuit complexity implies the existence of a pseudorandom generator suitable for derandomizing the process. Table 2 gives a summary of these trade-offs. The symbol $\mathcal{A}$ in Table 2 represents an arbitrary class of oracles and determines the complexity of computing the generator G.

TABLE 2
Overview of pseudorandom generator constructions.

Hardness assumption	Complexity of G	Seed length
$\exists f \in \mathsf{EXP}^{\mathcal{A}} : C_f^B(n) = n^{\omega(1)}$	$\mathsf{EXP}^{\mathcal{A}}$	$O(n^\epsilon)$ for arbitrary $\epsilon > 0$
$\exists f \in \mathsf{EXP}^{\mathcal{A}} : C_f^B(n) = 2^{n^{\Omega(1)}}$	$\mathsf{EXP}^{\mathcal{A}}$	$(\log n)^{O(1)}$
$\exists f \in \mathsf{E}^{\mathcal{A}} : C_f^B(n) = 2^{\Omega(n)}$	$\mathsf{E}^{\mathcal{A}}$	$O(\log n)$

In this paper, we illustrate the power of our generalization by applying it to the following randomized processes from different areas of theoretical computer science:

(i) the Valiant–Vazirani random hashing procedure which prunes the number of satisfying assignments of a propositional formula to one [VV86],
(ii) exact learning of circuits using equivalence queries and access to an NP oracle [BCG+96],
(iii) the construction of matrices with high rigidity [Val77],
(iv) the construction of polynomial size universal traversal sequences [AKL+79].

Consider the first item above. Valiant and Vazirani [VV86] give a randomized algorithm that takes as input a propositional formula ϕ and outputs with high probability a list of formulas with the following property: If ϕ is satisfiable, then some

formula in the output list has exactly one satisfying assignment; if ϕ is unsatisfiable, then no formula in the list is satisfiable. Strictly speaking, the Valiant–Vazirani procedure is not a "BPP algorithm," and it is not clear how one could use a pseudorandom generator for BPP to derandomize it. One reason for this is that checking if the Valiant–Vazirani procedure was "successful," (i.e., it outputs a list of formulas with the above property) does not seem to be computable in polynomial time. Checking if the Valiant–Vazirani procedure succeeds for a particular input and choice of random bits *is* polynomial-time computable, however, if we are given access to an oracle for SAT. From this observation, we can prove that the existence of a Boolean function in exponential time with high SAT-oracle circuit complexity implies the existence of a pseudorandom generator which can be used to derandomize the Valiant–Vazirani procedure.

More specifically, we show that, given a propositional formula ϕ, we can construct in subexponential time a collection of polynomial size formulas satisfying the above Valiant–Vazirani property unless the polynomial-time hierarchy collapses. If there exists a language in E with SAT-oracle circuit complexity $2^{\epsilon n}$ for some $\epsilon > 0$, then we achieve a polynomial-time deterministic procedure. It follows that, under the same hypothesis, we can find in polynomial time a satisfying assignment for a propositional formula using *nonadaptive* access to an oracle for SAT. The latter contrasts with the standard adaptive method of binary search. We obtain derandomization results of a similar kind for the other items in the above list. We refer to section 5 for the precise statements.

1.4. Relationship to extractors. Our generalized hardness versus randomness trade-offs have also played a crucial role in the recent breakthrough extractor constructions by Trevisan [Tre99]. An extractor is a combinatorial object used to "extract" randomness from a corrupt or weakly random source. More specifically, applying an extractor to a probability distribution with some crude randomness results in a distribution which is almost uniform over strings of length related to the amount of crude randomness in the original distribution. We refer to the survey paper by Nisan and Ta-Shma [NT99] for background, precise definitions, and constructions.

In retrospect, from the observation that the construction due to Impagliazzo and Wigderson [IW97] is a black-box transformation of a hard function, one can directly prove the existence of the type of extractors found in the aforementioned work of Trevisan [Tre99]. See section 6 for a discussion.

1.5. Universal traversal sequences and randomized logspace. We also obtain some results related to the construction of universal traversal sequences and the derandomization of randomized logspace. Universal traversal sequences are instructions for walks on arbitrary undirected graphs of a given size that have the property that any vertex of the graphs is visited at least once by the walk. The instructions depend only on the number of vertices of the graph and not on its structure. A major open problem is to give a deterministic polynomial-time (or logspace) construction of such sequences. A logspace construction of universal traversal sequences, for example, would show that undirected connectivity is computable in logspace.

We give polynomial-time constructions of universal traversal sequences under the assumption that there exists a language in E with circuit complexity $2^{\epsilon n}$ for some ϵ. We also show that, if there is a language in linear space that requires circuits of size $2^{\Omega(n)}$, then BPL = L, where BPL denotes the languages recognizable in randomized logspace with bounded two-sided error. In fact, we need only the weaker hypothesis that there exists a language in linear space that requires branching programs of size

$2^{\epsilon n}$. This answers a question raised by Clementi, Rolim, and Trevisan [CRT98]. As a corollary, under the same hypothesis, we can generate universal traversal sequences in logspace.

1.6. Organization. Section 2 introduces our notation. In section 3, we generalize the techniques used to derandomize BPP and show how to use them in the Arthur-Merlin setting. Section 4 defines a broad class of randomized processes and shows how our approach allows us to reduce the randomness of any process that fits within this class. In section 5, we apply this framework to the four examples mentioned above. We discuss directions for further research and some recent progress in section 6.

2. Notation. Most of our complexity theoretic notation is standard. We refer the reader to the textbooks by Balcázar, Díaz, and Gabarró [BDG95] and by Papadimitriou [Pap94].

An *oracle circuit* D is a circuit with AND, OR, and NOT gates as well as oracle gates, which compute membership of the string formed by their input bits to some unspecified oracle B. The function D^B the circuit computes depends on the oracle B. For fixed B, we will use the term *B-oracle circuit* to denote an oracle circuit with access to B as an oracle. In this paper, we measure the *size* of a circuit by its number of connections.

Given a Boolean function $f : \{0,1\}^* \to \{0,1\}$ and an oracle B, the *circuit complexity* $C_f^B(n)$ of f at length n relative to B is the smallest integer t such that there is a B-oracle circuit of size t that computes f on inputs of length n. The *hardness* $H_f^B(n)$ of f at length n relative to B is the largest integer t such that, for any oracle circuit D of size at most t with n inputs,

$$\left| \Pr_x[D^B(x) = f(x)] - \frac{1}{2} \right| < \frac{1}{t},$$

where x is uniformly distributed over $\{0,1\}^n$.

A *pseudorandom generator* G is a sequence of functions $(G_n)_{n\in\mathbb{N}}$ such that G_n maps $\{0,1\}^{s(n)}$ to $\{0,1\}^n$ for some function $s : \mathbb{N} \to \mathbb{N}$ with $s(n) < n$. The function s is called the *seed length* of G. We say that G is *computable in* $\mathcal{C}$ if the problem of deciding the ith bit of $G_n(\sigma)$ given $\langle n, \sigma, i\rangle$ belongs to $\mathcal{C}$. Given an oracle B, G_n is said to be *secure against* B if, for any oracle circuit D of size at most n,

$$|\Pr_\rho[D^B(\rho) = 1] - \Pr_\sigma[D^B(G_n(\sigma)) = 1]| < \frac{1}{n}, \tag{1}$$

where ρ is uniformly distributed over $\{0,1\}^n$ and σ is uniformly distributed over $\{0,1\}^{s(n)}$. G is said to be secure against B if G_n is secure against B for almost all n.

For any function $t : \mathbb{N} \to \mathbb{N}$, AM-TIME$[t(n)]$ represents the class of languages L for which there exists a deterministic Turing machine M that runs in time $O(t(n))$ on inputs of the form $\langle x, y, z\rangle$, where $x \in \{0,1\}^n$ and $y, z \in \{0,1\}^{t(n)}$, such that, for any input x,

$$x \in L \Rightarrow \Pr_{|y|=t(n)}[\exists z \in \{0,1\}^{t(n)} : M(\langle x, y, z\rangle) = 1] \geqslant \frac{2}{3}, \tag{2}$$

$$x \notin L \Rightarrow \Pr_{|y|=t(n)}[\exists z \in \{0,1\}^{t(n)} : M(\langle x, y, z\rangle) = 1] \leqslant \frac{1}{3}, \tag{3}$$

where $n = |x|$ and the probabilities are with respect to the uniform distribution. AM denotes $\cup_{c>0}\mathsf{AM\text{-}TIME}[n^c]$. The above definition of Arthur-Merlin games allows two-sided error. Requiring a one-sided error leads to the same class modulo polynomial factors. More precisely, an Arthur-Merlin game running in time $O(t(n))$ can be transformed into an equivalent Arthur-Merlin game running in time $(t(n))^{O(1)}$ satisfying (2) with the 2/3 on the right-hand side replaced by 1 [FGM+89]. In particular, one can assume without loss of generality that every language in AM has a polynomial-time one-sided error Arthur-Merlin game. We will not exploit that fact in this paper, but later work [MV99] crucially relies on it.

$\mathsf{EXP} = \cup_{c>0}\mathsf{DTIME}[2^{n^c}]$, and $\mathsf{E} = \cup_{c>0}\mathsf{DTIME}[2^{cn}]$. Similarly, $\mathsf{NEXP} = \cup_{c>0}\mathsf{NTIME}[2^{n^c}]$, and $\mathsf{NE} = \cup_{c>0}\mathsf{NTIME}[2^{cn}]$.

For any complexity class $\mathcal{C}$, the class infinitely-often-$\mathcal{C}$ (i.o.-$\mathcal{C}$) consists of all languages L for which there is a language $L' \in \mathcal{C}$ such that $L \cap \{0,1\}^n = L' \cap \{0,1\}^n$ for infinitely many lengths n.

For any function $s : \mathbb{N} \to \mathbb{N}$, $\Omega(s)$ denotes the class of all functions $t : \mathbb{N} \to \mathbb{N}$ for which there exists a constant $\epsilon > 0$ such that $t(n) \geqslant \epsilon \cdot s(n)$ for almost all n. $\omega(s)$ is the class of t such that, for any $c > 0$ and almost all n, $t(n) \geqslant c \cdot s(n)$.

3. Derandomizing Arthur-Merlin games. In this section, we develop methods for derandomizing Arthur-Merlin games and give evidence that the class AM of languages with bounded round Arthur-Merlin games is not much larger than NP and may even coincide with it. For clarity of exposition and to provide an initial outline for our arguments, we first state all of our theorems without proofs. We then fill in details starting in section 3.1.

As is customary in the area of derandomization, our approach will be to construct pseudorandom generators with appropriate security properties. The following lemma states that, to derandomize $\mathsf{AM} = \cup_{c>0}\mathsf{AM\text{-}TIME}[n^c]$, the pseudorandom generator need only be secure against SAT. See section 3.1 for a proof.

LEMMA 3.1. *Let $s, t, \tau : \mathbb{N} \to \mathbb{N}$ be constructible functions. If there is a pseudorandom generator G computable in $\mathsf{NTIME}[\tau(n)] \cap \mathsf{coNTIME}[\tau(n)]$ and with seed length s that is secure against* SAT*, then*

$$\mathsf{AM\text{-}TIME}[t(n)] \subseteq \mathsf{NTIME}[2^{s(t'(n))} \cdot \tau(s(t'(n))) \cdot t(n)], \tag{4}$$

where $t'(n) = O(t(n) \log^2 t(n))$.

In order to build such a pseudorandom generator, we will extend previous work [NW94, BFNW93, IW97, STV01] to the nondeterministic setting of Arthur-Merlin games. The main construction in these papers is a reduction from a circuit that distinguishes the output of a pseudorandom generator based on f from the uniform distribution to a circuit capable of computing f. We argue that this construction works for any nonuniform model of computation satisfying certain closure properties and, in particular, for B-oracle circuits for any fixed oracle B. In this way, we obtain pseudorandom generators secure against B from functions which small B-oracle circuits cannot compute.

We begin with a generalized version of the main result of Nisan and Wigderson [NW94]. Their construction works for any nonuniform model which is closed under precomputation and complementation. In particular, it carries through for oracle circuits. See section 3.2 for a more detailed argument.

THEOREM 3.2. *Let B be any oracle, let g be a Boolean function, and let $\ell : \mathbb{N} \to \mathbb{N}$ be a constructible function. If*

$$H_g^B(\ell(n)) \geqslant n^2,$$

then there exists a pseudorandom generator G which is secure against B and has a constructible seed length $s(n) = O(\ell^2(n)/\log n)$. Computing any bit of $G(\sigma)$ for $\sigma \in \{0,1\}^{s(n)}$ takes time $2^{O(s(n))}$ plus one evaluation of g on an input of length $\ell(n)$.

Theorem 3.2 states that the existence of functions with high average-case oracle circuit complexity, say, SAT oracles, implies the existence of pseudorandom generators secure against the corresponding oracle, in this case SAT. We would like to relax this assumption from average-case to worst-case, however, so we will need to examine worst-case to average-case reductions, also known as *hardness amplifications*.

The worst-case to average-case hardness transformation of Babai et al. [BFNW93] requires only closure of the model of computation under majority, complementation, and certain arithmetic field operations. Since oracle circuits have these closure properties, the transformation also works for this model. This observation enables us to establish the first two lines of derandomization results for Arthur-Merlin games in Table 1 but falls short in establishing the last line. For the last line, sometimes referred to as a "high-end" derandomization, we appeal to the reduction in Impagliazzo and Wigderson [IW97], which also carries through for oracle circuits and is powerful enough to establish all of Table 1. An alternate to the Impagliazzo–Wigderson construction is described by Sudan, Trevisan, and Vadhan [STV01] and is more amenable to generalization. In section 3.3, we provide the details of this generalization and obtain the following hardness amplification result.

THEOREM 3.3. *There exist positive constants γ and δ such that, for any oracle B, Boolean function f, and constructible function $h : \mathbb{N} \to \mathbb{N}$ satisfying*

$$h(n) \leqslant (C_f^B(\gamma n))^\delta / n,$$

the following holds: There exists a Boolean function g such that $H_g^B(n) \geqslant h(n)$. Computing g on an input of length n takes time $2^{O(n)}$ plus evaluating f on all inputs of length γn.

Combining Theorems 3.2 and 3.3 yields our main derandomization tool. Some instantiations are given in Table 2.

THEOREM 3.4. *There exists a positive constant c such that the following holds for any class $\mathcal{A}$ of oracles, oracle B, a Boolean function $f \in \mathsf{E}^{\mathcal{A}}$, and a constructible function $\ell : \mathbb{N} \to \mathbb{N}$: If*

$$C_f^B(\ell(n)) \geqslant n^c,$$

then there exists a pseudorandom generator G computable in $\mathsf{E}^{\mathcal{A}}$ which is secure against B and has a constructible seed length $s(n) = O(\ell^2(n)/\log n)$. The same holds if E is replaced by EXP.

In this section, we will apply Theorem 3.4 with $\mathcal{A} = \mathsf{NP} \cap \mathsf{coNP}$ and $B = \mathrm{SAT}$. Note that $\mathsf{EXP}^{\mathsf{NP} \cap \mathsf{coNP}} = \mathsf{NEXP} \cap \mathsf{coNEXP}$. Together with Lemma 3.1, Theorem 3.4 yields the following result.

THEOREM 3.5. *If there is a Boolean function $f \in \mathsf{NEXP} \cap \mathsf{coNEXP}$ such that $C_f^{\mathrm{SAT}}(n) = n^{\omega(1)}$, then*

$$\mathsf{AM} \subseteq \cap_{\epsilon > 0} \mathsf{NTIME}[2^{n^\epsilon}].$$

We can rephrase the weak version of Theorem 3.5 as follows.

THEOREM 3.6. *If $\mathsf{NEXP} \cap \mathsf{coNEXP} \not\subseteq \mathsf{P}^{\mathsf{NP}}/\mathsf{poly}$, then $\mathsf{AM} \subseteq \cap_{\epsilon > 0} \text{i.o.-}\mathsf{NTIME}[2^{n^\epsilon}]$.*

So, if the conclusion of Theorem 3.6 fails to hold, then $\mathsf{NEXP} \cap \mathsf{coNEXP} \subseteq$

$\mathsf{P}^{\mathsf{NP}}/\mathsf{poly}$, which implies that $\mathsf{EXP} = \Sigma_3^{\mathsf{p}} \cap \Pi_3^{\mathsf{p}}$ [KL82]. Therefore, we obtain the following theorem.

THEOREM 3.7. *If exponential time does not collapse to the third level of the polynomial-time hierarchy, then every language in* AM*, and graph nonisomorphism, in particular, has subexponential size proofs for infinitely many lengths.*

More generally, Theorem 3.4 yields the following derandomization result for Arthur-Merlin games. Note that $\mathsf{E}^{\mathsf{NP}\cap\mathsf{coNP}} = \mathsf{NE} \cap \mathsf{coNE}$.

THEOREM 3.8. *If there is a Boolean function $f \in \mathsf{NE} \cap \mathsf{coNE}$ and a constructible function $\ell : \mathbb{N} \to \mathbb{N}$ such that*

$$C_f^{\mathrm{SAT}}(\ell(n)) = \Omega(n),$$

then

$$\mathsf{AM} \subseteq \cup_{c>0}\mathsf{NTIME}[2^{\ell^2(n^c)/\log n}].$$

Theorem 3.8 yields a range of hardness versus randomness trade-offs for the various choices of the parameter ℓ. Since $C_f^B(n)$ is always at most $O(2^n/n)$, the hypothesis of Theorem 3.8 cannot be met for ℓ sublogarithmic. On the other hand, the conclusion becomes trivial in the case where ℓ is polynomial. Therefore, the interesting range for ℓ lies between logarithmic and subpolynomial. For example, for ℓ polylogarithmic, Theorem 3.8 (combined with some padding) yields the second line in Table 1 and the following. Recall that quasi-polynomial means $2^{\log^{O(1)} n}$.

THEOREM 3.9. *If there is a Boolean function $f \in \mathsf{NEXP} \cap \mathsf{coNEXP}$ such that $C_f^{\mathrm{SAT}}(n) = \Omega(2^{n^\epsilon})$ for some $\epsilon > 0$, then every language in* AM *has quasi-polynomial size proofs.*

Theorem 3.8 achieves complete derandomizations of Arthur-Merlin games in case of logarithmic ℓ.

THEOREM 3.10. *If there is a Boolean function $f \in \mathsf{NE}\cap\mathsf{coNE}$ such that $C_f^{\mathrm{SAT}}(n) \in 2^{\Omega(n)}$, then $\mathsf{AM} = \mathsf{NP}$. In particular, the same hypothesis implies that graph nonisomorphism has polynomial size proofs.*

We wish to point out that, under the hypothesis of Theorem 3.10, the proof of Lemma 3.1 describes an explicit certificate for graph nonisomorphism or any other AM-language.

We also note that, for derandomizing AM, it is actually sufficient to construct efficient pseudorandom generators that are secure against SAT-oracle circuits with *parallel* access to the oracle. All theorems in this section also hold for oracle circuits with such restricted access.

Finally, we note that our derandomizations also apply to Arthur-Merlin games where the number of rounds of interaction is not necessarily bounded by a constant. Via standard transformations (see [BM88]), one can show that any language possessing an Arthur-Merlin game with ℓ rounds and verifier complexity $t(n)$ is contained in $\mathsf{AM}\text{-}\mathsf{TIME}[t'(n)]$, where $t'(n) = t(n)^{O(\ell)}$), so we can apply the theorems from this section directly.

3.1. Proof of Lemma 3.1. Let L be a language satisfying (2) and (3), and consider

$$L' = \{\langle x, y\rangle \mid y \in \{0,1\}^{t(|x|)} \text{ and } \exists\, z \in \{0,1\}^{t(|x|)} : M(\langle x, y, z\rangle) = 1\}.$$

By Cook's theorem [Coo71, Coo88], since L' is in nondeterministic linear time, there exists a circuit of size $t'(n) = O(t(n)\log^2 t(n))$ that many-one reduces L' to SAT on

```
counter ← 0
for every σ ∈ {0,1}^{s(t'(n))}
    for  i ← 1,...,t(n)
         guess ρ_i ∈ {0,1}
         guess a computation path p for N(⟨n,σ,i,ρ_i⟩)
         if N(⟨n,σ,i,ρ_i⟩) rejects along p
            then reject and abort
    guess z ∈ {0,1}^{t(n)}
    counter ← counter + M(⟨x,ρ,z⟩)
if counter > 2^{t'(n)-1} then accept
```

FIG. 1. *Nondeterministic algorithm for deciding L.*

inputs $\langle x, y\rangle$ with $x \in \{0,1\}^n$ and $y \in \{0,1\}^{t(n)}$. For any fixed value of $x \in \{0,1\}^n$, let C_x denote the corresponding oracle circuit obtained by hardwiring x. Note that C_x makes a single oracle query and outputs the answer to that query. Since G is a pseudorandom generator secure against SAT, conditions (2) and (3) imply that

$$\Pr_{\sigma}[C_x^{\mathrm{SAT}}(G_{t'(n)}(\sigma)) = 1] \begin{cases} > \frac{1}{2} & \text{if } x \in L, \\ < \frac{1}{2} & \text{if } x \notin L, \end{cases} \tag{5}$$

where the probability is with respect to the uniform distribution of σ over $\{0,1\}^{s(t'(n))}$.

Since G is computable in $\mathsf{NTIME}[\tau(n)] \cap \mathsf{coNTIME}[\tau(n)]$, there exists a nondeterministic Turing machine N running in time τ that accepts $\{\langle n, \sigma, i, b\rangle \mid i\text{th bit of } G_n(\sigma) \text{ equals } b\}$. Now consider the nondeterministic algorithm in Figure 1.

Figure 1 describes a nondeterministic Turing machine that runs in time $2^{s(t'(n))} \cdot \tau(s(t'(n))) \cdot t(n)$. The largest possible value of counter at the end of the outer loop over all possible nondeterministic choices in the algorithm of Figure 1 equals $2^{s(t'(n))} \cdot \Pr_\sigma[C_x^{\mathrm{SAT}}(G_{t'(n)}(\sigma)) = 1]$. It follows from (5) that the machine accepts L. This finishes the proof of Lemma 3.1.

3.2. Proof of Theorem 3.2. This and the next section establish our generalization of known hardness versus randomness trade-offs. The key observation is that, using the appropriate notion of relativization, the known hardness versus randomness trade-offs *relativize*.

Relativization is the idea of giving all computing devices access to one or more oracles for arbitrary but fixed computational problems. Most theorems in computational complexity relativize; i.e., they hold under relativization. This is often just stated as a remark, as it is straightforward to verify that the proof of the theorem carries through when oracles are introduced. The few known nonrelativizing results are almost all in the area of interactive proofs and probabilistically checkable proofs. One of their distinguishing features is that they rely on considerable combinatorics and algebra. The latter are also crucial ingredients in the known hardness versus randomness trade-offs, which might lead one to believe that these trade-offs also do not relativize. However, a careful examination in this and the next section will reveal that the hardness versus randomness trade-offs do relativize. In fact, they relativize in the very strong way stated in Theorem 3.4.

We start with the construction of a pseudorandom generator out of a Boolean function that is hard on average. Nisan and Wigderson use the following notion of a combinatorial design in their construction of a pseudorandom generator.

DEFINITION 3.11 (see [NW94]). *An (m, ℓ) design of size k over a universe Ω is a collection $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ of subsets of Ω, each of size ℓ, such that, for any $1 \leqslant i < j \leqslant k$, the intersection $S_i \cap S_j$ has size at most m.*

In particular, in order to construct G_n, Nisan and Wigderson need a $(\log n, \ell)$ design of size n for a given value of $\ell \geqslant \log n$. The size of the design universe will be the seed length $s(n)$ of G_n and therefore should be small. Nisan and Wigderson [NW94] show that the greedy approach works and that it yields a $(\log n, \ell)$ design of size n over a universe of size $O(\ell^2/\log n)$. See [Tre99, Lemma 7] for a detailed analysis.

LEMMA 3.12 (see [NW94]). *A $(\log n, \ell)$ design of size n can be constructed over a universe of size $s \in O(\ell^2/\log n)$ in time $2^s \cdot n^{O(1)}$.*

Given a function $g : \{0,1\}^\ell \to \{0,1\}$ and a $(\log n, \ell)$ design $\mathcal{S} = (S_1, S_2, \ldots, S_n)$ over $\{1, 2, \ldots, s\}$, Nisan and Wigderson define their pseudorandom generator G_n as follows:

$$G_n : \{0,1\}^s \to \{0,1\}^n : \sigma \to g(\sigma|_{S_1})\, g(\sigma|_{S_2}) \ldots g(\sigma|_{S_n}), \tag{6}$$

where $\sigma|_{S_i}$ denotes the substring of σ consisting of the bits indexed by the elements from S_i.

The following theorem, combined with Lemma 3.12, finishes the proof of Theorem 3.2.

THEOREM 3.13. *Let B be any oracle, let $g : \{0,1\}^\ell \to \{0,1\}$, and let $\mathcal{S} = (S_1, S_2, \ldots, S_n)$ a $(\log n, \ell)$ design over $\{1, 2, \ldots, s\}$. If $H_g^B \geqslant n^2$, then G_n as defined by (6) is secure against B.*

The proof follows along the lines of the one by Nisan and Wigderson [NW94]. It is an application of Yao's observation that distinguishability from the uniform distribution implies predictability [Yao82] and uses the so-called hybrid argument [GM84].

Proof. We proceed by contradiction. Assume that G_n is not secure against B. This means that there exists an oracle circuit D of size at most n such that either

$$\Pr_\sigma[D^B(G_n(\sigma)) = 1] - \Pr_\rho[D^B(\rho) = 1] > \frac{1}{n} \tag{7}$$

or else $\Pr_\rho[D^B(\rho) = 1] - \Pr_\sigma[D^B(G_n(\sigma)) = 1] > \frac{1}{n}$, where ρ is uniformly distributed over $\{0,1\}^n$ and σ over $\{0,1\}^s$. We will assume that (7) holds. The other case is similar.

Consider the following sequence of "hybrid" distributions $\mathcal{D}_0, \ldots, \mathcal{D}_n$:

$$\begin{array}{lcccccccccc}
\mathcal{D}_0 & = & g(\sigma|_{S_1}) & g(\sigma|_{S_2}) & & & \ldots & & & g(\sigma|_{S_{n-1}}) & g(\sigma|_{S_n}) \\
\mathcal{D}_1 & = & \rho_1 & g(\sigma|_{S_2}) & & & \ldots & & & g(\sigma|_{S_{n-1}}) & g(\sigma|_{S_n}) \\
\vdots & & & & & & \vdots & & & & \\
\mathcal{D}_i & = & \rho_1 & \rho_2 & \ldots & \rho_{i-1} & \rho_i & g(\sigma|_{S_{i+1}}) & \ldots & g(\sigma|_{S_{n-1}}) & g(\sigma|_{S_n}) \\
\vdots & & & & & & \vdots & & & & \\
\mathcal{D}_{n-1} & = & \rho_1 & \rho_2 & & & \ldots & & & \rho_{n-1} & g(\sigma|_{S_n}) \\
\mathcal{D}_n & = & \rho_1 & \rho_2 & & & \ldots & & & \rho_{n-1} & \rho_n
\end{array}$$

As before, ρ is uniformly distributed over $\{0,1\}^n$ and σ over $\{0,1\}^s$.

Since the left-hand side of (7) can be written as

$$\Pr_{y\in\mathcal{D}_0}[D^B(y)=1]-\Pr_{z\in\mathcal{D}_n}[D^B(z)=1]=\sum_{i=1}^{n}\Pr_{y\in\mathcal{D}_{i-1}}[D^B(y)=1]-\Pr_{z\in\mathcal{D}_i}[D^B(z)=1],$$

there must exist an index i, $1\leqslant i\leqslant n$, such that

$$\Pr_{y\in\mathcal{D}_{i-1}}[D^B(y)=1]-\Pr_{z\in\mathcal{D}_i}[D^B(z)=1]\geqslant\frac{1}{n^2}. \tag{8}$$

The only difference between $\mathcal{D}_{i-1}$ and $\mathcal{D}_i$ is the way the ith bit is generated: In $\mathcal{D}_{i-1}$, it is set to $g(\sigma|_{S_i})$, whereas in $\mathcal{D}_i$, it is a random bit. Equation (8) says that, in the former case, the B-oracle circuit D^B is more likely to accept than in the latter case. This suggests the following randomized predictor $P(x)$ for $g(x)$:

(i) Pick $\rho_1,\rho_2,\ldots,\rho_i\in\{0,1\}$ uniformly at random.
(ii) Set $\sigma|_{S_i}$ equal to x, and pick the other bits of σ uniformly at random.
(iii) If $D^B(\rho_1,\rho_2,\ldots,\rho_i,g(\sigma|_{S_{i+1}}),\ldots,g(\sigma|_{S_n}))$ accepts, then output ρ_i; otherwise, output $\overline{\rho_i}$ (the complement of ρ_i).

Claim 3.14.

$$\Pr_{\rho,\sigma}[P(x)=g(x)]-\frac{1}{2}=\Pr_{y\in\mathcal{D}_{i-1}}[D^B(y)=1]-\Pr_{z\in\mathcal{D}_i}[D^B(z)=1], \tag{9}$$

where x denotes $\sigma|_{S_i}$.

We argue the claim by conditioning on all of ρ and σ except ρ_i. Let $\tilde{y}$ denote y with the ith bit flipped.

(i) Case $D^B(y)=D^B(\tilde{y})$. Since the value of the ith component does not affect D^B, the right-hand side of (9) vanishes. $P(x)$ is a random bit in this case, so the left-hand side also vanishes.

(ii) Case $D^B(y)\neq D^B(\tilde{y})$. Then $D^B(z)$ accepts with probability $\frac{1}{2}$, and $P(x)$ equals $g(x)$ precisely when $D^B(y)=1$.

This finishes the proof of the claim.

Note that $x\doteq\sigma|_{S_i}$ is uniformly distributed over $\{0,1\}^m$. By an averaging argument, (8) and (9) imply that we can fix ρ and all of σ outside of S_i such that

$$\Pr_x[P(x)=g(x)]\geqslant\frac{1}{2}+\frac{1}{n^2}. \tag{10}$$

Note that

$$P(x)=D^B(\rho_1,\rho_2,\ldots,\rho_i,g(\sigma|_{S_{i+1}}),\ldots,g(\sigma|_{S_n}))\oplus\overline{\rho_i}. \tag{11}$$

Since $|S_j\cap S_i|\leqslant\log n$ for $j\neq i$, each of the components $g(\sigma|_{S_j})$, $i<j\leqslant n$, depend only on $\log n$ bits of x and therefore can be computed by circuits of size at most n with input x. There are at most $n-1$ such components. As ρ is fixed and the size of D is at most n, it follows that the right-hand side of (11) can be evaluated by a B-oracle circuit C^B of size at most $(n-1)n+n=n^2$. In combination with (10), this leads to the contradiction that $H_g^B<n^2$. □

3.3. Proof of Theorem 3.3. Sudan, Trevisan, and Vadhan showed the following list decoding result which is used to give a strong worst-case to average-case hardness amplification [STV01, Definition 22 and Lemma 28].

Theorem 3.15 (see [STV01]). *There exists a family of maps $(\mathcal{C}_{N,\epsilon})_{N\in\mathbb{N},\epsilon>0}$, where $\mathcal{C}_{N,\epsilon}:\{0,1\}^N\to\{0,1\}^{N'(N,\epsilon)}$ such that the following hold.*

(i) *$\mathcal{C}_{N,\epsilon}$ is computable in time polynomial in N and $1/\epsilon$. In particular, $N' \doteq N'(N,\epsilon)$ is polynomial in N and $1/\epsilon$.*

(ii) *There exists a randomized algorithm that, on input N and ϵ and given oracle access to a string $y \in \{0,1\}^{N'}$, runs in time polynomial in $\log N$ and $1/\epsilon$ and outputs with high probability a list of randomized oracle machines M_j with the following properties:*

- *For every $x \in \{0,1\}^N$ such that $(\mathcal{C}_{N,\epsilon}(x))_i = y_i$ for at least $(\frac{1}{2}+\epsilon)N'$ of the positions i, $1 \leqslant i \leqslant N'$, there exists an index j such that, for any position i, $1 \leqslant i \leqslant N$, $M_j^y(i)$ outputs x_i with high probability.*
- *Each machine M_j runs in time polynomial in $\log N$ and $1/\epsilon$.*

By Adleman's argument [Adl78], we can transform the randomized procedures in (ii) of Theorem 3.15 into circuits. This yields the following corollary.

COROLLARY 3.16. *There exists a constant c such that, for any $N \in \mathbb{N}$, $x \in \{0,1\}^N$, and $y \in \{0,1\}^{N'}$ satisfying $(\mathcal{C}_{N,\epsilon}(x))_i = y_i$ for at least $(\frac{1}{2}+\epsilon)N'$ of the positions i, $1 \leqslant i \leqslant N'$, there exists an oracle circuit D of size $(\frac{1}{\epsilon}\log N)^c$ such that $D^y(i) = x_i$ for every $1 \leqslant i \leqslant N$.*

We define g such that its truth-table at length $n' \doteq \log N'$ equals the encoding $\mathcal{C}_{N,\epsilon}$ of the truth-table of f at length $n \doteq \log N$. We set $\epsilon = 1/h(n')$, where $h(n')$ is the hardness level we want to achieve. Since $h(n') \in O(2^{n'}/n')$, (i) in Theorem 3.15 implies that N and N' are polynomially related, so we can set $n = \gamma \cdot n'$ for some positive constant γ.

Now suppose that there exists a B-oracle circuit of size at most $h(n')$ that computes g correctly on at least $(\frac{1}{2}+\frac{1}{h(n')})N'$ of the inputs of length n'. By plugging this circuit in as the oracle y in Corollary 3.16, we obtain a B-oracle circuit that computes f at length $n = \gamma \cdot n'$ correctly and has size at most $(\frac{1}{\epsilon}\log N)^c \cdot h(n') = (\gamma n')^c \cdot (h(n'))^{c+1}$. In other words,

$$C_f^B(\gamma n') \leqslant (\gamma n')^c \cdot (h(n'))^{c+1}. \tag{12}$$

For $\delta < 1/(c+1)$, (12) contradicts the upper bound on $h(n')$ in the hypothesis of Theorem 3.3. Therefore, $H_g^B(n') \geqslant h(n')$. □

4. A general framework for derandomization. In the previous section, we showed that an Arthur-Merlin protocol can be viewed as an SAT-oracle distinguisher for a pseudorandom generator, and, if a Boolean function f exists with sufficient hardness against SAT-oracle circuits, we can construct a pseudorandom generator based on f that will look random to our Arthur-Merlin protocol. Still, we have only applied our results to randomized *decision* algorithms. In this section, we show how to relax this condition and obtain hardness versus randomness trade-offs for a broader class of randomized processes. Under a sufficient hardness condition depending upon the particular randomized algorithm, we are able to reduce the algorithm's randomness to a logarithmic factor and, in some cases, provide a complete derandomization. Weaker hardness conditions yield partial derandomizations.

We first define the notion of a *randomized process* to which our approach applies.

DEFINITION 4.1. *A randomized process that uses $r(n)$ random bits on inputs of length n is a pair (F,π), where the following hold:*

(i) *F is a function that takes a string x and a string ρ of length $r(|x|)$, and outputs the outcome of the process on input x using ρ as the random bit sequence.*

(ii) *π is a predicate with the same domain as F. Intuitively π indicates whether the process succeeds on input x using ρ, i.e., whether ρ is a "good" choice*

of random bits for the given input x. We call π the success predicate of the randomized process.

In the case of Arthur-Merlin games, F is Boolean. More specifically, for the game defined by (2) and (3) in section 2, $F(x,\rho)$ coincides with the predicate $\exists z \in \{0,1\}^{t(n)} : M(\langle x,\rho,z\rangle) = 1$; the success predicate $\pi(x,\rho)$ equals $F(x,\rho)$ if $x \in L$ and the complement of $F(x,\rho)$ otherwise. In the specific randomized processes we will consider in section 5, F will be non-Boolean.

What matters for our derandomization results are the complexity of the success predicate π and, more precisely, the property stated in the following straightforward lemma.

LEMMA 4.2. *Let B be any oracle, and let (F,π) be a randomized process using $r(n)$ random bits such that, for any fixed input x of length n, the predicate $\pi_x : \{0,1\}^{r(n)} \to \{0,1\}$, where $\pi_x(\rho) \doteq \pi(x,\rho)$, can be decided by a B-oracle circuit of size $t(n) \geqslant r(n)$. If G is a pseudorandom generator with seed length $s(n)$ which is secure against B, then, for any input x of length n,*

$$|\Pr_{\rho}[\pi(x,\rho)=1] - \Pr_{\sigma}[\pi(x,G_{t(n)}(\sigma)[1..r(n)])=1]| = O(1/t(n)),$$

where ρ is uniformly distributed over $\{0,1\}^{r(n)}$ and σ is uniformly distributed over $\{0,1\}^{s(t(n))}$.

In our applications, we will be concerned with randomized processes that use a polynomial number of random bits. We will choose the oracle B in Lemma 4.2 powerful enough so that it can check the process efficiently in the following sense.

DEFINITION 4.3. *Let B be an oracle, and let (F,π) be a randomized process using a polynomial number $r(n)$ of random bits. We say that B can* efficiently check *(F,π) if there is a polynomial p such that, for any fixed input x of length n, the predicate $\pi_x : \{0,1\}^{r(n)} \to \{0,1\}$, where $\pi_x(\rho) \doteq \pi(x,\rho)$, can be decided by a B-oracle circuit of size $p(n)$.*

Using the success predicate as a distinguisher as in Lemma 4.2, Theorem 3.4 yields our general derandomization result.

THEOREM 4.4. *Let $\mathcal{A}$ be a class of oracles, B an oracle, d a positive constant, and $\ell : \mathbb{N} \to \mathbb{N}$ a constructible function. Let (F,π) be a randomized process using a polynomial number of random bits such that B can efficiently check (F,π). If there exists a Boolean function $f \in \mathsf{E}^{\mathcal{A}}$ such that $C_f^B(\ell(n)) = \Omega(n)$, then there exist a function G computable in $\mathsf{E}^{\mathcal{A}}$ and a constructible function $s(n) = O(\ell^2(n^{O(1)})/\log n)$ such that, for any input x of length n,*

$$|\Pr_{\rho}[\pi(x,\rho)=1] - \Pr_{\sigma}[\pi(x,G(\sigma))=1]| = O(1/n^d),$$

where ρ is uniformly distributed over $\{0,1\}^{r(n)}$ and σ is uniformly distributed over $\{0,1\}^{s(n)}$. The same holds if E is replaced by EXP.

See Table 2 for some interesting instantiations of Theorem 4.4.

In order to reduce the randomness of a randomized process, we will first analyze the complexity of an oracle B capable of efficiently checking the associated success predicate and then construct a pseudorandom generator secure against B based on a function with presumed hardness against B. We will see several examples of this in the next section.

5. More applications. We will now apply the general framework of section 4 to various other constructions in computational complexity. As is customary, we

state only our results in terms of the strongest of the assumptions in Table 2, yielding polynomial-time deterministic simulations. It should be noted, however, that weaker assumptions can be taken (e.g., that the polynomial-time hierarchy does not collapse) in order to achieve weaker, but still subexponential, deterministic simulations.

5.1. Valiant–Vazirani. Our first example is the randomized Boolean hashing protocol developed by Valiant and Vazirani [VV86]. They give a method for pruning the satisfying assignments of a propositional formula to one.

THEOREM 5.1 (see [VV86]). *There exists a randomized polynomial-time algorithm that, on input a propositional formula ϕ, outputs a list of propositional formulas such that*

(i) *every satisfying assignment to any of the formulas on the list also satisfies ϕ;*
(ii) *if ϕ is satisfiable, then, with high probability, at least one of the formulas on the list has exactly one satisfying assignment.*

Let $F(x, \rho)$ denote the list of formulas the Valiant–Vazirani algorithm produces on input x using coin flips specified by ρ. We define the success predicate $\pi(x, \rho)$ to hold unless x is a satisfiable formula and none of the formulas in $F(x, \rho)$ has a unique satisfying assignment. It is clear that (F, π) corresponds to a formalization of the Valiant–Vazirani process and fits within our framework.

THEOREM 5.2. *If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\mathrm{SAT}}(n) = 2^{\Omega(n)}$, then, given a propositional formula ϕ, we can generate in polynomial time a list of propositional formulas such that*

(i) *every satisfying assignment to any of the formulas on the list also satisfies ϕ;*
(ii) *if ϕ is satisfiable, then at least one of the formulas on the list has exactly one satisfying assignment.*

Proof. Let (F, π) be the formalization as described above. Notice that checking whether a given propositional formula has at least two satisfying assignments is an NP question. Hence we can check, in polynomial time, whether a propositional formula has a unique satisfying assignment using two queries to SAT. It follows that SAT can efficiently check (F, π). Applying Theorem 4.4 to (F, π) yields a pseudorandom generator G computable in E that looks random to the Valiant–Vazirani process. Enumerating over all seeds and collecting all formulas produce the desired list of formulas in polynomial time. □

We now have the following corollary about computing satisfying assignments nonadaptively.

COROLLARY 5.3. *If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\mathrm{SAT}}(n) = 2^{\Omega(n)}$, then, given a satisfiable propositional formula ϕ, we can find a satisfying assignment for ϕ in polynomial time given nonadaptive oracle access to* SAT.

Proof. A satisfying assignment to a uniquely satisfiable propositional formula ψ can be found in polynomial time using nonadaptive oracle queries to SAT: We set a variable v of ψ iff the formula obtained from ψ by substituting TRUE for v is satisfiable.

We run the latter procedure on each formula ψ of the list produced by Theorem 5.2 on input ϕ and output the first satisfying assignment we obtain. □

We also obtain interesting structural observations. Recall that the class $\#\mathsf{P}$ contains all functions $f : \Sigma^* \to \mathbb{N}$ for which there exists a nondeterministic polynomial-time Turing machine M such that $f(x)$ equals the number of accepting computations of M on input x. A language L belongs to $\oplus\mathsf{P}$ if there exists a $\#\mathsf{P}$ function f such that an input x belongs to L iff $f(x)$ is odd. A GapP function is the difference of two $\#\mathsf{P}$ functions, and SPP denotes the class of all languages whose characteristic

function is a GapP function. The class PP contains all languages L for which there exists an $f \in \mathsf{GapP}$ such that $x \in L$ iff $f(x) > 0$. SPP is contained in both $\oplus\mathsf{P}$ and PP. We refer the reader to the survey by Fortnow [For97] for background on these counting classes. Theorem 5.2 implies the following corollary.

COROLLARY 5.4. *If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\mathrm{SAT}}(n) = 2^{\Omega(n)}$, then NP is contained in SPP.*

Proof. Let $h(\phi, i)$ denote the #P function that gives the number of satisfying assignments to the ith formula of the list produced by Theorem 5.2 on input ϕ. Then

$$1 - \prod_i (1 - h(\phi, i)) \tag{13}$$

equals the characteristic function of SAT. By the closure properties of GapP [FFK94], (13) belongs to GapP. □

Similarly, we can conditionally derandomize the result by Toda and Ogiwara [TO92] that the polynomial-time hierarchy does not add power to GapP in a randomized setting. Applying our techniques to their main lemma yields the following theorem.

LEMMA 5.5. *Let B be any oracle. If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\mathrm{SAT}^B}(n) = 2^{\Omega(n)}$, then $\mathsf{GapP}^{\mathsf{NP}^B}$ is contained in GapP^B.*

This allows us to show the following theorem.

THEOREM 5.6. *For any integer $k \geqslant 1$, the following holds: If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\mathrm{TQBF}_k}(n) = 2^{\Omega(n)}$, then Σ_k^{p} is contained in SPP.*

Here TQBF_k denotes the language of all true fully quantified propositional formulas with at most $k-1$ quantifier alternations.

Proof. Corollary 5.4 relativizes and, under the given assumption, implies that the characteristic function of any language in Σ_k^{p} is contained in $\mathsf{GapP}^{\mathrm{TQBF}_{k-1}}$. The successive application of Lemma 5.5 with $B = \mathrm{TQBF}_{k-2}$, $B = \mathrm{TQBF}_{k-3}$, ..., $B = \mathrm{TQBF}_1$, and $B = \varnothing$ yields, under the given assumption, that $\mathsf{GapP}^{\mathrm{TQBF}_k}$ is contained in GapP. □

COROLLARY 5.7. *Let B be any oracle hard for the polynomial-time hierarchy under polynomial-time Turing reductions. If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^B(n) = 2^{\Omega(n)}$, then the polynomial-time hierarchy is contained in SPP.*

Corollary 5.7 is strongly related to Toda's theorem [Tod91] that the polynomial-time hierarchy lies in $\mathsf{BP} \cdot \oplus\mathsf{P}$. Applying our derandomization technique directly to Toda's theorem, along with the observation that $\oplus\mathsf{P}$ is closed under majority [PZ83] (in fact $\oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$), yields the following theorem.

THEOREM 5.8. *If there is a Boolean function $f \in \mathsf{E}$ such that $C_f^{\oplus\mathrm{SAT}}(n) = 2^{\Omega(n)}$, then the polynomial-time hierarchy is contained in $\oplus\mathsf{P}$.*

5.2. Learning circuits. Learning theory represents another area where we can apply our techniques. We will focus on exact concept learning with equivalence queries [Ang88, Lit88], in which the learner presents hypotheses to a teacher, who then tells the learner whether the hypothesis agrees with the concept in question. If it does, the learner has succeeded; otherwise, the teacher provides a counterexample, and the learner continues.

A fundamental question is whether we can efficiently learn Boolean circuits in this model. If $\mathsf{P} = \mathsf{NP}$, we can; if one-way functions exist, we cannot [PW90]. Without any complexity theoretic assumption, Bshouty et al. [BCG+96] showed that access to an NP oracle and to a source of randomness suffice to efficiently learn Boolean circuits. We give evidence that we may be able to dispense with the source of randomness.

THEOREM 5.9. *If there is a Boolean function $f \in \mathsf{NE} \cap \mathsf{coNE}$ such that $C_f^{\mathrm{TQBF}_2}(n) = 2^{\Omega(n)}$, then we can perform the following task in deterministic polynomial time given access to an* NP *oracle: exactly learn Boolean circuits of size t using equivalence queries with hypotheses that are circuits of size $O(tn + n \log n)$.*

As before, TQBF_2 denotes the language of all true, fully quantified propositional formulas with at most one quantifier alternation.

Proof. We apply Theorem 4.4 to a randomized process (F, π) underlying the algorithm of Bshouty et al. [BCG+96]. We define F in the obvious way, namely, as the candidate-equivalent circuit the learner outputs. The specification of the success predicate π is somewhat more involved than in the examples we have seen so far. We want only to consider a random seed as "good" if the learner, when using this seed, finds an equivalent circuit no matter how the teacher picks the counterexamples. Therefore, we define $\pi(x, \rho)$ to indicate whether, for every choice of candidate counterexamples, either one of them fails to be a valid counterexample on input x and random seed ρ or the learner ends up with an equivalent circuit to x. It follows from the construction of Bshouty et al. that π is a Π_2^{p} predicate, and their analysis shows that, for any input x, $\pi(x, \rho)$ holds with high probability.

Using the derandomization provided by Theorem 4.4, we end up with a polynomial number of candidate circuits, at least one of which is equivalent to x. So we just present each of these to the teacher and will succeed. □

Bshouty et al. used their learning theory result to improve the known collapse of the polynomial-time hierarchy in case NP would have polynomial size circuits: They showed that the latter implies that the polynomial-time hierarchy is contained in $\mathsf{ZPP}^{\mathsf{NP}}$. Along similar lines, we obtain the following derandomization.

COROLLARY 5.10. *If* NP *has polynomial size circuits and there is a Boolean function $f \in \mathsf{E}^{\mathsf{NP}}$ such that $C_f(n) = 2^{\Omega(n)}$, then the polynomial-time hierarchy is contained in P^{NP}.*

Proof. If NP has polynomial size circuits and there is a Boolean function $f \in \mathsf{E}^{\mathsf{NP}}$ such that $C_f(n) = 2^{\Omega(n)}$, then the same function f also satisfies $C_f^{\mathrm{SAT}}(n) = 2^{\Omega(n)}$. Therefore, by applying Theorem 3.4 with $\mathcal{A} = \mathsf{NP}$ and $B = \mathrm{SAT}$, we obtain a pseudorandom generator G computable in E^{NP} which is secure against SAT and has logarithmic seed length. If follows that $\mathsf{ZPP}^{\mathsf{NP}} = \mathsf{P}^{\mathsf{NP}}$. To finish the proof, it suffices to combine this with the Bshouty et al. [BCG+96] result that the polynomial-time hierarchy collapses to $\mathsf{ZPP}^{\mathsf{NP}}$ if NP has polynomial size circuits. □

5.3. Rigid matrices. Several researchers have studied the problem of finding explicit constructions of combinatorial objects that have been proven to exist nonconstructively (by using the probabilistic method, for example). In many cases, an explicit construction of some combinatorial object yields an interesting complexity theoretic result. One of the notable examples of this is the problem of matrix rigidity. The rigidity of a matrix M over a ring S, denoted $R_M^S(r)$, is the minimum number of entries of M that must be changed to reduce its rank to r or below. (An entry can be changed to any element of S.) Valiant [Val77] proved that an explicit construction of an infinite family of highly rigid matrices yields a circuit lower bound.

THEOREM 5.11 (see [Val77]). *Let $\epsilon, \delta > 0$ be constants. For any positive integer n, let M_n be an $n \times n$ matrix over a ring S_n. If $R_{M_n}^{S_n}(\epsilon n) \geqslant n^{1+\delta}$ for infinitely many values of n, then the linear transformations defined by the family M_n cannot be computed by linear size log-depth circuits consisting of gates computing binary linear operators on S_n.*

Valiant also proved that almost all matrices over an infinite field have rigidity

$(n-r)^2$, and almost all matrices over a fixed finite field have rigidity $\Omega((n-r)^2/\log n)$. The best known explicit constructions achieve rigidity $\Omega(n^2/r)$ over infinite fields [PV91, Raz] and $\Omega(\frac{n^2}{r}\log(\frac{n}{r}))$ [Fri90] over finite fields. These are not sufficient to obtain circuit lower bounds using Theorem 5.11. Under a hardness assumption, our derandomization technique will give an explicit construction to which Theorem 5.11 applies—a family of matrices M_n over $S_n = \mathbb{Z}_{p(n)}[x]$ such that $R^{S_n}_{M_n}(r) \in \Omega((n-r)^2/\log n)$, where $p(n)$ is polynomially bounded.

We will use the following lemma, which follows directly from Valiant's paper [Val77].

LEMMA 5.12 (see [Val77]). *Let S be a ring with at least two elements, and let n be a positive integer. All but at most a $\frac{1}{n}$ fraction of the $n \times n$ matrices M over S satisfies*

$$R^S_M(r) \geqslant \frac{(n-r)^2 - 2n - 2\log n}{1 + 2\log n}. \tag{14}$$

We can use our technique to achieve the nonconstructive rigidity bounds by noticing that there exist polynomial size SAT-oracle circuits which can check if a matrix is rigid.

THEOREM 5.13. *If there exists a Boolean function $f \in \mathsf{E}$ such that $C^{\mathrm{SAT}}_f(n) = 2^{\Omega(n)}$, then, given integers $n \geqslant 1$ and $p \geqslant 2$, we can construct in time polynomial in $n + \log p$ a list of $n \times n$ matrices M over $S = \mathbb{Z}_p$, most of which satisfy* (14).

Proof. Let ρ be a string of length $n^2 \cdot \lceil \log p \rceil$. We will view ρ as the concatenation of n^2 blocks of $\lceil \log p \rceil$ bits each. Let $F(\langle 1^n, p\rangle, \rho)$ denote the matrix whose ijth entry is the $(n(i-1)+j)$th block of ρ interpreted as a number in binary and taken modulo p. We define the predicate $\pi(\langle 1^n, p\rangle, \rho)$ to be true if $M = F(\langle 1^n, p\rangle, \rho)$ satisfies (14). Note that the latter is a coNP predicate: If (14) is violated for some r, we can guess modified values for fewer than the right-hand side of (14) many entries of M and verify that the rank of the modified matrix over S is at most r. So we can decide π by one query to an oracle for SAT. Moreover, Lemma 5.12 states that, for most sequences ρ, the predicate holds. By applying Theorem 4.4 to (F, π), we obtain a pseudorandom generator which, on most seeds, outputs a matrix with the required rigidity property. Enumerating over all seeds gives us the desired list of matrices in polynomial time. □

Now we need to combine this list of matrices into a single matrix with similarly high rigidity. We can do so by switching to the ring of univariate polynomials over $\mathbb{Z}_p$.

LEMMA 5.14. *Given $n \times n$ matrices $M_0, M_1, \ldots, M_k$ over $\mathbb{Z}_p$, where p is a prime larger than k, we can construct in time polynomial in $n + k + \log p$ an $n \times n$ matrix N over $\mathbb{Z}_p[x]$ such that*

$$R^{\mathbb{Z}_p[x]}_N(r) \geqslant \max_{0 \leqslant i \leqslant k} R^{\mathbb{Z}_p}_{M_i}(r), \tag{15}$$

where the entries of N are polynomials of degree at most k.

Proof. Let $q_{ij}(x)$ be the polynomial of degree at most k such that $q_{ij}(\ell)$ equals the ijth entry of M_ℓ, $0 \leqslant \ell \leqslant k$; i.e., q_{ij} interpolates the ijth entries of all $k+1$ matrices. Note that the q_{ij}'s exist and that each coefficient can be computed in time polynomial in $k + \log p$. Let N be the matrix whose ijth entry is q_{ij}. We now argue that (15) holds.

Let $m = R^{\mathbb{Z}_p[x]}_N(r)$, and let N' be a matrix obtained by changing m entries of N such that the rank of N' is at most r. Then every $(r+t) \times (r+t)$ minor of N' has

determinant 0 for $0 < t \leqslant n - r$. Thus the determinant of any $(r + t) \times (r + t)$ minor of N' can be viewed as an identically 0 polynomial in its entries. Let $\phi_a(N')$ be the matrix obtained by substituting $a \in \mathbb{Z}_p$ for x in every entry of N'. (Recall that N' has entries from $\mathbb{Z}_p[x]$.) Every $(r+t) \times (r+t)$ minor of $\phi_a(N')$ has determinant 0 for $0 < t \leqslant n - r$. We conclude by noticing that changing m or fewer entries of $\phi_a(N)$ has reduced its rank to a value less than or equal to r. However, $\phi_a(N)$ equals M_a for $a \in \{0, 1, \ldots, k\}$. Therefore, $R^{\mathbb{Z}_p}_{M_i}(r) \leqslant m$ for any $0 \leqslant i \leqslant k$; i.e., (15) holds. □

Given this construction of rigid matrices, we can conclude the following relationship among circuit lower bounds.

THEOREM 5.15. *If there exists a function $f \in \mathsf{E}$ such that $C^{\mathrm{SAT}}_f(n) = 2^{\Omega(n)}$, then there exists a polynomially bounded function $p(n)$ and a polynomial-time computable family of matrices M_n, where M_n is an $n \times n$ matrix over $\mathbb{Z}_{p(n)}[x]$ such that the linear transformations defined by the family M_n cannot be computed by log-depth linear size circuits which have special gates that can compute binary linear operators over $\mathbb{Z}_{p(n)}[x]$.*

Proof. For any polynomial time computable function $p(n)$, Theorem 5.13 allows us to efficiently compute a list of $(n + \log p(n))^c$ matrices M over $\mathbb{Z}_{p(n)}$ for some constant c, most of which are rigid. Provided $p(n)$ is a prime satisfying

$$p(n) \geqslant (n + \log p(n))^c, \tag{16}$$

Lemma 5.14 efficiently combines them into a single matrix N over $\mathbb{Z}_{p(n)}[x]$ which satisfies a rigidity condition sufficient for Theorem 5.11. The smallest prime value for $p(n)$ that satisfies (16) is polynomially bounded in n, and we can compute it in time polynomial in n. □

We point out that the weaker assumption that there exists a function $f \in \mathsf{E}$ such that $C_{f(n)} = 2^{\Omega(n)}$ immediately implies (by padding) the existence of a function in P with superpolynomial circuit complexity, regardless of the depth restrictions. Viewed that way, Theorem 5.15 uses a stronger hypothesis to produce a weaker conclusion, namely, the existence of a function in P that requires superpolynomial size but only for circuits of restricted depth. However, Theorem 5.15 adds credibility to Valiant's program of exhibiting an explicit function that cannot be computed by linear size log-depth circuits using matrix rigidity. Theorem 5.15 shows that, under a reasonable assumption, Valiant's program will work.

5.4. Universal traversal sequences. Universal traversal sequences, introduced by Cook, form another example where explicit constructions have important complexity theoretic implications. A universal traversal sequence for size n is a sequence σ of labels from $\{1, 2, \ldots, n-1\}$ such that, for any undirected connected graph G with n vertices in which the incident edges at every vertex have been assigned distinct labels from $\{1, 2, \ldots, n-1\}$, the following process always visits every vertex of G: Pick an arbitrary start vertex, and, in subsequent steps, go along the edge with the label matching the next symbol of σ; in case of no match, stay put during that step and continue with the next symbol of σ.

If we can construct universal traversal sequences in logspace, then we can solve undirected graph connectivity in logspace, and symmetric logspace equals logspace [LP82]. However, we do not know how to generate universal traversal sequences in logspace or even in polynomial time. Aleliunas et al. [AKL+79] showed that most sequences of length $O(n^3)$ over $\{1, 2, \ldots, n-1\}$ are universal traversal sequences for size n, but, as of now, the best explicit construction, due to Nisan [Nis92], yields universal traversal sequences of length $n^{O(\log n)}$. We give evidence supporting the

belief that explicit universal traversal sequences of polynomial size can be generated efficiently.

A straightforward application of our technique would yield a polynomial-time construction under the assumption that E requires exponential size SAT-oracle circuits. Since being a universal traversal sequence is a coNP predicate, Theorem 4.4 applied to the Aleliunas et al. process and oracle $B = \text{SAT}$ would efficiently generate under that hypothesis a collection of sequences, most of which are universal traversal sequences. Concatenating all of them would yield the desired universal traversal sequence of polynomial size. However, we can do better and dispense with the oracle B.

THEOREM 5.16. *If there is a Boolean function $f \in E$ such that $C_f(n) = 2^{\Omega(n)}$, then we can construct universal traversal sequences in polynomial time.*

Proof. Let G encode a graph with n vertices with edge labels as above. For any sequence σ over $\{1, 2, \ldots, n-1\}$, let $\tau(G, \sigma)$ indicate whether, for every vertex v of G, the walk in G starting from v and dictated by σ visits every vertex of G. Let $F(G, \rho)$ denote the sequence of length $c \cdot n^3$ over $\{1, 2, \ldots, n-1\}$ (where c is some sufficiently large constant) as specified by the successive bits of ρ. Let $\pi(G, \rho)$ equal $\tau(G, F(G, \rho))$. Note that π is a P predicate, so $B = \varnothing$ can efficiently check the randomized process (F, π). Since every universal traversal sequence σ satisfies $\tau(G, \sigma)$, the result of Aleliunas et al. [AKL+79] shows that for any graph G, $\pi(G, \rho)$ holds for most ρ. Therefore, Theorem 4.4 allows us to generate in polynomial time a collection of sequences σ, most of which satisfy $\tau(G, \sigma)$. Their concatenation forms a single sequence σ' satisfying $\tau(G, \sigma')$. Since the σ's are independent of G, so is their concatenation σ'. Hence we have constructed a sequence σ' which satisfies $\tau(G, \sigma')$ for every edge labeled graph G with n vertices; i.e., we have found a universal traversal sequence σ'. □

Under the assumption that *linear space* requires exponential size circuits, we can actually construct universal traversal sequences in *logspace*. In fact, that assumption allows us to build logspace computable pseudorandom generators for logspace and hence to derandomize BPL, the class of languages accepted by logspace randomized Turing machines with bounded two-sided error.

THEOREM 5.17. *If there is a Boolean function $f \in \text{DSPACE}[n]$ such that $C_f(n) = 2^{\Omega(n)}$, then $\text{BPL} = \text{L}$.*

Along the lines of Babai, Nisan, and Szegedy [BNS92], the pseudorandom generators behind Theorem 5.17 let us conclude the following corollary.

COROLLARY 5.18. *If there is a Boolean function $f \in \text{DSPACE}[n]$ such that $C_f(n) = 2^{\Omega(n)}$, then we can construct universal traversal sequences in logspace.*

Remark. We can actually relax the hypothesis in both Theorem 5.17 and Corollary 5.18 to the existence of a Boolean function in linear space that requires *branching programs* of size $2^{\Omega(n)}$: *If there is a Boolean function $f \in \text{DSPACE}[n]$ that requires branching programs of size $2^{\Omega(n)}$, then $\text{BPL} = \text{L}$, and we can construct universal traversal sequences in logspace.*

This stronger statement forms a more natural space-bounded analogue of the Impagliazzo–Wigderson [IW97] result than Theorem 5.17. The proof of the statement goes along the lines of the proof of Theorem 3.4. However, it needs a more space-efficient transformation of worst-case hardness into average-case hardness than the one by Sudan, Trevisan, and Vadhan [STV01] which we used in the proof of Theorem 3.4. The original construction by Impagliazzo and Wigderson [IW97], building on previous work by Babai et al. [BFNW93] and by Impagliazzo [Imp95], would do. Instead of going through the details of the latter constructions, we provide the details

for the weaker statement given in Theorem 5.17, for which the Sudan, Trevisan, and Vadhan construction used in our proof of Theorem 3.4 works fine.

5.5. Proof of Theorem 5.17. In order to establish Theorem 5.17, it suffices to show the following: If the function f in Theorem 3.4 is in linear space and $\ell(n) = \Theta(\log n)$, then we can make the pseudorandom generator G provided by Theorem 3.4 computable in linear space. This is because the seed length $s(n)$ of G in Theorem 3.4 is logarithmic for logarithmic $\ell(n)$.

The proof of Theorem 3.4 consists of two steps:

(i) Converting the worst-case hard function f into an average-case hard function g using Theorem 3.3.

(ii) Applying Theorem 3.2 to the average-case hard function g to obtain the pseudorandom generator G.

First, we argue that, if the function f in Theorem 3.3 is in linear space, then so is the function g. This is because the code $\mathcal{C}_{N,\epsilon}$ in Theorem 3.15 is actually computable in space $O(\log N + \log 1/\epsilon)$, as follows from a straightforward analysis of the paper by Sudan, Trevisan, and Vadhan [STV01].

We next apply Theorem 3.2 to the function g in order to obtain the pseudorandom generator G. Computing the ith bit of $G(\sigma)$ for $\sigma \in \{0,1\}^{s(n)}$ amounts to computing the set S_i of a $(\log n, \ell)$ design $\mathcal{S} = (S_1, S_2, \ldots, S_n)$ over $\{1, 2, \ldots, s(n)\}$ and evaluating g on input $\sigma|_{S_i}$. The latter can be done in space $O(s(n))$ since g is in linear space.

The greedy design construction of Lemma 3.12 requires too much space, namely, $\Theta(n)$. However, the next lemma describes a different approach, which needs only $O(s(n)) = O(\log n)$ space. The construction goes along the lines of Impagliazzo and Wigderson [IW97]. Allender informed us that Wigderson showed him the same construction [ARZ99]. For any $\ell(n) = \Theta(\log n)$, it yields a $(\log n, \ell(n))$ design of size n^δ over $\{1, 2, \ldots, s(n)\}$ for some positive δ and $s(n) = O(\log n)$. Note that this design is smaller than the one obtained through the greedy construction, which yields $\delta = 1$. However, designs of the former type are good enough for Theorem 3.13 (with appropriately modified parameters) and Theorem 3.2 to carry through. So, the next lemma finishes the proof of Theorem 5.17.

LEMMA 5.19. *For any positive constant c, there are positive constants α and γ such that we can generate an $(\alpha s, c\alpha s)$ design of size $2^{\gamma s}$ over $\{1, 2, \ldots, s\}$ in space $O(s)$.*

Proof. We will show that the following process has a positive probability of generating an (m, ℓ) design of size $k \doteq 2^{\gamma s}$ over $\Omega = \{1, 2, \ldots, s\}$ for sufficiently small positive constants α and γ, where $m \doteq \alpha s$ and $\ell \doteq \beta s$ with $\beta \doteq c\alpha$: Pick k subsets of Ω of size ℓ in a pairwise independent way such that each of the $\binom{s}{\ell}$ subsets has about the same probability of being selected.

More precisely, we will do the following. We choose an integer $a \geqslant 0$ such that $2^a \leqslant k^3 \cdot \binom{s}{\ell} < 2^{a+1}$, pick k numbers $r_i \in \{0, 1, \ldots, 2^a - 1\}$ at random in a pairwise independent way, and set S_i to be the $[(r_i \bmod \binom{s}{\ell}) + 1]$st subset of Ω of size ℓ (say, using the lexicographical order). Known constructions of such sample spaces [CG89] need only $O(a)$ random bits and can generate the samples from the random bits in space $O(a)$. Moreover, checking whether a given sample $S_1, \ldots, S_k$ forms an (m, ℓ) design can be done within the same space bounds. Therefore, we can cycle through all possible random bit sequences, check the corresponding candidate design, and output the first valid one. This process runs in space $O(a) = O(s)$ and succeeds provided the probability of picking a valid design this way is positive. The remainder of the proof will argue the latter.

CLAIM 5.20. *Let S, S' be subsets of Ω of size ℓ chosen independently and uniformly at random. Then, for some constants d_β and $d' > 0$,*

$$\Pr\left[\,|S \cap S'| \geqslant 2\beta^2 s\,\right] \leqslant d_\beta \cdot s \cdot \exp(-d'\beta^2 s).$$

Proof of Claim 5.20. First, consider a subset S of Ω obtained by doing the following independently for every $i \in \Omega$: Put i in S with probability β. Similarly, and independently, construct S'. Then, by Chernoff's bounds [MR97, Theorem 4.1],

$$\Pr\left[\,|S \cap S'| \geqslant 2\beta^2 s\,\right] \leqslant \exp(-d'\beta^2 s)$$

for some constant $d' > 0$. Stirling's formula yields that

$$\begin{aligned}\Pr\left[\,|S| = \beta s\,\right] &= \binom{s}{\beta s} \cdot \beta^{\beta s}(1-\beta)^{s-\beta s} \\ &\sim \frac{1}{\sqrt{2\pi\beta(1-\beta)}} \cdot \frac{1}{\sqrt{s}}.\end{aligned}$$

So

$$\begin{aligned}\Pr\left[\,|S \cap S'| \geqslant 2\beta^2 s \;\middle|\; |S| = |S'| = \beta s\right] &\leqslant \Pr\left[\,|S \cap S'| \geqslant 2\beta^2 s\,\right] \cdot \Pr\left[\,|S| = |S'| = \beta s\,\right] \\ &\leqslant d_\beta \cdot s \cdot \exp(-d'\beta^2 s)\end{aligned}$$

for some constant d_β depending on β. Since the above distribution of S and S' conditioned on $|S| = |S'| = \beta s$ coincides with the uniform distribution of the statement of the claim, this finishes the proof of the claim. □

There is a constant $d < 2k^3$ such that, for any $1 \leqslant i \leqslant k$ and any $T \subseteq \Omega$ with $|T| = \ell$,

$$\frac{d}{2^a} \leqslant \Pr[S_i = T] \leqslant \frac{d+1}{2^a}.$$

Because of the pairwise independence, it follows that, for any $1 \leqslant i < j \leqslant k$, the distribution of (S_i, S_j) differs from uniform by at most

$$\begin{aligned}\binom{s}{\ell}^2 \cdot \left[\left(\frac{d+1}{2^a}\right)^2 - \left(\frac{d}{2^a}\right)^2\right] &= (2d+1)\left(\frac{\binom{s}{\ell}}{2^a}\right)^2 \\ &< (2c+1)\left(\frac{2}{k^3}\right)^2 \\ &< \frac{4(4k^3+1)}{k^6} \\ &< \frac{17}{k^3}\end{aligned}$$

in L_1-norm. Therefore, by Claim 5.20,

$$\Pr\left[\,|S_i \cap S_j| \geqslant 2\beta^2 s\,\right] \leqslant \frac{17}{k^3} + d_\beta \cdot s \cdot \exp(-d'\beta^2 s).$$

Recall that $m \doteq \alpha s$, $\ell \doteq \beta s$, and $\beta \doteq c\alpha$. Hence, for $\alpha \leqslant 1/(2c^2)$,

$$\Pr[S_1, S_2, \ldots, S_k \text{ is not an } (m,\ell) \text{ design}] \leqslant \binom{k}{2} \cdot \left(\frac{17}{k^3} + d_\beta \cdot s \cdot \exp(-d'\beta^2 s)\right). \tag{17}$$

The right-hand side of (17) approaches 0 for $k = 2^{\gamma s}$ provided $0 < \gamma < \frac{d'\beta^2}{\ln 4} = \frac{d'c^2\alpha^2}{\ln 4}$. This finishes the proof of the lemma. □

6. Recent progress and further research. In this paper, we have demonstrated the power of relativization in the area of derandomization. We gave several striking examples, most notably Arthur-Merlin games. New applications have been discovered recently, and we are convinced that even more will follow.

Critical to our argument is the idea that the analyses of the Nisan–Wigderson and other generators work for a large class of predicates and statistical tests—whenever the generator fails a statistical test, the computation of the predicate on which the generator is based reduces to the computation of the test. This idea lies at the heart of the breakthrough extractor constructions of Trevisan [Tre99]. In fact, following Trevisan [Tre99], Miltersen [Mil98] explicitly describes the role of relativization there. He argues that every relativizable construction of a pseudorandom generator out of a hard Boolean function yields an extractor. We showed that the Impagliazzo–Wigderson construction relativizes. Applying Miltersen's argument to the latter yields Trevisan's extractors.

Another application of relativization along the lines of our work is the following: Nisan and Wigderson [NW94] observed that, if E has circuits of size $2^{o(n)}$, then, for every function $t(n) = 2^{\Omega(n)}$ and every positive constant ϵ,

$$\mathsf{DTIME}[t] \subseteq \mathsf{DSPACE}[t^{\epsilon}]. \tag{18}$$

They concluded from their main results that every language in BPP can be decided in deterministic polynomial time for infinitely many input lengths unless (18) holds. Lu [Lu00] noticed that the inclusion (18) even follows if E has B-oracle circuits of size $2^{o(n)}$ for some oracle B that is decidable in linear space. Therefore, by setting $B = \mathrm{SAT}$, the weak version of Theorem 3.10 implies that every language in AM has polynomial size proofs for infinitely many input lengths unless (18) holds [Lu00].

A different line of further research consists of relaxing the hardness condition for derandomizing Arthur-Merlin games. In particular, we would like to replace the SAT-oracle circuit model by the weaker nondeterministic circuit model in Theorem 3.8. Miltersen and Vinodchandran recently managed to do this for the special case of Theorem 3.10. They showed that, if $\mathsf{NE} \cap \mathsf{coNE}$ requires nondeterministic circuits of size $2^{\Omega(n)}$, then AM collapses to NP. They use a different technique, suited for one-sided error classes, and exploit the fact that all languages in AM possess a one-sided error-bounded round Arthur-Merlin game. More specifically, they prove that, if there exists a Boolean function $f \in \mathsf{E}^{\mathcal{A}}$ that requires nondeterministic B-oracle circuits of size $2^{\Omega(n)}$, then there exists a hitting set generator computable in $\mathsf{E}^{\mathcal{A}}$ with logarithmic seed length that works for conondeterministic B-oracle circuits. Instantiating this statement with $\mathcal{A} = \mathsf{NP} \cap \mathsf{coNP}$ and $B = \varnothing$ yields their main result. In a similar way, the SAT-oracle circuits in section 5.3 on rigid matrices can be replaced by nondeterministic circuits [MV99], and the TQBF_2-oracle circuits in section 5.2 on exact learning can be replaced by nondeterministic SAT-oracle circuits.

Due to its one-sided character, the technique of Miltersen and Vinodchandran does not apply to our results in section 5.1 on derandomizing the Valiant–Vazirani hashing procedure. It also only works at the "high-end" of the derandomization spectrum, i.e., under strong hardness assumptions, as in the last lines of Tables 1 and 2. The weakest hardness assumption Miltersen and Vinodchandran can handle is the existence of a Boolean function in $\mathsf{NEXP} \cap \mathsf{coNEXP}$ that requires nondeterministic circuits of size $2^{n^{\alpha}}$ for some constant $\alpha > \frac{1}{2}$. (Then they obtain quasi-polynomial size proofs for any language in AM, as in Theorem 3.9.) In particular, it remains open whether the existence of a Boolean function $f \in \mathsf{NEXP} \cap \mathsf{coNEXP}$ that requires

superpolynomial size nondeterministic circuits implies that graph nonisomorphism has subexponential size proofs. We present this as an open problem for further research.

Acknowledgments. We want to express our sincere gratitude to Oded Goldreich, Madhu Sudan, Luca Trevisan, and Salil Vadhan for their encouragement and collaboration. We would also like to thank Rudi Seitz for his contribution to section 5.4; Theorem 5.16 was proved jointly with him.

The first author wishes to thank Dan Spielman and Salil Vadhan for their support on all aspects of this paper. The first author also wishes to thank Henry Cohn, Steven Rudich, and Avi Wigderson for important conversations.

The second author wishes to thank his adviser Lance Fortnow for the guidance, suggestions, and support he received, and for the statement of Corollary 5.10, in particular. He also wants to thank the following people for stimulating and helpful discussions about issues related to this paper: Eric Allender, Laci Babai, Behfar Bastani, Harry Buhrman, Valentine Kabanets, Peter Bro Miltersen, Mike Saks, Madhu Sudan, Avi Wigderson, and Shiyu Zhou. The second author is grateful to Eric Allender, Peter Bro Miltersen, Valentine Kabanets, and Salil Vadhan for the comments he received on an earlier version of the paper [vM98].

The authors would also like to thank the journal referees for their helpful suggestions and comments.

REFERENCES

[Adl78] L. Adleman, *Two theorems on random polynomial time*, in Proceedings of the 19th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1978, pp. 75–83.

[AK97] V. Arvind and J. Köbler, *On pseudorandomness and resource-bounded measure*, in Proceedings of the 17th Conference on the Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1346, Springer-Verlag, New York, 1997, pp. 235–249.

[AKL+79] R. Aleliunas, R. Karp, R. Lipton, L. Lovász, and C. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in Proceedings of the 20th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1979, pp. 218–223.

[Ang88] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou, *Isolation, matching, and counting: Uniform and nonuniform upper bounds*, J. Comput. System Sci., 59 (1999), pp. 164–181.

[Bab85] L. Babai, *Trading group theory for randomness*, in Proceedings of the 17th ACM Symposium on Theory of Computing, ACM, New York, 1985, pp. 421–429.

[BCG+96] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.

[BDG95] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, Texts Theoret. Comput. Sci. EATCS Ser. 11, Springer-Verlag, Berlin, 1995.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.

[BKL83] L. Babai, W. Kantor, and E. Luks, *Computational complexity and the classification of finite simple groups*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1983, pp. 162–171.

[BL83] L. Babai and E. Luks, *Canonical labeling of graphs*, in Proceedings of the 15th ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 171–183.

[BM84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[BM88] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[BNS92] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[CG89] B. Chor and O. Goldreich, *On the power of two-point based sampling*, J. Complexity, 5 (1989), pp. 96–106.

[Coo71] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the 3rd ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[Coo88] S. Cook, *Short propositional formulas represent nondeterministic computations*, Inform. Process. Lett., 26 (1988), pp. 269–270.

[CRT98] A. Clementi, J. Rolim, and L. Trevisan, *Recent advances towards proving $P = BPP$*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 64 (1998), pp. 96–103.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.

[FGM+89] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, *On completeness and soundness in interactive proof systems*, in Randomness and Computation, S. Micali, ed., Advances in Computing Research 5, JAI Press, Greenwich, CT, 1989, pp. 429–442.

[For97] L. Fortnow, *Counting complexity*, in Complexity Theory Retrospective II, L. Hemaspaandra and A. Selman, eds., Springer-Verlag, New York, 1997, pp. 81–107.

[Fri90] J. Friedman, *A Note on Matrix Rigidity*, Tech. report TR-308-91, Department of Computer Science, Princeton University, Princeton, NJ, 1990.

[GM84] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.

[GS89] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Randomness and Computation, S. Micali, ed., Advances in Computing Research 5, JAI Press, Greenwich, CT, 1989, pp. 73–90.

[Imp95] R. Impagliazzo, *Hard-core distributions for somewhat hard problems*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 538–545.

[IW97] R. Impagliazzo and A. Wigderson, *P=BPP unless E has sub-exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 220–229.

[KL82] R. Karp and R. Lipton, *Turing machines that take advice*, Enseign. Math. (2), 28 (1982), pp. 191–209. A preliminary version appeared in STOC 1980.

[KvM99] A. Klivans and D. van Melkebeek, *Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 659–667.

[Lit88] N. Littlestone, *Learning when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1988), pp. 285–318.

[LP82] H. Lewis and C. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[Lu00] C.-J. Lu, *Derandomizing Arthur-Merlin games under uniform assumptions*, in Algorithms and Computation (Taipei, 2000), Lecture Notes in Comput. Sci. 1969, Springer-Verlag, Berlin, 2000, pp. 302–312.

[Mil98] P. B. Miltersen, *Relativizable Pseudorandom Generators and Extractors*, Comment to ECCC Tech. report TR98-055, 1998.

[MR97] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1997.

[MV99] P. B. Miltersen and N. Vinodchandran, *Derandomizing Arthur-Merlin games using hitting sets*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 71–80.

[Nis92] N. Nisan, *Pseudorandom generators for space-bounded computation*, Combinatorica, 12 (1992), pp. 449–461.

[NT99] N. Nisan and A. Ta-Shma, *Extracting randomness: A survey and new constructions*, J. Comput. System Sci., 58 (1999), pp. 148–173.

[NW94] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[Pap94] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.

[PV91] P. Pudlák and Z. Vavřín, *Computation of rigidity of order n^2/r for one simple matrix*, Comment. Math. Univ. Carolin., 32 (1991), pp. 213–218.

[PW90] L. Pitt and M. Warmuth, *Prediction preserving reducibility*, J. Comput. System Sci., 41 (1990), pp. 430–467.

[PZ83] C. Papadimitriou and S. Zachos, *Two remarks on the power of counting*, in Proceedings of the 6th GI Conference on Theoretical Computer Science, Lecture Notes in Comput. Sci. 145, Springer-Verlag, New York, 1983, pp. 269–276.

[Raz] A. Razborov, *On Rigid Matrices*, manuscript, 1989 (in Russian).

[Rud97] S. Rudich, *Super-bits, demi-bits, and $N\tilde{P}$/qpoly-natural proofs*, in Randomization and Approximation Techniques in Computer Science (Bologna, 1997), Lecture Notes in Comput. Sci. 1269, Springer-Verlag, Berlin, 1997, pp. 85–93.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma*, J. Comput. System Sci., 62 (2001), pp. 236–266.

[TO92] S. Toda and M. Ogiwara, *Counting classes are at least as hard as the polynomial-time hierarchy*, SIAM J. Comput., 21 (1992), pp. 316–328.

[Tod91] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–877.

[Tre99] L. Trevisan, *Construction of extractors using pseudo-random generators*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 141–148.

[Val77] L. Valiant, *Graph-theoretic arguments in low-level complexity*, in Proceedings of the 6th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 53, Springer-Verlag, Berlin, 1977, pp. 162–176.

[vM98] D. van Melkebeek, *Derandomizing Arthur-Merlin Games*, Tech. report TR-98-08, Department of Computer Science, The University of Chicago, Chicago, IL, 1998.

[VV86] L. Valiant and V. Vazirani, *NP is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.

[Yao82] A. Yao, *Theory and applications of trapdoor functions*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1527–1541

ON COUNTING INDEPENDENT SETS IN SPARSE GRAPHS*

MARTIN DYER†, ALAN FRIEZE‡, AND MARK JERRUM§

Abstract. We prove two results concerning approximate counting of independent sets in graphs with constant maximum degree Δ. The first implies that the Markov chain Monte Carlo technique is likely to fail if $\Delta \geq 6$. The second shows that no fully polynomial randomized approximation scheme can exist for $\Delta \geq 25$, unless RP = NP.

Key words. Primary, 68Q17; Secondary, 05C69, 60J10, 68E10, 68Q25, 68W40

AMS subject classifications. approximation algorithms, computational complexity, independent sets, mixing times of Markov chains, randomized algorithms

PII. S0097539701383844

1. Introduction.

Counting independent sets in graphs is one of several combinatorial counting problems which have received recent attention. The problem is known to be #P-complete, even for low-degree graphs [5]. On the other hand, it has been shown that, for graphs of maximum degree $\Delta = 4$, randomized approximate counting is possible [9, 5]. This success has been achieved using the *Markov chain Monte Carlo* method [8] to construct a *fully polynomial randomized approximation scheme (fpras)*. This has led to a natural question of how far this success might extend.

Here we consider in more detail this question of counting independent sets in graphs with constant maximum degree. We prove two results. The first, in section 2, shows that the Monte Carlo Markov chain method is likely to fail for graphs with $\Delta = 6$. This leaves open only the case $\Delta = 5$.

Our second result gives an explicit value of Δ above which approximate counting, using any kind of polynomial-time algorithm, is impossible unless RP = NP. The bound we obtain is $\Delta = 25$, though we suspect that the true value is in single figures, probably 6.

We note that Berman and Karpinski [2] have recently given new explicit bounds for the approximation ratio for the maximum independent set and other problems in low-degree graphs. These directly imply an inapproximability result for counting. (See Luby and Vigoda [9], specifically the proof of their Theorem 4.) However, the bound on Δ obtained this way is larger than ours by at least two orders of magnitude.

The questions we address in this article could also be studied in a more general setting in which vertices included in an independent set have weights or "fugacities" other than 1. In this setting, the weight of an independent set of size k is deemed to be λ^k for some constant k, and the goal is to compute the sum of the weights of all independent sets. One could then ask, for each Δ, at what exact λ an fpras ceases to

*Received by the editors January 20, 2001; accepted for publication (in revised form) March 15, 2002; published electronically August 5, 2002. A preliminary version of this article appeared in *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, New York, 1999, pp. 210–217.

http://www.siam.org/journals/sicomp/31-5/38384.html

†School of Computer Studies, University of Leeds, LS2 9JT, United Kingdom (dyer@scs.leeds.ac.uk). This author's research was supported by EC ESPRIT Working Group RAND2.

‡Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA 15213 (alan@random.math.cmu.edu). This author's research was supported by NSF grant CCR 9520974.

§Department of Computer Science, University of Edinburgh, EH9 3JZ, United Kingdom (mrj@dcs.ed.ac.uk).

exist (assuming such a λ exists). This question is a more precise version of the one we ask: for $\lambda = 1$, what is the largest Δ for which an fpras exists?

A reasonable guess is that the critical λ just identified is greater than 1 when $\Delta \leq 5$, and less than 1 when $\Delta \geq 6$. One might even rashly conjecture (though we shall not do so) that this critical λ is the same as that marking the boundary between unique and multiple Gibbs measures in the independent set (hard core gas) model in the regular infinite tree of degree Δ (the so-called Bethe lattice). Brightwell and Winkler have computed the fugacity λ at which multiple Gibbs measures appear in the Bethe lattice [3]. The only observation we offer here is that our results are consistent with the critical λ's being the same in both situations.

2. Markov chain Monte Carlo. For a graph G, let $\mathcal{I}(G)$ denote the collection of independent sets of G. Let $\mathcal{M}(G)$ be any Markov chain, asymptotically uniform on $\mathcal{I}(G)$, with transition matrix P_G. In this section, G will be a bipartite graph with a vertex bipartition into classes of equal size n. Let $b(n) \leq n$ be any function of n, and suppose we have $P_G(\sigma_1, \sigma_2) = 0$ whenever $|\sigma_1 \oplus \sigma_2| > b(n)$, where $\oplus$ denotes symmetric difference. We will say that $\mathcal{M}(G)$ is $b(n)$-*cautious*. Thus a $b(n)$-cautious chain is not permitted to change the status of more than $b(n)$ vertices in G at any step. Ideally, for ease of implementation, we would wish to have $b(n)$ a constant (as in [9, 5]). However, we will show that no $b(n)$-cautious chain on $\mathcal{I}(G)$ can mix rapidly unless $b(n) = \Omega(n)$. Thus any chain which does mix rapidly on $\mathcal{M}(G)$ must change the status of a sizable proportion of the vertices at each step.

Before stating our result, we need to formalize what we mean by mixing, rapid or otherwise. Let $\mathcal{M}$ be an ergodic Markov chain with state space Ω, distribution p_t at time t, and asymptotic distribution $p_\infty = \pi$. Let $x_0 \in \Omega$ be the initial state of $\mathcal{M}$, so that p_0 assigns unit mass to state x_0. Define the *mixing time* $\tau(x_0)$ *of* $\mathcal{M}$*, with initial state* $x_0 \in \Omega$, as the first t for which $\mathrm{d_{TV}}(p_t, \pi) \stackrel{\text{def}}{=} \frac{1}{2}||p_t - \pi||_1 \leq e^{-1}$; then define the *mixing time* τ as the maximum of $\tau(x_0)$ over choices of initial state x_0. We are able to show the following.

THEOREM 2.1. *Suppose* $\Delta \geq 6$ *and* $b(n) \leq 0.35n$*. Then there exists a constant* $\gamma > 0$ *and a bipartite graph* G_0*, regular of degree* Δ*, on* $n+n$ *vertices (more precisely a sequence of such graphs parameterized by* n*) with the following property: any* $b(n)$*-cautious Markov chain on* $\mathcal{I}(G_0)$ *has mixing time* $\tau = \Omega(e^{\gamma n})$*.*

Since, of course, there does exist a $2n$-cautious chain which mixes rapidly, our result cannot be strengthened much further. Although we do not identify a specific initial state x_0 satisfying $\tau(x_0) = \Omega(e^{\gamma n})$, our proof does provide a definite (and natural) initial distribution p_0 from which $\tau = \Omega(e^{\gamma n})$ steps are required to achieve $\mathrm{d_{TV}}(p_\tau, \pi) \leq e^{-1}$. The remainder of this section is devoted to the proof of Theorem 2.1.

The counterexample graph G_0 is just a random regular graph of degree Δ. Specifically, let $K_{n,n}$ denote the complete bipartite graph with vertex bipartition V_1, V_2, where $|V_1| = |V_2| = n$, and let G be the union of Δ perfect matchings selected independently and uniformly at random in $K_{n,n}$. (Since the perfect matchings are independent, they may well share some edges.) Denote by $\mathcal{G}(n, n, \Delta)$ the probability space of bipartite graphs G so defined. Where no confusion can arise, we simply write $\mathcal{G}$ for this class below. Note that $\mathcal{G}$ is a class of graphs with degree bound Δ. It is well known (see [1]) that, provided Δ is taken as constant, Δ-regular graphs occur in $\mathcal{G}(n, n, \Delta)$ with probability bounded away from 0. Since we prove that almost every graph $G \in \mathcal{G}(n, n, \Delta)$, for $\Delta \geq 6$, has the property we seek, it will follow that almost every Δ-regular graph (in the induced probability space) has the property too.

Let $0 < \alpha, \beta < 1$ be chosen values. For $G \in \mathcal{G}$, we consider the collection $\mathcal{I}_G(\alpha, \beta)$ of $\sigma \in \mathcal{I}(G)$ such that $|\sigma \cap V_1| = \alpha n$ and $|\sigma \cap V_2| = \beta n$. We will call $\sigma \in \mathcal{I}_G(\alpha, \beta)$ an (α, β)-*set*. Using linearity of expectation, we may easily compute the expected number $\mathcal{E}(\alpha, \beta) = \mathbf{E}(|\mathcal{I}_G(\alpha, \beta)|)$ of (α, β)-sets in G: it is just the number of ways of choosing an αn-subset from V_1 and a βn-subset from V_2, multiplied by the probability that all Δ perfect matchings avoid connecting the αn-subset to the βn-subset. Thus, using Stirling's formula,

$$\begin{aligned}\mathcal{E}(\alpha, \beta) &= \binom{n}{\alpha n}\binom{n}{\beta n}\left[\binom{(1-\beta)n}{\alpha n}\Big/\binom{n}{\alpha n}\right]^{\Delta} \\ &= \left(\frac{(1-\beta)^{(\Delta-1)(1-\beta)}(1-\alpha)^{(\Delta-1)(1-\alpha)}}{\alpha^{\alpha}\beta^{\beta}(1-\alpha-\beta)^{\Delta(1-\alpha-\beta)}}\right)^{n(1+o(1))} \\ &= e^{\varphi(\alpha,\beta)n(1+o(1))}, \qquad (1)\end{aligned}$$

where

$$\begin{aligned}\varphi(\alpha, \beta) = \varphi_{\Delta}(\alpha, \beta) &= -\alpha \ln \alpha - \beta \ln \beta - \Delta(1-\alpha-\beta)\ln(1-\alpha-\beta) \\ &\quad + (\Delta-1)\big((1-\alpha)\ln(1-\alpha) + (1-\beta)\ln(1-\beta)\big). \qquad (2)\end{aligned}$$

Mostly, Δ will be treated as a constant, and we shall suppress the subscript of φ except when we want to emphasize the dependence on Δ.

We shall treat φ as a function of real arguments α and β, even though a combinatorial interpretation is possible only when αn and βn are integers. Then φ is defined on the triangle

$$\mathcal{T} = \{(\alpha, \beta) : \alpha, \beta \geq 0 \text{ and } \alpha + \beta \leq 1\}$$

and is clearly symmetrical in α, β. (The function φ is defined by (2) on the interior of $\mathcal{T}$ and can be extended to the boundary by taking limits.) Moreover, the following facts are established in the appendix about the stationary points of φ on $\mathcal{T}$.

CLAIM 2.2.

(i) *The function φ has no local minima in the interior of $\mathcal{T}$, and no local maxima on the boundary of $\mathcal{T}$.*

(ii) *All local maxima of φ satisfy $\alpha + \beta + \Delta(\Delta-2)\alpha\beta \leq 1$.*

(iii) *If $\Delta \leq 5$, φ has only a single local maximum, which is on the line $\alpha = \beta$.*

(iv) *If $\Delta \geq 6$, φ has exactly two local maxima, symmetrical in α, β, and a single saddle-point on $\alpha = \beta$. The maximum with $\alpha < \beta$ occurs at $(\alpha, \beta) \approx (0.03546955, 0.40831988)$ when $\Delta = 6$ and at $(\alpha, \beta) \approx (0.01231507, 0.45973533)$ when $\Delta = 7$.*

Suppose, for the sake of discussion, we had the additional information that the number $|\mathcal{I}_G(\alpha, \beta)|$ of (α, β)-sets is reasonably well concentrated about its expectation $\mathcal{E}(\alpha, \beta)$. Then it would follow from (iii) and (iv) that a "typical" independent set in a random graph $G \in \mathcal{G}(n, n, \Delta)$ undergoes a dramatic change in passing from $\Delta = 5$ to $\Delta = 6$. For $\Delta \leq 5$, a typical independent set σ would be balanced, i.e., the sets $|\sigma \cap V_1|$ and $|\sigma \cap V_2|$ would be of nearly equal size, whereas for $\Delta \geq 6$ it would be unbalanced.

Unfortunately, we have not been able to prove a concentration result, and it is unclear whether such a result should be expected. Therefore, in examining the first (apparently) unbalanced case, $\Delta = 6$, we must make a slight detour. First, observe

that a knowledge of φ does at least provide an *upper* bound on $|\mathcal{I}_G(\alpha,\beta)|$ via Markov's inequality. In this way we can bound from above the number of (α,β)-sets that lie in the strip $|\alpha-\beta|\le\eta$ for some $\eta>0$. Then, we use a quite crude lower bound to show that the number of (α^*,β^*)-sets—for some chosen α^*,β^* with $\beta^*-\alpha^*>\eta$—is much greater that this.

We shall first deal with the boundary case $\Delta=6$. Once this has been done, it will be easy to dispense with the remaining cases, i.e., $\Delta\ge 7$, which are less finely balanced. So suppose for the time being that $\Delta=6$. Consider the function φ restricted to the region $\mathcal{D}=\mathcal{T}\cap\{(\alpha,\beta):|\alpha-\beta|\le\eta\}$, where $\eta=0.18$. Since the two local maxima of φ on $\mathcal{T}$ lie outside $\mathcal{D}$ (see Claim 2.2(iv)), it must be the case that the maxima of φ on $\mathcal{D}$ all lie on one or the other (and hence, by symmetry, both) of the lines $|\alpha-\beta|=\eta$. Numerically, the (unique) maximum with $\beta-\alpha=\eta$, achieved at $(\alpha,\beta)\approx(0.10021227,0.28021227)$, is a little less than $c=0.70824602$. (The uniqueness of the maximum may be verified by calculus; then the location of the maximum may be found to arbitrary precision by repeated evaluation of the derivative of $\varphi(\alpha,\alpha+0.18)$ with respect to α. Only simple function evaluations are required.)

Now define

$$\theta(\alpha)=-\alpha\ln\alpha-(1-\alpha)\ln(1-\alpha)+(\ln 2)(1-\Delta\alpha)$$

for $\Delta\alpha<1$. Then, for *any* graph $G\in\mathcal{G}$, the total number of independent sets σ with $|\sigma\cap V_1|=\alpha n$ is (crudely) at least

$$|\mathcal{I}_G(\alpha,*)|\ge e^{\theta(\alpha)n(1-o(1))}.$$

(Choose αn vertices from V_1; then choose any subset of vertices from the at least $(1-\Delta\alpha)n$ unblocked vertices in V_2.) Set $\alpha^*=0.015$. Then, by numerical computation, $\theta(\alpha^*)$ is a little greater than $0.70864644>c$. Thus, with high probability, the number of (α,β)-sets in $G\in\mathcal{G}$ lying in either connected component of $\mathcal{T}\setminus\mathcal{D}$ is greater than the number lying within $\mathcal{D}$ by an exponential factor, specifically $e^{\gamma n}$, where $\gamma=0.0004$. The graph G_0 of Theorem 2.1 is any graph $G_0\in\mathcal{G}$ that exhibits the exponential gap just described. (A randomly chosen graph will do with high probability.) The remainder of our argument concerns G_0.

Now consider a $0.35n$-cautious chain $\mathcal{M}(G_0)=\mathcal{M}_0$ on $\mathcal{I}(G_0)$. Let A comprise all (α,β)-sets with $\alpha\ge\beta$, i.e.,

$$A=\big\{\sigma\in\mathcal{I}(G_0):|\sigma\cap V_1|\ge|\sigma\cap V_2|\big\},$$

and assume without loss of generality that A is no larger than its complement $\overline{A}=\mathcal{I}\setminus A$. Denote by M the set of (α,β)-sets with $(\alpha,\beta)\in\mathcal{D}$. Since $\mathcal{M}_0$ is $0.35n$-cautious, it cannot make a transition from A to $\overline{A}$ except by using a state in M. Now, we have already seen that

$$|A|\ge e^{\gamma n}\,|M|. \tag{3}$$

Intuitively, since M is very small in relation to A, the mixing time of $\mathcal{M}_0$ must be very large. This intuition is captured in the following claim, which is implicit in a line of argument used by Jerrum [7].

Claim 2.3. *Let $\mathcal{M}$ be a Markov chain with state space Ω, transition matrix P, and stationary distribution π. Let $A\subset\Omega$ be a set of states such that $\pi(A)\le\frac{1}{2}$, and*

$M \subset \Omega$ be a set of states that form a "barrier" in the sense that $P_{ij} = 0$ whenever $i \in A \setminus M$ and $j \in \overline{A} \setminus M$. Then the mixing time of $\mathcal{M}$ is at least $\pi(A)/8\pi(M)$.

For completeness, a proof using "conductance" is provided in the appendix. Theorem 2.1, in the boundary case $\Delta = 6$, follows from Claim 2.3 and inequality (3) because the sets A and M that we defined earlier satisfy the conditions of the claim. Note that the proof of Claim 2.3 actually provides an explicit initial distribution p_0 from which the mixing time is large, namely, the uniform distribution on A.

Finally, suppose $\Delta \geq 7$. We shall see presently that

$$\varphi_\Delta(\alpha, \beta) < 0.6763 < \ln 2 \quad \text{for all } \Delta \geq 7 \text{ and } (\alpha, \beta) \in \mathcal{D}. \tag{4}$$

On the other hand, there are at least 2^n (α, β)-sets in either connected component of $\mathcal{T} \setminus \mathcal{D}$: this comes simply from considering independent sets with $\alpha = 0$ or $\beta = 0$. Once again, with high probability, the number of (α, β)-sets in $G \in \mathcal{G}$ lying in either connected component of $\mathcal{T} \setminus \mathcal{D}$ is greater than the number lying within $\mathcal{D}$ by an exponential factor, specifically $e^{\gamma n}$, where $\gamma = 0.015$. Theorem 2.1, in the general case $\Delta \geq 7$, follows as before.

It remains only to verify (4). By calculus, $\varphi_\Delta(\alpha, \beta)$ as a function of Δ is monotonically decreasing over the whole region $\mathcal{T}$; thus we need check only the case $\Delta = 7$. (The partial derivative $\partial \varphi_\Delta(\alpha, \beta)/\partial \Delta$ is a function of α and β only; it is zero on $\alpha = 0$ and monotonically decreasing as a function of β.) We now argue, as before, that the maxima of φ on $\mathcal{D}$ all lie on the lines $|\alpha - \beta| = 0.18$. (Here we again use Claim 2.2(iv).) Once again, by calculus, φ has a unique maximum on each of these lines, and direct calculation yields (4).

3. Hardness of approximate counting. The result of the previous section implies that the usual approach to approximating the number of independent sets in a low-degree graph must fail when $\Delta \geq 6$, at least in the worst case. Here we show that, if the degree bound is somewhat larger, then *any* approach to approximating the number of independent sets is doomed to failure, under a reasonable complexity assumption. Precisely, the remainder of this section is devoted to proving the following theorem.

THEOREM 3.1. *Unless* RP = NP*, there is no polynomial-time algorithm that estimates the* logarithm *of the number of independent sets in a Δ-regular graph ($\Delta \geq 25$) within relative error at most $\varepsilon = 10^{-4}$.*

We give a randomized reduction from the following problem E2LIN2, analyzed by Håstad. The input is a system $\mathcal{A}$ of m equations over $\mathbb{Z}_2$ in n variables $x_1, x_2, \ldots, x_n$, such that each equation has exactly two variables. (Thus each equation is of the form $x_i + x_j = 0$ or $x_i + x_j = 1$.) The objective is to find a maximum cardinality consistent subset of equations in $\mathcal{A}$, i.e., to assign values to the variables so as to maximize the number m_C of satisfied equations. Håstad [10] showed, using the powerful theory of probabilistically checkable proofs (PCPs), that it is NP-hard to estimate m_C within any constant factor smaller than $12/11$.[1] Therefore consider an instance $\mathcal{A}$ of E2LIN2, as above. We will construct (by a randomized algorithm) a graph $G = (V, E)$, regular of degree Δ. We then show that, if we can approximate the logarithm of the number of independent sets in G to within the required relative error, we can (with high probability) approximate the size of m_C in $\mathcal{A}$ to within a factor $12/11 - \varepsilon$. Theorem 3.1 will then follow.

[1] In other words, determining a number in the range $[(11/12+\varepsilon)m_C, m_C]$ is as hard as determining m_C exactly. Following convention, Håstad normalizes approximation ratios to be greater than 1, taking the reciprocal in the case of a maximization problem.

Let us write $[n] = \{1, 2, \ldots, n\}$. We construct the graph $G = G(\mathcal{A})$ as follows. We assume $m \geq n$; otherwise, $\mathcal{A}$ is decomposable or consistent. Let $M = m^6$ and, for each $i \in [n]$, let A_i be the multiset of equations containing x_i, with (multiset) cardinality d_i. We represent x_i by a regular bipartite graph H_i of degree $\delta = \Delta - 1$, with vertex partition (L_i, R_i) and edge set F_i. Here $L_i = \bigcup_{a \in A_i} L_{i,a}$, $R_i = \bigcup_{a \in A_i} R_{i,a}$, where the sets $L_{i,a}$, $R_{i,a}$ partition L_i and R_i, respectively, and for all i, a, $|L_{i,a}| = |R_{i,a}| = M$. Thus H_i is bipartite with both its vertex sets of size Md_i. Later, we will associate L_i with the assignment $x_i = 0$, and R_i with $x_i = 1$.

The graph $H_i = (L_i, R_i, F_i)$ will be sampled from $\mathcal{G}(Md_i, Md_i, \delta)$, where $\mathcal{G}$ is the class of random graphs defined in section 2. Just as in that section, and for the same reason, we are at liberty to reject graphs which are not δ-regular. Clearly, the property of being δ-regular can be checked in polynomial time.

The equations a of $\mathcal{A}$ determine the edges connecting the H_i in G, as follows. If a is the equation $x_i + x_j = 1$ ($x_i + x_j = 0$, resp.), we add an arbitrary perfect matching between $L_{i,a}$ and $L_{j,a}$ ($R_{j,a}$, resp.) and another between $R_{i,a}$ and $R_{j,a}$ ($L_{j,a}$, resp.). Thus G is a regular graph of degree Δ. We will show that approximating the logarithm of the number of independent sets in G to within a factor $(1 + 10^{-4})$ will allow us to approximate the E2LIN2 instance within the Håstad bound.

Before returning to the issue of approximation, we will need to establish some crucial properties of the "typical" independent set in G. For this purpose, let I be sampled uniformly from $\mathcal{I}(G)$, the set of all independent sets in G. First we show that I "occupies about half the available space" in each $L_{i,a}$ or $R_{i,a}$.

Let $\mathcal{L}_{i,a}$ be the set of vertices in $L_{i,a}$ with no neighbor in I, and let $\mathcal{L}_i = \bigcup_{a \in A_i} \mathcal{L}_{i,a}$.

LEMMA 3.2. *Suppose that I is sampled uniformly at random (u.a.r.) from $\mathcal{I}(G)$. Then, except for probability $e^{-\Omega(m^2)}$, either $|\mathcal{L}_{i,a}| < m^4$ or $|\mathcal{L}_{i,a}| = (2 \pm O(1/m)) \times |I \cap L_{i,a}|$.*

Proof. If we condition on $I \cap (V \setminus L_{i,a})$, then $I \cap L_{i,a}$ is a random subset of $\mathcal{L}_{i,a}$. If $|\mathcal{L}_{i,a}| \geq m^4$, then Chernoff's bound implies that

$$\mathbf{Pr}\left[|I \cap L_{i,a}| \notin \frac{1}{2}\Big(1 \pm \frac{1}{m}\Big)\,|\mathcal{L}_{i,a}|\right] \leq 2\exp\Big(-\frac{1}{3}m^2\Big),$$

from which the lemma follows. ∎

Clearly, we may define $\mathcal{R}_{i,a}$ and $\mathcal{R}_i$ symmetrically and prove an analogous result. It is also clear that we may claim Lemma 3.2 for all i, a simultaneously, since there are fewer than m^2 such pairs. Now imagine that we choose an independent set $I \in \mathcal{I}(G)$ u.a.r. in two steps: first the part of I that lies outside H_i, followed by the restriction of I to H_i. We now deduce from Lemma 3.2 that, with high probability, at least around half of L_i is "available" to I in the second step.

Let $\mathcal{L}'_i$ be the set of vertices in L_i with no neighbor in I outside of H_i.

LEMMA 3.3. *Suppose that I is sampled u.a.r. from $\mathcal{I}(G)$. Except for probability $e^{-\Omega(m^2)}$,*

$$|\mathcal{L}'_i| \geq \left(\frac{1}{2} - O\left(\frac{1}{m}\right)\right)|L_i|. \tag{5}$$

Proof. If $L_{i,a}$ is joined by a matching to $V_{j,a}$ ($V \in \{L, R\}$), then, from Lemma 3.2, $M \geq (2 - O(1/m))|I \cap V_{j,a}|$. Hence

$$\big|\{v \in L_{i,a} : \{v, w\} \in E \setminus F_i \text{ implies } w \notin I\}\big| \geq \left(\frac{1}{2} - O\left(\frac{1}{m}\right)\right)|L_{i,a}|.$$

Summing this over all $a \in A_i$ gives the lemma. ∎

Again, we may define $\mathcal{R}_i'$ and prove a corresponding result. We now show that for each i either $|\mathcal{L}_i|$ or $|\mathcal{R}_i|$ is "small." We will temporarily drop the suffix i and write H rather than H_i, etc. Let $N = |L| = dM \le m^7$, $a = |\mathcal{L}'|/N$, $b = |\mathcal{R}'|/N$. Write $\sigma = I \cap H$, where I is a uniformly chosen independent set in G. We will say that σ is an (α, β)-set if $|\sigma \cap L| = \alpha a N$, $|\sigma \cap R| = \beta b N$.

LEMMA 3.4. *Let $\delta \ge 24$. If I is a uniformly chosen independent set in G, then, except for probability $e^{-\Omega(m^2)}$,*

$$\min(|\mathcal{L}_i|, |\mathcal{R}_i|) \le \lambda N, \tag{6}$$

where $\lambda = 0.009$.

Proof. We focus attention on a particular H in G (corresponding to a particular variable in the E2LIN2 instance). Suppose that the whole of G aside from the edges within H has been fixed (i.e., the random choices have already been made), except that we have not chosen the edges of H itself. Ultimately, we want to argue about a random independent set I. However, for the time being, suppose that we simply fix the portion of I that lies outside of H; doing this fixes the sets $\mathcal{L}'$ and $\mathcal{R}'$ of vertices in H that have no neighbor in I. About I, we assume only that it satisfies inequality (5) of Lemma 3.3 so that $a \ge b \ge \frac{1}{2} - O(\frac{1}{m})$, where, without loss of generality, we have taken $a \ge b$.

We now reveal H and examine the number of extensions of I to H as a function of α and β. It is easy to see that there are at least 2^{aN} independent sets in H in total. We will show that, for α, β not satisfying the condition of the lemma, the number of (α, β)-sets is so much smaller than this that they appear with probability $e^{-\Omega(m^2)}$. It will be sufficient to show that the expected number of (α, β)-sets in such a case is $2^{aN - \Omega(m^2)}$, because Markov's inequality will then imply the required inequality for the actual number. Now the expected number of (α, β)-sets in H is

$$\begin{aligned}
\mathcal{E}(\alpha,\beta) &= \binom{aN}{\alpha a N}\binom{bN}{\beta b N}\left[\binom{(1-b\beta)N}{\alpha a N}\Big/\binom{N}{\alpha a N}\right]^{\delta} \\
&\le \binom{aN}{\alpha a N}\binom{bN}{\beta b N}\left[\frac{[(1-b\beta)N]^{\underline{\alpha a N}}}{(\alpha a N)!} \times \frac{(\alpha a N)!}{N^{\underline{\alpha a N}}}\right]^{\delta} \\
&\le \binom{aN}{\alpha a N}\binom{bN}{\beta b N}(1-b\beta)^{\alpha a \delta N} \\
&\le \left[\left(\alpha^{\alpha}(1-\alpha)^{(1-\alpha)}\right)^{-a}\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{-b} e^{-\alpha\beta a b \delta}\right]^{N(1+o(1))} \\
&= e^{\psi(\alpha,\beta)N(1+o(1))}, \qquad (7)
\end{aligned}$$

where an underlined superscript denotes "falling factorial power," and

$$\begin{aligned}
\psi(\alpha,\beta) = &-a(\alpha\ln\alpha + (1-\alpha)\ln(1-\alpha)) \\
&- b(\beta\ln\beta + (1-\beta)\ln(1-\beta)) - \alpha\beta a b\delta. \qquad (8)
\end{aligned}$$

Note that ψ is defined in the unit square $\mathcal{U} = \{(\alpha, \beta) : 0 \le \alpha, \beta \le 1\}$. As before, we shall treat α and β (and indeed a and b) as real variables, even though a combinatorial interpretation requires aN, bN, $\alpha a N$, and $\beta b N$ to be integers. The key property of ψ is captured in the following claim, whose proof can be found in the appendix.

CLAIM 3.5. *Let $\delta = 24$, $\eta > 0$ be sufficiently small, and suppose $\frac{1}{2} - \eta \le b \le a \le 1$. For any $(\alpha, \beta) \in \mathcal{U}$, the inequality $\psi(\alpha, \beta) \ge a \ln 2 - \eta$ entails $\min\{\alpha a, \beta b\} \le 0.004$.*

Recall the crude lower bound 2^{aN} on the total number of independent sets σ extending I to H. The claim tells us that only very unbalanced independent sets—those with either $|\sigma \cap L| \le 0.004$ or $|\sigma \cap R| \le 0.004$—make a significant contribution to that total. All of the above argument was for an independent set I that is fixed outside H, so we have not yet proved Lemma 3.4. Nevertheless, all the key calculations are out of the way, and we can complete the proof with a little algebra.

Let $\mathcal{I}$ be the set of all independent sets on $V(G) \setminus V(H)$. Let $\mathcal{I}_{\text{good}} \subseteq \mathcal{I}$ be the independent sets I that satisfy inequality (5) of Lemma 3.3, and $\mathcal{I}_{\text{bad}} = \mathcal{I} \setminus \mathcal{I}_{\text{good}}$. Let $N(I, H)$ be the number of independent sets in H consistent with I, and let $N^*(I, H)$ be the number of such that do not satisfy inequality (6) of Lemma 3.4. Denote by $\mathcal{H}$ the (multi)set of all possible choices for the graph H viewed as a disjoint union of δ perfect matchings. (Thus each possible graph H will occur with multiplicity μ, where μ is the number of 1-factorizations of H—i.e., decompositions into disjoint perfect matchings—of H. Note that our reduction requires us to select *uniformly* from $\mathcal{H}$.) For convenience, set $\varepsilon = e^{-\Omega(m^2)}$. We have shown in Lemma 3.3 that

$$\sum_{I \in \mathcal{I}_{\text{bad}}} N(I, H) \le \varepsilon \sum_{I \in \mathcal{I}} N(I, H) \qquad \text{for all } H \in \mathcal{H}. \tag{9}$$

(Note that the sum on the right-hand side is the total number of independent sets in G, while that on the left-hand side is the number violating inequality (5).) We will show below that a random H satisfies

$$\sum_{I \in \mathcal{I}_{\text{good}}} N^*(I, H) \le \varepsilon \sum_{I \in \mathcal{I}_{\text{good}}} N(I, H) \tag{10}$$

with high probability, specifically, with probability at least $1 - \varepsilon$. Putting (9) and (10) together, a random H satisfies

$$\begin{aligned} \frac{\sum_{I \in \mathcal{I}} N^*(I, H)}{\sum_{I \in \mathcal{I}} N(I, H)} &\le \frac{\varepsilon \sum_{I \in \mathcal{I}_{\text{good}}} N(I, H) + \sum_{I \in \mathcal{I}_{\text{bad}}} N(I, H)}{\sum_{I \in \mathcal{I}} N(I, H)} \\ &\le \varepsilon + \varepsilon = 2\varepsilon \end{aligned}$$

with high probability, which is what we require.

We now prove (10). Claim 3.5 taken in conjunction with Lemma 3.2 shows that

$$\frac{\sum_{H \in \mathcal{H}} N^*(I, H)}{|\mathcal{H}|} \le \varepsilon^2 \widehat{N}(I) \qquad (I \in \mathcal{I}_{\text{good}})$$

for some $\widehat{N}(I)$ satisfying $\widehat{N}(I) \le N(I, H)$ for all $H \in \mathcal{H}$. (Specifically, $\widehat{N} = 2^{aN}$ will do here.) Summing this over $I \in \mathcal{I}_{\text{good}}$ gives

$$\frac{1}{|\mathcal{H}|} \sum_{H \in \mathcal{H}} \sum_{I \in \mathcal{I}_{\text{good}}} N^*(I, H) \le \varepsilon^2 \sum_{I \in \mathcal{I}_{\text{good}}} \widehat{N}(I),$$

giving

$$\frac{1}{|\mathcal{H}|} \sum_{H \in \mathcal{H}} \frac{\sum_{I \in \mathcal{I}_{\text{good}}} N^*(I, H)}{\sum_{I \in \mathcal{I}_{\text{good}}} \widehat{N}(I)} \le \varepsilon^2,$$

which implies that

$$\frac{1}{|\mathcal{H}|}\sum_{H\in\mathcal{H}}\frac{\sum_{I\in\mathcal{I}_{\text{good}}}N^*(I,H)}{\sum_{I\in\mathcal{I}_{\text{good}}}N(I,H)}\le\varepsilon^2. \tag{11}$$

Let

$$\mathcal{H}^*=\left\{H\in\mathcal{H}:\sum_{I\in\mathcal{I}_{\text{good}}}N^*(I,H)\ge\varepsilon\sum_{I\in\mathcal{I}_{\text{good}}}N(I,H)\right\}.$$

Then, from (11),

$$\frac{1}{|\mathcal{H}|}\,\varepsilon\,|\mathcal{H}^*|\le\varepsilon^2,$$

so

$$\frac{|\mathcal{H}^*|}{|\mathcal{H}|}\le\varepsilon,$$

as is required to establish (10) and complete the proof. □

We now establish the relationship between the number of independent sets in G and the maximum size of a consistent subset of $\mathcal{A}$. Let $\mathcal{I}=\mathcal{I}(G)$. For $\sigma\in\mathcal{I}$ let $S_\sigma\subseteq[n]$ be defined by

$$S_\sigma=\{i:\ |L_i\cap\sigma|>|R_i\cap\sigma|,\ i\in[n]\}.$$

For $S\subseteq[n]$ let $\mathcal{I}_S=\{\sigma\in\mathcal{I}:\ S_\sigma=S\}$ and let $\mu_S=|\mathcal{I}_S|$. Recall that m is the number of equations in $\mathcal{A}$.

LEMMA 3.6. *For $S\subseteq[n]$ let $\theta(S)$ be the number of equations in $\mathcal{A}$ satisfied by the assignment $x_i=1$ $(i\in S)$, $x_i=0$ $(i\notin S)$. Then*

$$4^{M\theta(S)}3^{M(m-\theta(S))}\le\mu_S\le4^{M\theta(S)}3^{M(m-\theta(S))}2^{2\lambda mM}(1+o(1)), \tag{12}$$

where λ is as in Lemma 3.4.

Proof. Fix $S\subseteq[n]$, and for $\sigma\in\mathcal{I}_S$ let $J_\sigma=\sigma\cap\left(\bigcup_{i\in S}L_i\cup\bigcup_{i\notin S}R_i\right)$. Informally, J_σ restricts σ to the left or right of each subgraph H_i, according to which side contains the larger part of σ. Let

$$\hat{\mu}_S=|\{J_\sigma:\ \sigma\in\mathcal{I}_S\}|\le\mu_S.$$

We show that

$$\hat{\mu}_S=4^{M\theta(S)}3^{M(m-\theta(S))}. \tag{13}$$

This immediately proves the lower bound in (12). Furthermore, Lemma 3.4 implies that for a fixed value J of J_σ there are (up to a factor $(1+e^{-\Omega(m^2)})$) at most

$$\prod_{i\in[n]}2^{\lambda d_iM}=2^{\lambda M\sum_i d_i}=2^{2\lambda mM}$$

sets $\sigma\in\mathcal{I}_S$ with $J_\sigma=J$. The upper bound then follows.

To prove (13) we consider the number of possible choices for $J \cap L_{i,a}$, $J \cap R_{i,a}$, $J \cap L_{j,a}$, and $J \cap R_{j,a}$ for every equation $a : x_i + x_j = z_a$ ($z_a \in \{0,1\}$). For given S, let us define

$$X_{i,a} = \begin{cases} L_{i,a} & \text{if } i \in S; \\ R_{i,a} & \text{if } i \notin S. \end{cases}$$

Then there are two cases, determined by the status of a.

(1) Equation a is satisfied by the assignment derived from S. Then there are 2^M choices for each of $J \cap X_{i,a}$, $J \cap X_{j,a}$, giving 4^M in all.
(2) Equation a is not satisfied by the assignment derived from S. Then the subgraph of G induced by $X_{i,a} \cup X_{j,a}$ is a matching of size M and hence contains 3^M independent sets.

Multiplying the estimates from the two cases over all $a \in \mathcal{A}$ proves (13) and the lemma. □

We now proceed to the proof of Theorem 3.1. Let $Z_I = Z_I(G)$ denote the logarithm of the number of independent sets of $G(\mathcal{A})$. Let $Z_C = Z_C(\mathcal{A})$ denote the maximum number of consistent equations in $\mathcal{A}$.

Let Y_I be some estimate of Z_I satisfying $|Y_I/Z_I - 1| \le \varepsilon = 10^{-4}$. Using Y_I, we define

$$Y_C = \left(\frac{Y_I}{M} - m \ln 3\right) \frac{1.001}{\ln(4/3)}.$$

A simple calculation will then show that $1 \le Y_C/Z_C \le 12/11 - \varepsilon$, so that Y_C determines Z_C with sufficient accuracy to beat the approximability bound for E2LIN2.

From Lemma 3.6 we see that

$$Y_I \ge (1-\varepsilon)M(Z_C \ln(4/3) + m \ln 3).$$

Hence, since $Z_C \ge m/2$,

$$\frac{Y_C}{1.001} \ge (1-\varepsilon)Z_C - \frac{\varepsilon m \ln 3}{\ln(4/3)} \ge Z_C\left(1 - \frac{\varepsilon \ln 12}{\ln(4/3)}\right) \ge 0.9991 Z_C,$$

which implies that $Y_C \ge Z_C$. On the other hand, Lemma 3.6 also implies that

$$Y_I \le (1+\varepsilon)\big[M\big(Z_C \ln(4/3) + m \ln 3 + 2m\lambda \ln 2\big) + n \ln 2\big],$$

where $\lambda \le 0.009$. Hence

$$\begin{aligned} \frac{Y_C}{1.001} &\le (1+\varepsilon)Z_C + \frac{\varepsilon m \ln 3}{\ln(4/3)} + \frac{(1+\varepsilon)2m\lambda \ln 2}{\ln(4/3)} + \frac{(1+\varepsilon)\ln 2}{n \ln(4/3)} \\ &\le Z_C\left(1 + \varepsilon + \frac{2\varepsilon \ln 3}{\ln(4/3)} + \frac{4(\ln 2)(1+\varepsilon)\lambda}{\ln(4/3)} + O\left(\frac{1}{n^2}\right)\right) \\ &\le Z_C\left(1.0877 + O\left(\frac{1}{n^2}\right)\right), \end{aligned}$$

which implies that Y_C/Z_C is bounded away from $12/11$ for n large enough. Summarizing, the existence of a polynomial-time algorithm, meeting the specification in Theorem 3.1, for estimating the number of independent sets in a 25-regular graph

would entail the existence of a randomized (two-sided error) algorithm for approximating the solution to an E2LIN2 instance with relative error better than 12/11. (The algorithm is randomized because the reduction is too.) Because the latter problem is NP-hard, we could deduce that NP $\subseteq$ BPP. But this inclusion in turn implies that RP = NP (see Papadimitriou [11, Problem 11.5.18]). Thus we have established Theorem 3.1.

Appendix.

Proof of Claim 2.2. We start by computing partial derivatives of φ up to order two:

$$\frac{\partial\varphi}{\partial\alpha} = -\ln\alpha - (\Delta-1)\ln(1-\alpha) + \Delta\ln(1-\alpha-\beta), \tag{14}$$

$$\frac{\partial\varphi}{\partial\beta} = -\ln\beta - (\Delta-1)\ln(1-\beta) + \Delta\ln(1-\alpha-\beta), \tag{15}$$

$$\frac{\partial^2\varphi}{\partial\alpha^2} = -\frac{1}{\alpha} + \frac{\Delta-1}{1-\alpha} - \frac{\Delta}{1-\alpha-\beta}, \tag{16}$$

$$\frac{\partial^2\varphi}{\partial\beta^2} = -\frac{1}{\beta} + \frac{\Delta-1}{1-\beta} - \frac{\Delta}{1-\alpha-\beta}, \tag{17}$$

$$\frac{\partial^2\varphi}{\partial\alpha\,\partial\beta} = -\frac{\Delta}{1-\alpha-\beta}. \tag{18}$$

Parts (i)–(iv) of Claim 2.2 may then be verified as follows:

(i) From (16), it can easily be checked that $\partial^2\varphi/\partial\alpha^2 < 0$ on the interior of $\mathcal{T}$, and hence φ can have no interior local minima. On $\alpha = 0$, φ has a maximum at $\beta = \frac{1}{2}$ using (15), but then from (14) we find $\partial\varphi/\partial\alpha = +\infty$ at $\alpha = 0$, $\beta = \frac{1}{2}$. Similarly $\beta = 0$. On $\alpha + \beta = 1$, both $\partial\varphi/\partial\alpha, \partial\varphi/\partial\beta = -\infty$, so φ can have no maximum.

(ii) Since both $\partial^2\varphi/\partial\alpha^2, \partial^2\varphi/\partial\beta^2 < 0$, φ has a maximum if and only if the Hessian of φ has a positive determinant. The condition for this is $\alpha + \beta + \Delta(\Delta-2)\alpha\beta \le 1$, as may be checked from (16)–(18).

(iii) From (14) and (15), the conditions for a stationary point of φ may be written

$$\beta = f(\alpha), \quad \alpha = f(\beta),$$

where

$$f(x) = 1 - x - x^{1/\Delta}(1-x)^{1-1/\Delta} = (1-x)\left[1 - \left(\frac{x}{1-x}\right)^{1/\Delta}\right] \quad (0 \le x \le 1).$$

Thus, at any stationary point,

$$\alpha = f(f(\alpha)). \tag{19}$$

Clearly $f(x) \le 0$ for $x \ge \frac{1}{2}$, so $\alpha < \frac{1}{2}$ at any stationary point. Similarly $\beta < \frac{1}{2}$. To study the roots of (19), the change of variable $y = (\alpha/(1-\alpha))^{1/\Delta}$ proves to be convenient. With a little calculation we may express α, $f(\alpha)$, and $f(f(\alpha))$ in terms of y:

$$\alpha = \frac{y^\Delta}{1+y^\Delta}, \tag{20}$$

$$f(\alpha) = (1-\alpha)(1-y) = \frac{1-y}{1+y^\Delta},$$

and

$$\begin{aligned} f(f(\alpha)) &= (1 - f(\alpha)) - \big(f(\alpha)(1 - f(\alpha))^{\Delta-1}\big)^{1/\Delta} \\ &= \left(\alpha + \frac{y}{1+y^\Delta}\right) - \frac{\big((1-y)(y+y^\Delta)^{\Delta-1}\big)^{1/\Delta}}{1+y^\Delta} \\ &= \alpha + \frac{y}{1+y^\Delta}\left[1 - \left(\frac{(1-y)(1+y^{\Delta-1})^{\Delta-1}}{y}\right)^{1/\Delta}\right], \end{aligned}$$

and hence (19) is equivalent to

$$(1 + y^{\Delta-1})^{\Delta-1} = \frac{y}{1-y} \qquad (0 \le y < 1). \tag{21}$$

Note that the implicit mapping from α to y is a bijection, so we may legitimately study the solution set of (19) through that of (21). Note also that (21) has a root y' satisfying $y + y^\Delta = 1$, and this exists for any $\Delta > 0$. The reader may check that $y + y^\Delta = 1$ is equivalent to $\alpha = f(\alpha)$, and thus y' satisfies $\alpha = \beta$. To analyze (21) in general, let

$$g(y) = (\Delta - 1)\ln(1 + y^{\Delta-1}) + \ln(1 - y) - \ln y,$$

so $g(y) = 0$ has the same roots as (21). Then one may check that $g'(y) = 0$ if and only if

$$h(y) \stackrel{\text{def}}{=} \Delta(\Delta - 2)y^{\Delta-1} - (\Delta - 1)^2 y^\Delta - 1 = 0.$$

But $h(0) = -1$, $h(1) = -2$, and h has a single maximum on $[0, 1]$ at $y'' = (\Delta - 2)/(\Delta - 1)$. Now $h(y'') = (\Delta - 2)^\Delta/(\Delta - 1)^{\Delta-1} - 1 > 0$ if and only if $\Delta \ge 6$, and $h(y'') < 0$ otherwise. Therefore h has two roots in $[0, 1]$ if $\Delta \ge 6$; otherwise, it has no roots. Thus g has a single root in $[0, 1]$ if $\Delta \le 5$; otherwise, it has at most three roots. In the latter case, however, $g(0) = +\infty$, $g(1) = -\infty$, $g(y') = 0$, and a simple calculation shows

$$g'(y') = \frac{(\Delta - 1)^2(1 - y')^2 - 1}{y'(1 - y')} > 0$$

if and only if $\Delta \ge 6$, and $g'(y') < 0$ otherwise. These facts imply that g has exactly three roots if $\Delta \ge 6$.

Now the reader may check that the point (α', α') corresponding to y' (i.e., given by solving $y' = (\alpha'/(1 - \alpha'))^{1/\Delta}$) satisfies

$$\alpha + \beta + \Delta(\Delta - 2)\alpha\beta \le 1,$$

i.e.,

$$\left(\frac{1-\alpha}{\alpha}\right)\left(\frac{1-\beta}{\beta}\right) \ge (\Delta - 1)^2$$

if and only if $y' \ge y''$. This holds if and only if $\Delta \le 5$. Thus this point is a maximum for $\Delta \le 5$; otherwise, it is a saddle-point.

Thus φ has one stationary point in $\mathcal{T}$ (on $\alpha = \beta$) if $\Delta \le 5$, and this is a maximum.

(iv) By the above, if $\Delta \geq 6$, φ has no boundary maximum on $\mathcal{T}' = \{(\alpha, \beta) \in \mathcal{T} : \alpha \leq \beta\}$ and therefore by continuity has a maximum in the interior of $\mathcal{T}'$. By symmetry there is also a maximum in $\mathcal{T} \setminus \mathcal{T}'$. Thus, when $\Delta \geq 6$, φ has two symmetrical maxima and a single saddle-point on the line $\alpha = \beta$. Numerical values for the two maximum points can be obtained by solving (21) for y. Since we are assured that (21) has exactly three roots, we may locate these roots to arbitrary precision by repeated function evaluations. Once y is known to adequate precision, α can be recovered from (20). □

Proof of Claim 2.3. Let $\Omega = \{1, \ldots, N\}$ be an enumeration of the state space. When x is an N-vector and P an $N \times N$ matrix, we will use x_A to mean the vector $(x_i : i \in A)$ and P_{AB} to mean the matrix $(P_{ij} : i \in A, j \in B)$. First note that

$$\begin{aligned} \mathrm{d_{TV}}(p_{t+1}, p_t) = \mathrm{d_{TV}}(p_t P, p_{t-1} P) &= \tfrac{1}{2} \max_{||z||_\infty \leq 1} (p_t - p_{t-1}) P z \\ &\leq \tfrac{1}{2} \max_{||w||_\infty \leq 1} (p_t - p_{t-1}) w = \mathrm{d_{TV}}(p_t, p_{t-1}), \end{aligned}$$

since $||Pz||_\infty \leq ||z||_\infty$. Hence, by induction, $\mathrm{d_{TV}}(p_{t+1}, p_t) \leq \mathrm{d_{TV}}(p_1, p_0)$ and hence, using the triangle inequality, $\mathrm{d_{TV}}(p_t, p_0) \leq t\, \mathrm{d_{TV}}(p_1, p_0)$. Now, for $\emptyset \subset S \subset \Omega$, define

$$\Phi(S) = \sum_{i \in S} \sum_{j \in \overline{S}} \pi_i P_{ij} / \pi(S).$$

Thus $\Phi = \min\{\Phi(S) : S \subset \Omega \text{ and } 0 < \pi(S) \leq \frac{1}{2}\}$ is the "conductance" of $\mathcal{M}$. (Conductance is normally considered in the context of time-reversible Markov chains. However, both the definition and the line of argument employed here apply to non–time-reversible chains.) Now

$$\sum_{\substack{i \in A \\ j \in \overline{A}}} \pi_i P_{ij} \leq \sum_{\substack{i \in A \\ j \in \overline{A} \cap M}} \pi_i P_{ij} + \sum_{\substack{i \in A \cap M \\ j \in \overline{A}}} \pi_i P_{ij} \leq \pi(\overline{A} \cap M) + \pi(A \cap M) = \pi(M).$$

So by setting $(p_0)_A = \pi_A / \pi(A)$, $(p_0)_{\overline{A}} = 0$, we have that

$$\mathrm{d_{TV}}(p_1, p_0) = \tfrac{1}{2} ||\pi_A - \pi_A P||_1 / ||\pi_A||_1 = ||\pi_A P_{A\overline{A}}||_1 / ||\pi_A||_1 = \Phi(A) \leq \pi(M)/\pi(A).$$

But $\mathrm{d_{TV}}(\pi, p_0) \geq \frac{1}{2}$, because $\pi(A) \leq \frac{1}{2}$, and hence

$$\mathrm{d_{TV}}(\pi, p_t) \geq \mathrm{d_{TV}}(\pi, p_0) - \mathrm{d_{TV}}(p_t, p_0) \geq \tfrac{1}{2} - t\Phi(A).$$

Thus we cannot achieve $\mathrm{d_{TV}}(\pi, p_t) \leq e^{-1}$ until

$$t \geq (\tfrac{1}{2} - e^{-1})/\Phi \geq \pi(A)/8\pi(M).$$

By an averaging argument there must exist some initial state $x_0 \in A$ for which $\tau(x_0) \geq \pi(A)/8\pi(M)$. □

Proof of Claim 3.5. Differentiating (8), we have

$$\begin{aligned} \frac{\partial \psi}{\partial \alpha} &= a(-\ln \alpha + \ln(1-\alpha) - \beta b \delta), \\ \frac{\partial \psi}{\partial \beta} &= b(-\ln \beta + \ln(1-\beta) - \alpha a \delta), \end{aligned} \tag{22}$$

and

$$\frac{\partial^2\psi}{\partial\alpha^2} = \frac{-a}{\alpha(1-\alpha)}, \qquad \frac{\partial^2\psi}{\partial\beta^2} = \frac{-b}{\beta(1-\beta)}, \qquad \frac{\partial^2\psi}{\partial\alpha\,\partial\beta} = -ab\delta. \tag{23}$$

The following three facts about ψ are easily verified:

$$\psi(\alpha,\beta) \geq \psi(1-\alpha,\beta) \qquad \text{if } \alpha \leq \tfrac{1}{2}, \tag{24}$$

$$\psi(\alpha,\beta) \geq \psi(\alpha,1-\beta) \qquad \text{if } \beta \leq \tfrac{1}{2}, \tag{25}$$

$$\psi(\alpha,\beta) \geq \psi(\beta,\alpha) \qquad \text{if } \beta \leq \alpha \leq 1-\beta. \tag{26}$$

We wish to determine the regions where $\psi \geq a\ln 2$. These are connected neighborhoods of the local maxima of ψ. From (22) we see that ψ has no boundary maxima for α,β in the unit square $\mathcal{U}$. Thus, from (23), ψ has only local maxima or saddle-points in $\mathcal{U}$, and a stationary point is a local maximum if and only if

$$\alpha(1-\alpha)\beta(1-\beta) \leq 1/(ab\delta^2). \tag{27}$$

Thus, at any local maximum, either $\beta(1-\beta) \leq 1/(b\delta)$ or $\alpha(1-\alpha) \leq 1/(a\delta)$. If the former holds, this and $b\delta \geq 11.5$ (which holds for η sufficiently small) imply that $\beta < 0.1$, and hence $\beta < 1.2/b\delta$. An identical argument holds for α. Let us denote the rectangle $[\ell_\alpha, u_\alpha] \times [\ell_\beta, u_\beta]$ by $[\ell_\alpha, u_\alpha \,|\, \ell_\beta, u_\beta]$. Thus any local maximum of ψ must lie in the region $[0,1\,|\,0,1.2/b\delta] \cup [0,1.2/a\delta\,|\,0,1]$ and hence in the enclosing region $[0,1\,|\,0,1.2/b\delta] \cup [0,1.2/b\delta\,|\,0,1]$. (Recall that $a \geq b$.) In the square $[0,1.2/b\delta\,|\,0,1.2/b\delta]$, we have $\alpha,\beta \leq 1.2/b\delta < 0.11$ and hence

$$\psi(\alpha,\beta) < 2a(-0.11\ln(0.11) - 0.89\ln(0.89)) < a\ln 2.$$

Then, from (24) and (25), we also have $\psi(\alpha,\beta) < a\ln 2$ in $[1-1.2/b\delta, 1\,|\,0,1.2/b\delta]$ and $[0,1.2/b\delta\,|\,1-1.2/b\delta,1]$. Now, if $\beta \leq 1.2/b\delta$, let $\rho = 1-2\alpha$ and consider the upper bound

$$\psi(\alpha,\beta) \leq \Psi(\rho,\beta) \stackrel{\text{def}}{=} a(\ln 2 - \tfrac{1}{2}\rho^2) + b\beta(1-\ln\beta) - \tfrac{1}{2}(1-\rho)\beta ab\delta. \tag{28}$$

For fixed β, it is easily shown that Ψ is maximized if $\rho = \frac{1}{2}b\delta\beta \leq 0.6$. If $b\delta\beta = 1.2$, then $\rho = 0.6$ and

$$\max_\rho \Psi(\rho,\beta) \leq a(\ln 2 - 0.18) + 0.11a(1-\ln(0.11)) - 0.24a < a\ln 2.$$

Thus $\psi < a\ln 2$ everywhere on the boundary of $[1.2/b\delta, 1-1.2/b\delta\,|\,0,1.2/b\delta]$ (but not including the shared boundary with $\mathcal{U}$). Hence, by (26), $\psi < a\ln 2$ everywhere on the boundary of $[0,1.2/b\delta\,|\,1.2/b\delta, 1-1.2/b\delta]$. Moreover, $\psi(\alpha,\beta) \geq \psi(\beta,\alpha)$ for all points (α,β) in $[1.2/b\delta, 1-1.2/b\delta\,|\,0,1.2/b\delta]$. It follows that it is sufficient to determine β^* such that $\psi(\alpha,\beta) < a\ln 2$ everywhere in $[1.2/b\delta, 1-1.2/b\delta\,|\,\beta^*, 1.2/b\delta]$. To this end, again consider

$$\Psi_0(\beta) = \max_\rho \Psi(\rho,\beta) = a\ln 2 + b\beta(1-\ln\beta) - \tfrac{1}{2}ab\beta\delta + \tfrac{1}{8}ab^2\beta^2\delta^2.$$

Now $\Psi_0 < a\ln 2$ if

$$b\beta\delta^2 - 4\delta + 8(1-\ln\beta)/a < 0.$$

This inequality is satisfied, provided

$$2\left(1-\sqrt{1-2b\beta(1-\ln\beta)/a}\right) < \ b\beta\delta < \ 2\left(1+\sqrt{1-2b\beta(1-\ln\beta)/a}\right).$$

The right-hand inequality is clearly irrelevant since we are assuming that $\beta \le 1.2/b\delta$. Thus we need consider only the left-hand inequality; i.e., for fixed $\gamma = b\beta < 1.2/\delta$, we require that

$$\gamma\delta > 2\max_{a,b}\left(1-\sqrt{1-2\gamma(1-\ln\gamma+\ln b)/a}\right),$$

where the maximum is over $\frac{1}{2}-\eta \le b \le a \le 1$. Considering b first, the maximum occurs when $b=a$. So we have

$$\gamma\delta > \max_{\frac{1}{2}-\eta\le a\le 1} 2\left(1-\sqrt{1-2\gamma(1-\ln\gamma+\ln a)/a}\right). \tag{29}$$

But, because $a \ge \gamma$, the maximum now occurs when $a = \frac{1}{2}-\eta$. Thus it is enough to require that

$$\gamma\delta > 2\left(1-\sqrt{1-4\gamma(1-\ln\gamma-\ln 2)}\right),$$

because this will imply (29), provided that η is sufficiently small. To achieve $\gamma = 0.004$, it is sufficient that $\delta \ge 23.9$. □

REFERENCES

[1] E. A. Bender, *The asymptotic number of non-negative integer matrices with given row and column sums*, Discrete Math., 10 (1974), pp. 217–223.

[2] P. Berman and M. Karpinski, *On Some Tighter Inapproximability Results, Further Improvements*, Electronic Colloquium on Computational Complexity, Report TR98-065, 1998.

[3] G. Brightwell and P. Winkler, *Graph homomorphisms and phase transitions*, J. Combin. Theory Ser. B, 77 (1999), pp. 221–262.

[4] M. Dyer, A. Frieze, and M. Jerrum, *On counting independent sets in sparse graphs*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS'99), IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 210–217.

[5] M. Dyer and C. Greenhill, *On Markov chains for independent sets*, J. Algorithms, 35 (2000), pp. 17–49.

[6] M. Jerrum, L. Valiant, and V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[7] M. Jerrum, *Large cliques elude the Metropolis process*, Random Structures Algorithms, 3 (1992), pp. 347–359.

[8] M. Jerrum and A. Sinclair, *The Markov chain Monte Carlo method: An approach to approximate counting and integration*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1996, pp. 482–520.

[9] M. Luby and E. Vigoda, *Approximately counting up to four*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1997, pp. 682–687.

[10] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1997, pp. 1–10.

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

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1542–1556

SAFE WEAK MINIMIZATION REVISITED*

DIETER SPREEN†

Abstract. Minimization operators of different strengths have been studied in the framework of "predicative (safe) recursion." In this paper, a modification of these operators is presented. By adding the new operator to those used by Bellantoni–Cook and Leivant to characterize the polynomial-time computable functions, one obtains a characterization of the nondeterministic polynomial-time computable multifunctions. Thus the generation of the nondeterministic polytime multifunctions from the deterministic polytime functions parallels the generation of the computable functions from the primitive recursive ones.

Key words. implicit computational complexity, safe (predicative) recursion, recursion on notation, minimization, nondeterministic polynomial time, multifunction, normal form theorem

AMS subject classifications. 03D15, 03D20, 68Q15

PII. S0097539701387854

1. Introduction.

As is well known, unbounded minimization allows the generation of the computable functions from the primitive recursive ones. If one restricts to bounded minimization, then the latter class is closed under this operation. By relaxing the minimization condition in such a way that the smallest zero of a given function is no longer asked for but the smallest argument that is mapped onto an even result is asked for, Bellantoni [1, 2] could derive a machine-independent characterization of the class $\Box^p$ of functions computable in deterministic polynomial time by querying oracles in the polynomial-time hierarchy. The result is in the style of Cobham's classical characterization of the polytime functions [5]. Bellantoni showed that the class $\Box^p$ is the smallest class of functions containing certain basic functions and being closed under substitution, limited recursion on notation, and his weakened minimization operator.

Surprisingly, as is also demonstrated by Bellantoni, the characterization remains true in a tiered, resource-bound–free framework, where, in addition to safe composition and safe (predicative) recursion on notation, an unbounded version of relaxed minimization is used that allows us only to minimize safe arguments.

The ramified (predicative) approach to resource-bounded computations, now referred to as *implicit computational complexity*, has independently been introduced by Simmons [11], Leivant [7, 8], and Bellantoni and Cook [3]. One underlying idea is that data objects are used computationally in different guises. One has two types of values: values which are known in their entirety and which therefore can be examined completely, e.g., being recursed upon; and those values which are still emerging and which therefore can only be accessed in a more restricted way, e.g., by examining a few low-order bits. In the first case, the values are called normal; in the other they are called safe.

Bellantoni's result shows that, even in its relaxed form, minimization (bounded or unbounded) is still a powerful operation. In a recent paper, Danner and Pollett [6]

*Received by the editors April 12, 2001; accepted for publication (in revised form) March 29, 2002; published electronically August 5, 2002.
http://www.siam.org/journals/sicomp/31-5/38785.html
†Theoretische Informatik, Fachbereich Mathematik, Universität Siegen, 57068 Siegen, Germany (spreen@informatik.uni-siegen.de).

weakened Bellantoni's safe minimization operation again by first limiting the verification that a computed function argument c is minimal to only those numbers d that are less than the length of c and then prescribing which bits of c a further computation may at most have access to. This operation, called limited safe weak minimization, is necessarily multivalued. They showed that the smallest class of multifunctions generated from certain initial functions by safe composition, safe recursion on notation, and this new operation is exactly the class **NPMV** of partial multifunctions computable in nondeterministic polynomial time.

As already stated by Danner and Pollet, the definition of limited safe weak minimization is reminiscent of limited minimization and thus not in the spirit of implicit computational complexity. In this paper, we propose a modified version of safe weak minimization which remedies this problem. Moreover, we show that the abovementioned characterization of **NPMV** holds true when the new version of safe weak minimization is used.

The paper is organized as follows. Section 2 contains basic definitions. Here the various minimization operators considered in the literature are introduced. Our modification of these operators as well as the main result are presented in section 3. The proof of this theorem follows from two propositions, one of which is a normal form result for multifunctions in **NPMV**. It entails that every function in this class can be generated from certain basic functions by safe composition, safe recursion on notation, and safe minimization on notation, the new minimization operator introduced in this paper. The proof is given in section 4. Final remarks appear in section 5.

The normal form result is based on an appropriate encoding of Turing machine computations. In the appendix, a coding of finite sequences is presented, and functions are defined by using only safe composition and safe recursion on notation that allow us to check whether a number is the code of a finite sequence and, if this is the case, the computation of the elements of the sequence as well as its length from the code.

2. Basic definitions and facts. A *partial multifunction* is a map $f\colon \mathbb{N}^k \rightharpoonup \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ for some k, where $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ is the collection of all finite subsets of the natural numbers. Alternatively, f can be viewed as a relation on $\mathbb{N}^{k+1}$ satisfying the constraint that, for all $\bar{x}$, $\{\, y \mid (\bar{x}, y) \in f \,\}$ is finite. We write $f(\bar{x}) \mapsto y$ when y is a (possible) outcome of f and read $f(\bar{x})$ as the set of all y with $f(\bar{x}) \mapsto y$. Therefore, we also write $y \in f(\bar{x})$ instead of $f(\bar{x}) \mapsto y$. If f is single-valued, i.e., $f(\bar{x}) = \{y\}$, we identify y with $\{y\}$ and write $f(\bar{x}) = y$. For two partial multifunctions f and g, $f(\bar{x}) \leq g(\bar{x})$ means that, for every $y \in f(\bar{x})$, there is some $z \in g(\bar{x})$ such that $y \leq z$.

The *length* $|x|$ of a number x is defined as $\lceil \log_2(x+1) \rceil$. If $\bar{x}$ is a vector of n numbers, we write $|\bar{x}|$ for the vector $|x_1|, \ldots, |x_n|$. Similarly, we write $\bar{f}(\bar{x})$ for $f_1(\bar{x}), \ldots, f_m(\bar{x})$ so that, e.g., $\bar{y} \in \bar{f}(\bar{x})$ means that $y_1 \in f_1(\bar{x}), \ldots, y_m \in f_m(\bar{x})$.

Inputs to functions are categorized as *normal* or *safe*; the normal ones are written to the left of a semicolon, and the safe ones are written to the right. The variables x, y, z are usually used in normal position, and a, b, c are usually used in safe position. For a function class B, the subclass consisting of the functions with only normal arguments is denoted by $\mathrm{Norm}(B)$.

In [1, 3], Bellantoni and Cook characterized the class **FP** of polytime functions inductively by using a set B_0 of initial functions and safe composition as well as safe recursion on notation as closure operations.

DEFINITION 2.1. *The set B_0 consists of the functions* (1)–(5) *below.*

(i) (Constant) 0 *(a zero-ary function).*

(ii) (Projections) $\pi_j^{n,m}(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{n+m}) = x_j$ *for* $1 \leq j \leq n+m$.

(iii) (Successors) $s_0(;a) = 2a$, *and* $s_1(;a) = 2a+1$. *Write "*ai*" for* $s_i(;a)$, *where* $i \in \{0,1\}$.

(iv) (Predecessor) p *such that* $p(;0) = 0$ *and* $p(;a0) = p(;a1) = a$.

(v) (Conditional)

$$\operatorname{cond}(;a,b,c) = \begin{cases} b & \text{if } a \bmod 2 = 0, \\ c & \text{otherwise,} \end{cases}$$

where $x \bmod 2$ *is the value of the low-order bit of* x.

Observe that the predecessor is a string predecessor just as the two successor operations are string successors.

The operations of safe composition and safe recursion on notation are defined as follows. Note that we directly consider multifunctions.

DEFINITION 2.2 (safe composition). *A multifunction* f *is defined by* safe composition *from given multifunctions* h, $\bar{r}$, *and* $\bar{t}$ *(in symbols,* $f = \mathrm{SC}(h,\bar{r},\bar{t})$*) when*

$$f(\bar{x};\bar{a}) = \bigcup \{\, h(\bar{y};\bar{b}) \mid \bar{y} \in \bar{r}(\bar{x};) \wedge \bar{b} \in \bar{t}(\bar{x};\bar{a}) \,\}.$$

As usual, we write $f(\bar{x};\bar{a}) = h(\bar{r}(\bar{x};);\bar{t}(\bar{x};\bar{a}))$.

DEFINITION 2.3 (safe recursion on notation). *A multifunction* f *is defined by* safe recursion on notation *from given multifunctions* g *and* h_0, h_1 *(in symbols,* $f = \mathrm{SRN}(g,h_1,h_2)$*) when, for* $i \in \{0,1\}$,

$$\begin{aligned} f(0,\bar{x};\bar{a}) &= g(\bar{x};\bar{a}), \\ f(yi,\bar{x};\bar{a}) &= h_i(y,\bar{x};\bar{a},f(y,\bar{x};\bar{a})) \quad \text{for } yi \neq 0. \end{aligned}$$

Let $[B_0;\mathrm{SC},\mathrm{SRN}]$ denote the smallest class of functions containing B_0 and closed under safe composition as well as safe recursion on notation.

THEOREM 2.4 (Bellantoni and Cook). $\mathbf{FP} = \mathrm{Norm}([B_0;\mathrm{SC},\mathrm{SRN}])$.

This theorem gives strong reasons to consider the polytime functions as the complexity-theoretic analogue of the primitive recursive functions. So the following question comes up: What is the complexity-theoretic analogue of the partial recursive functions, and can the functions in this class be generated from the polytime functions by applying a suitable minimization operator? This question was the starting point for recent investigations of Danner and Pollett [6].

In his dissertation [1, 2], Bellantoni introduced a safe minimization operator.

DEFINITION 2.5 (safe minimization). *A function* f *is defined by* safe minimization *from a given function* h *(in symbols,* $f = \mathrm{SM}(h)$*) when*

$$f(\bar{x};\bar{a}) = \begin{cases} s_1(;\mu b.\, h(\bar{x};\bar{a},b) \bmod 2 = 0) & \text{if there is some such } b, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, he showed that the functions in $\Box^p$, i.e., the functions computable on a polynomial-time bounded oracle Turing machine with an oracle in the polynomial-time hierarchy, are exactly the functions that can be generated from functions in B_0 by safe composition, safe recursion on notation, and safe minimization and have only normal arguments.

THEOREM 2.6 (Bellantoni). $\Box^p = \mathrm{Norm}([B_0;\mathrm{SC},\mathrm{SRN},\mathrm{SM}])$.

This class is far beyond what could be considered as a complexity-theoretic analogue of the partial recursive functions. In their paper [6], Danner and Pollett intro-

duced two weakenings of safe minimization, safe weak minimization and limited safe weak minimization, and, by using the latter operator instead of safe minimization, they showed that the collection of functions in the resulting class that have only normal arguments is exactly the class **NPMV** of all partial multifunctions computable in nondeterministic polynomial time, thus answering the above-posed question.

DEFINITION 2.7 (safe weak minimization). *A multifunction f is defined by* safe weak minimization *from a given a multifunction g when*

$$f(\bar{x};\bar{a}) \mapsto b \Leftrightarrow g(\bar{x};\bar{a},b) \bmod 2 \mapsto 0 \wedge (\forall c < |b|) g(\bar{x};\bar{a},c) \bmod 2 \mapsto 1.$$

If f is defined by safe weak minimization from g, we write

$$f(\bar{x};\bar{a}) = \mu^{w} b.\, g(\bar{x};\bar{a},b) \bmod 2 = 0.$$

As is shown in the next example, if f is defined from g by safe weak minimization, then, for any inputs $\bar{x}$ and $\bar{a}$, $f(\bar{x};\bar{a})$ may have superexponentially many outputs even if g is single-valued.

Example 2.8. Set $g(x;a) = \mathrm{cond}(;P(x;b),1,0)$, where the function P is defined by, for $i \in \{0,1\}$,

$$P(0;b) = b, \qquad P(xi;b) = p(;P(x;b)).$$

$P(x;b)$ takes $|x|$ predecessors of b.

It follows that $g(x;b)$ has value 0 if the $(|x|+1)$st low-order bit of b is 1; otherwise, its value is 1. Now let

$$f(x;) = \mu^{w} b.\, g(x;b) \bmod 2 = 0.$$

Then $f(x;) \mapsto b$ for all numbers b such that $|b| \le 2^{|x|}$ and the $(|x|+1)$st low-order bit of b is 1.

By definition, if the $(|x|+1)$st low-order bit of b is 0, then $g(x;b) = 1$, and hence $f(x;) \not\mapsto b$. Now suppose that $|b| > 2^{|x|}$ and $g(x;b) \bmod 2 = 0$. As $g(x;2^{|x|}) \bmod 2 = 0$, it follows that $f(x;) \not\mapsto b$. On the other hand, if $|b| \le 2^{|x|}$ so that the $(|x|+1)$st low-order bit of b is 1, then $f(x;) \mapsto b$, since for d with $d < |b|$ one has that $|d| \le |x|$ and hence that the $(|x|+1)$st low-order bit of d is 0, which means that $g(x;d) = 1$.

Thus the cardinality of $f(x;)$ is equal to the cardinality of the set of binary sequences of length $2^{|x|}-1$; i.e.,

$$\|f(x;)\| = 2^{2^{|x|}-1}.$$

Since, for any function $h \in \mathbf{NPMV}$, one has that $\|h(\bar{x})\| \le 2^{p(|\bar{x}|)}$ for some polynomial p, it follows that the class **NPMV** is not closed under safe weak minimization, which means that this operation is still too strong.

Define the function $a \underline{\bmod} v = a \bmod 2^{|v|}$. Then $a \underline{\bmod} v$ is the number given by the $|v|$ low-order bits of a. For a sequence $\bar{a} = a_1, \ldots, a_k$, write $\bar{a} \underline{\bmod} v$ for $a_1 \underline{\bmod} v$, $\ldots$, $a_k \underline{\bmod} v$.

DEFINITION 2.9 (limited safe weak minimization). *A multifunction f is defined by* limited safe weak minimization *from given multifunctions g and h (in symbols, $f = \mathrm{LSWM}(g,h)$) when*

$$f(\bar{x};\bar{a}) = (\mu^{w} b.\, g(\bar{x};\bar{a},b) \bmod 2 = 0) \underline{\bmod} h(\bar{x};).$$

THEOREM 2.10 (Danner and Pollett).

$$\mathbf{NPMV} = \mathrm{Norm}([B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{LSWM}]).$$

If f is defined by limited safe weak minimization from g and h, then the application of $\underline{\mathrm{mod}}$ cuts the possibly superexponentially many outputs b of $\mu^w b.\, g(\bar{x}; \bar{a}, b) \bmod 2 = 0$ down to those with $|b| \leq |z|$ for some z with $h(\bar{x};) \mapsto z$, thus to exponentially many. Limited safe weak minimization is reminiscent of limited minimization. As Danner and Pollett remark, by using this operation, one enters the "gray area" of implicit computational complexity.

3. Safe minimization on notation. In this section, we present a modification of safe weak minimization which follows the ideas of implicit computational complexity and show that an analogue of Theorem 2.10 holds.[1] The idea is to minimize with respect to the prefix order on binary representations.

For numbers a and b, respectively, let $a_n \dots a_0$ and $b_m \dots b_0$ be their binary representations. Define $a \sqsubseteq b$ if $n \leq m$ and $a_i = b_i$ for all $i \leq n$. Write $a \sqsubset b$ if $a \sqsubseteq b$ and $a \neq b$.

DEFINITION 3.1. *Let $g(\bar{x}; \bar{a})$ be a multifunction. A multifunction $h(\bar{x}; \bar{a})$ is a* companion *of g if*

$$h(\bar{x}; \bar{a}) \bmod 2 \mapsto 1 \Rightarrow g(\bar{x}; \bar{a}) \bmod 2 \not\mapsto 0.$$

DEFINITION 3.2 (safe minimization on notation). *A multifunction f is defined by* safe minimization on notation *from a given multifunction g and its companion h (in symbols, $f = \mathrm{SMN}(g, h)$) when*

$$f(\bar{x}; \bar{a}) \mapsto b \Leftrightarrow g(\bar{x}; \bar{a}, b) \bmod 2 \mapsto 0 \wedge (\forall c \sqsubset b) h(\bar{x}; \bar{a}, c) \bmod 2 \mapsto 1.$$

Call a multifunction $g(\bar{x}; \bar{a})$ *consistent* if not both $g(\bar{x}; \bar{a}) \bmod 2 \mapsto 0$ and $g(\bar{x}; \bar{a}) \bmod 2 \mapsto 1$. Then a multifunction is consistent exactly if it is its own companion. If a multifunction f is defined by safe minimization on notation from a given consistent multifunction g (and its companion g), we say that f is defined by *consistent safe minimization on notation* (in symbols: $f = \mathrm{CSMN}(g)$). We also write $f(\bar{x}; \bar{a}) = \mu^c b.\, g(\bar{x}; \bar{a}, b) \bmod 2 = 0$ in this case.

In order to see the effect of the modified quantification in the second condition in Definition 3.2, let us consider the functions defined in Example 2.8 again.

Example 3.3. As has been shown for the multifunction f,

$$f(x;) = \{\, b \mid |x| + 1 \leq |b| \leq 2^{|x|} \,\} = \{\, b \mid 2^{|x|} \leq b < 2^{2^{|x|}} \,\}.$$

Since the function g is single-valued, it is consistent. Define

$$f'(x;) = \mu^c b.\, g(x; b) \bmod 2 = 0.$$

Then $f'(x;)$ is the set of all numbers of minimal length such that the $(|x|+1)$st bit is 1, which means that it is the set of all numbers of exactly length $|x| + 1$. Hence

$$f'(x;) = \{\, b \mid 2^{|x|} \leq b < 2^{|x|+1} \,\}.$$

[1] This research has been motivated by an earlier version of Danner and Pollett's paper in which only safe weak minimization was studied. It should be noted that the definitions and results in this section have been obtained independently of what the authors presented in their revised version.

For numbers a and b, let $a \preceq b$ if $|a| \leq |b|$. Then $\preceq$ is a preorder. Write $a \prec b$ if $a \preceq b$ and not $b \preceq a$.

Obviously, $a \sqsubset b$ exactly if $a \prec b$ and $a = b \operatorname{mod} a$. Therefore, we have the following lemma, which is useful in the derivation of our main result below.

LEMMA 3.4. *Let f, g, and h be multifunctions, and let h be a companion of g. Then f is obtained by safe minimization on notation from g and h if and only if, for any $\bar{x}$, $\bar{a}$, and b,*

$$f(\bar{x}; \bar{a}) \mapsto b \Leftrightarrow g(\bar{x}; \bar{a}, b) \bmod 2 \mapsto 0 \wedge (\forall c \prec b) h(\bar{x}; \bar{a}, b \operatorname{mod} c) \bmod 2 \mapsto 1.$$

THEOREM 3.5.

$$\mathbf{NPMV} = \mathrm{Norm}([B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{CSMN}]) = \mathrm{Norm}([B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{SMN}]).$$

The theorem follows from the following propositions, the first of which will be proved in the next section. Obviously $[B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{CSMN}] \subseteq [B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{SMN}]$.

PROPOSITION 3.6. *Let $f \in \mathbf{NPMV}$. Then there are functions $T(z; \bar{a}, c)$, $\mathrm{res}(z; a)$, and $\mathrm{bnd}(\bar{x};)$ in $[B_0; \mathrm{SC}, \mathrm{SRN}]$ such that, for all inputs $\bar{x}$,*

$$f(\bar{x}) = \mathrm{res}(\mathrm{bnd}(\bar{x};); \mu^c b. T(\mathrm{bnd}(\bar{x};); \bar{x}, b) \bmod 2 = 0).$$

COROLLARY 3.7. *Let $f(\bar{x}) \in \mathbf{NPMV}$. Then $f(\bar{x};) \in [B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{CSMN}]$.*

The proof of the converse implication uses a technique of Bellantoni.

DEFINITION 3.8. *Let f be a multifunction (note that we do not separate the arguments into normal and safe here), and let q be a polynomial.*

(i) *Function $f(\bar{x}, \bar{a})$ is a* polynomial checking function on $\bar{x}$ with threshold q *if, for all $\bar{x}$, $\bar{a}$, w, and v satisfying $|v| \geq q(|\bar{x}|) + |w|$,*

$$f(\bar{x}, \bar{a}) \operatorname{mod} w = f(\bar{x}, \bar{a} \operatorname{mod} v) \operatorname{mod} w.$$

(ii) *Function $f(\bar{x}, \bar{a})$ is* polymax bounded by q on $\bar{x}$ *if, for all $\bar{x}$, $\bar{a}$, $|f(\bar{x}, \bar{a})| \leq q(|\bar{x}|) + \max_i |a_i|$.*

Note that the equation in Definition 3.8 (i) has to be understood as an equation between sets.

PROPOSITION 3.9. *If $f(\bar{x}; \bar{a})$ is in $[B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{SMN}]$, then f is a polymax-bounded polynomial checking function on $\bar{x}$.*

Proof. The proof proceeds by induction on the length of the derivation of f as a function in $[B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{SMN}]$. Without restriction we can assume that the polynomial of the size bound and the polynomial of the checking threshold are identical by summing the two together.

For the initial functions zero, projection, predecessor, conditional, and successors, the polynomial is at most 1, and the statement of the proposition is easily verified.

The proof that the statement holds in the induction cases is the same as in [1] and/or [3, 2], except in the case of safe minimization on notation discussed below.

Let h be a companion of g, and assume that $f = \mathrm{SMN}(g, h)$. By the induction hypothesis, there are polynomials q_g and q_h, respectively, witnessing that g and h are polychecking and polymax-bounded on $\bar{x}$. Let q be the sum of q_g and q_h.

In order to verify that f is polymax-bounded on $\bar{x}$, we show that, for all $c \in f(\bar{x}; \bar{a})$,

$$|c| \leq q(|\bar{x}|) + 2.$$

Assume, to the contrary, that $|c| > q(|\bar{x}|) + 2$, and set $v = 2^{q(|\bar{x}|)+1}$ as well as $b = c \operatorname{\underline{mod}} v$. Then $|v| = q(|\bar{x}|)+2$. Since g is polychecking on $\bar{x}$ with polynomial threshold q, it follows that

$$\begin{aligned} g(\bar{x};\bar{a},c) \bmod 2 &= g(\bar{x};\bar{a} \operatorname{\underline{mod}} v, c \operatorname{\underline{mod}} v) \bmod 2 \\ &= g(\bar{x};\bar{a} \operatorname{\underline{mod}} v, b \operatorname{\underline{mod}} v) \bmod 2 \\ &= g(\bar{x};\bar{a},b) \bmod 2. \end{aligned}$$

As $c \in f(\bar{x};\bar{a})$, we have that $g(\bar{x};\bar{a},c) \bmod 2 \mapsto 0$. Thus $g(\bar{x};\bar{a},b) \bmod 2 \mapsto 0$ as well. From our assumption on $|c|$, we obtain that $v \prec c$. Hence $h(\bar{x};\bar{a},b) \bmod 2 \mapsto 1$, by Lemma 3.4, in contradiction to the fact that h is a companion of g. Define $p(|\bar{x}|) = q(|\bar{x}|) + 2$. Then p witnesses that f is polymax-bounded on $\bar{x}$.

For the proof that f is also polychecking on $\bar{x}$ with threshold p, let $\bar{x}$, $\bar{a}$, w, and v be such that $|v| \geq p(|\bar{x}|) + |w|$. Then $|v| \geq q(|\bar{x}|) + 2$. By the induction hypothesis, it thus follows for c with $|c| \leq q(|\bar{x}|) + 2$ that

$$g(\bar{x};\bar{a},c) \bmod 2 = g(\bar{x};\bar{a} \operatorname{\underline{mod}} v, c \operatorname{\underline{mod}} v) \bmod 2 = g(\bar{x};\bar{a} \operatorname{\underline{mod}} v, c) \bmod 2$$

and correspondingly for h. As we have seen above, it follows for $c \in f(\bar{x};\bar{a})$ as well as for $c \in f(\bar{x};\bar{a} \operatorname{\underline{mod}} v)$ that $|c| \leq q(|\bar{x}|) + 2$. Hence $f(\bar{x};\bar{a}) = f(\bar{x};\bar{a} \operatorname{\underline{mod}} v)$, which entails that

$$f(\bar{x};\bar{a}) \operatorname{\underline{mod}} w = f(\bar{x};\bar{a} \operatorname{\underline{mod}} v) \operatorname{\underline{mod}} w. \qquad \square$$

COROLLARY 3.10. *Let $f(\bar{x};\bar{a})$ be a multifunction in $[B_0; \mathrm{SC}, \mathrm{SRN}, \mathrm{SMN}]$. Then $f(\bar{x},\bar{a})$ is computable in nondeterministic polynomial time.*

Proof. The proof is by induction on the definition of f. The initial functions are clearly computable in polynomial time and hence in **NPMV**. When g is given inductively, we assume that it is computed by a nondeterministic Turing machine M_g in time p_g.

For safe composition, note that **NPMV** is closed with respect to composition. Now suppose that $f(y,\bar{x};\bar{a})$ is defined by safe recursion on notation from g and h_0, h_1. Then, for any y, we have that $f(y,\bar{x};\bar{a}) \mapsto z$ exactly if there is a sequence $t_0, \ldots, t_{|y|}$ such that $g(\bar{x};\bar{a}) \mapsto t_{|y|}$, $h_{y(i)}(\lfloor y/2^{i+1} \rfloor, \bar{x};\bar{a},t_{i+1}) \mapsto t_i$ for all $i < |y|$, and $t_0 = z$. Here, $y(i)$ denotes the ith bit of y. Furthermore, for any such sequence and all i, we have that $f(\lfloor y/2^i \rfloor, \bar{x};\bar{a}) \mapsto t_i$, so $|t_i| \leq q(|y|,|\bar{x}|,|\bar{a}|)$, where q is the polynomial length-bound existing by the preceding lemma. Thus, to compute $f(y,\bar{x},\bar{a})$, guess a sequence $t_0, \ldots, t_{|y|}$ so that, for every t_i, $|t_i| \leq q(|y|,|\bar{x}|,|\bar{a}|)$, and verify that the above condition holds for each element of the sequence by running M_g, M_{h_0}, or M_{h_1}, respectively, and comparing the output to the previous element of the sequence. If none of the verifications fails and $t_0 = z$, then accept and output z. Guessing the sequence takes $|y|q(|y|,|\bar{x}|,|\bar{a}|)$ steps, and the verification takes time $|y|p(|y|,|\bar{x}|,|\bar{a}|,q(|y|,|\bar{x}|,|\bar{a}|))$, where p is the sum of the polynomials p_g, p_{h_0}, and p_{h_1}.

Finally, assume that $f = \mathrm{SMN}(g,h)$, and let q be the polynomial length-bound existing by the preceding lemma so that $|f(\bar{x};\bar{a})| \leq q(|\bar{x}|,|\bar{a}|)$. We compute $f(\bar{x},\bar{a})$ as follows. Guess b with $|b| \leq q(|\bar{x}|,|\bar{a}|)$. Next verify that $g(\bar{x},\bar{a},b) \bmod 2 \mapsto 0$, and reject if not. If the verification is successful, run M_h on input $\bar{x},\bar{a},c$ for every $c \sqsubset b$. If M_h accepts with output 1 in each case, then accept with output b; otherwise, reject. Guessing b takes $q(|\bar{x}|,|\bar{a}|)$ steps, and the verifications take at most time $(1 + q(|\bar{x}|,|\bar{a}|))p(|\bar{x}|,|\bar{a}|,q(|\bar{x}|,|\bar{a}|))$, where p is the sum of the polynomials p_g and p_h. $\square$

In the proof of Proposition 3.9, we took advantage of the proviso made in the definition of safe minimization on notation, namely, that the second argument of this operator has to be a companion of the first argument. As can be seen from the following example, without this condition, the class **NPMV** would not be closed under safe minimization on notation, which means that Theorem 3.5 and hence Proposition 3.9 would be false. The same holds for the operation of consistent safe minimization on notation and the requirement that the argument of this operator be consistent.

Example 3.11. Let $g_1(x,b) \mapsto 0,1$ and $g_2(x,b) \mapsto 1$ for all $x,b \in \mathbb{N}$. Then $g_1, g_2 \in \mathbf{NPMV}$, but neither is g_1 consistent nor is g_2 a companion of g_1. Now, for $i = 1,2$, define f_i by

$$f_i(x) \mapsto b \Leftrightarrow g_1(x,b) \bmod 2 \mapsto 0 \wedge (\forall c \sqsubset b) g_i(x,c) \bmod 2 \mapsto 1.$$

Then it is readily verified that $f_i(x) \mapsto z$ for all $z \in \mathbb{N}$. Thus $f_i \notin \mathbf{NPMV}$, since otherwise there exists some polynomial p_i such that $\|f_i(x)\| \leq 2^{p_i(|x|)}$.

If, however, we consider the function $h \in \mathbf{NPMV}$ with $h(x,b) \mapsto 0$ for all $x,b \in \mathbb{N}$, then h is a companion of g_1, and $\mathrm{SMN}(g_1,h)$ is the nowhere defined function.

4. Proof of the normal form result. The proof of Proposition 3.6 proceeds in the usual way by an appropriate encoding of Turing machine computations. In the appendix, a coding of finite sequences of numbers is presented such that there are functions $\mathrm{first}, \mathrm{last}, \mathrm{lth}, \mathrm{dc}, \mathrm{seq} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, with the following properties: if w is the Gödel number of a sequence $a_1, \ldots, a_k$, then $\mathrm{first}(y;a) = a_1$ for all y with $|y| \geq |a_1|$, $\mathrm{last}(y;w) = a_k$ for all y with $|y| \geq |w|$, $\mathrm{lth}(y;w) = k$, and $\mathrm{dc}(y,z;w) = a_{|z|+1}$ for all y with $|y| \geq k, |a_j|$ $(1 \leq j \leq k)$ and z with $|z| < \mathrm{lth}(y;w)$.

Moreover, for z,w with $|z| \geq |w|$, $\mathrm{seq}(z;w) = 1$ if w is the Gödel number of a finite sequence, and $\mathrm{seq}(z;w) = 0$ otherwise.

In addition, functions $\leq, <, \mathrm{ebit}, \mathrm{obit} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$ are defined such that, for z,a,b with $|z| \geq |a|, |b|$, $\leq(z;a,b) = 1$ if $a \leq b$, and $\leq(z;a,b) = 0$ otherwise, and correspondingly for $<(z;a,b)$. Moreover, $\mathrm{ebit}(x;a)$ and $\mathrm{obit}(x;a)$, respectively, are the coefficients of $2^{2|x|}$ and $2^{2|x|+1}$ in the binary expansion of a.

The Turing machines we work with have n read-only input tapes, m read/write work tapes, and one write-only output tape. The alphabet consists of the symbols 0, 1, and $*$ ($*$ for blank), and the state set is $Q = \{q_0, q_1, q_2, \ldots, q_r\}$. We always presume that q_0 is the starting and q_1 the accepting state.

Our machines work nondeterministically and in polynomial time. Moreover, we assume the following, again without loss of generality.

(i) The accepting state q_1 can only be entered within the given time limit.

(ii) The head on the output tape prints only 0's and 1's. If the machine is in state q_1, the finite content of the output tape right to its head is the binary representation of the output of the machine.

(iii) In the start situation, the machine is in state q_0, and on each input tape the input is written in such a way that it starts one cell right of the cell scanned by the head.

A *configuration* is a $(3n+3m+2)$-tuple

$$(q, u_1, a_1, v_1, \ldots, u_{n+m}, a_{n+m}, v_{n+m}, v_{n+m+1}),$$

where

(i) q is the present state of the machine,

(ii) v_{n+m+1} is the finite content of the output tape right to its head, written in reverse order, and, for $1 \leq j \leq n+m$,

(iii) u_j, is the finite content of tape j left to its head,
(iv) a_j is the content of the cell scanned by the head on tape j, and
(v) v_j is the finite content of tape j right to its head, written in reverse order.

We always assume that the words u_j and v_j are of the form $b_1 \dots b_k$ such that b_1 is not $*$.

In order to encode configurations, we code every state q_i by i. The words u_j, a_j, and v_j are coded as follows. Replace each 0 by "10," each 1 by "11," and each $*$ by "00." The resulting strings of zeros and ones are then the binary representations of the codes of these words. In this way, we obtain a $(3n + 3m + 2)$-tuple $(i, \bar{u}_1, \bar{a}_1, \bar{v}_1, \dots, \bar{u}_{n+m}, \bar{a}_{n+m}, \bar{v}_{n+m}, \bar{v}_{n+m+1})$ of numbers, which is encoded as explained in the beginning of this section.

As is easily verified, a number w is the code of an entry of some configuration just if, for every odd i such that the coefficient of 2^{i-1} in the binary expansion of w is not 0, the coefficient of 2^i is 1. This is tested by the function $\mathrm{seq1} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$ defined in the appendix. Moreover, w is the code of the binary representation of some number, written in reverse order, exactly if, in addition, the binary representation of w contains no substring of the form 100.

Let the function P be as in Example 2.8, set $\hat{s}_1(z;) = s_1(;z)$, and define

$$\begin{aligned}
&\mathrm{seq4}(0; w) = 1,\\
&\mathrm{seq4}(zi; w) = \mathrm{cond}(;\mathrm{seq4}(z; w), 0, \mathrm{cond}(; P(z; w),\\
&\qquad\qquad \mathrm{cond}(; P(\hat{s}_1(z;); w), \mathrm{cond}(; P(\hat{s}_1^2(z;); w), 1, 0), 1), 1))
\end{aligned}$$

for $zi \neq 0$. Then $\mathrm{seq4} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, and, for z, w with $|z| \geq |w|$, we have that $\mathrm{seq4}(z; w) = 1$ if the binary representation of w contains no substring of the form 100 and $\mathrm{seq4}(z; w) = 0$ otherwise.

If w is the code of the binary representation of some number, written in reverse order, we need to compute that number. Let

$$\begin{aligned}
&\mathrm{nb}(0; w) = 0,\\
&\mathrm{nb}(yi; w) = \mathrm{cond}(; \mathrm{obit}(y; w), \mathrm{nb}(y; w),\\
&\qquad\qquad \mathrm{cond}(; \mathrm{ebit}(y; w), s_0(; \mathrm{nb}(y; w)), s_1(; \mathrm{nb}(y; w))))
\end{aligned}$$

for $yi \neq 0$. Then $\mathrm{nb} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, and, in case w is such a code, we have for $|y| \geq |w|$ that $\mathrm{nb}(y; w)$ is the corresponding number.

Now let M be a Turing machine in the way described above. We need to construct a Turing predicate $T_M(y; \bar{x}, c) \in [B_0; \mathrm{SC}, \mathrm{SRN}]$ such that, for $y, \bar{x}, c$ with $|y| \geq r, |c|, |x_i|$, for $1 \leq i \leq n$, $T_M(y; \bar{x}, c) = 1$ exactly if c is the Gödel number of an accepting computation of machine M with respect to input $\bar{x}$.

1. Set

$$\begin{aligned}
\mathrm{cf}_M(y; c) = \mathrm{seq}(y; c) \wedge \mathrm{lth}(y; c) = 3n + 3m + 2 \wedge {\leq}(y; \mathrm{first}(y; c), r)\\
\wedge \bigwedge_{i=1}^{3n+3m+1} \mathrm{seq1}(y; \mathrm{dc}(y, 2^i - 1; c)).
\end{aligned}$$

Since $3n+3m+2$ is a fixed number, the term $\mathrm{lth}(y; c) = 3n+3m+2$ can be expressed in such a way that it is in $[B_0; \mathrm{SC}, \mathrm{SRN}]$. Hence $\mathrm{cf}_M \in [B_0; \mathrm{SC}, \mathrm{SRN}]$. If $|y| \geq r, |c|$, $\mathrm{cf}_M(y; c) = 1$ just if c is the Gödel number of a configuration of M.

2. By using the transition table of machine M, one can further construct a function $\mathrm{trans}_M \in [B_0; \mathrm{SC}, \mathrm{SRN}]$ such that, for y, c, d with $|y| \geq r, |c|, |d|$, $\mathrm{trans}_M(y; c, d) = 1$ if c and d, respectively, are Gödel numbers of configurations c' and d' of M and there is a transition from c' to d' and $\mathrm{trans}_M(y; c, d) = 0$ otherwise.

3. Let $a \vee b = \mathrm{cond}(; a, \mathrm{cond}(; b, 0, 1), 1)$, define

$$\begin{aligned} f_6(0, y; c) &= \mathrm{seq}(y; c), \\ f_6(zi, y; c) &= [f_6(z, y; c) \wedge \mathrm{trans}_M(y; \mathrm{dc}(y, z; c), \mathrm{dc}(y, \hat{s}_1(z;); c)) \\ &\qquad \wedge <(y; s_1(; z), \mathrm{lth}(y; c))] \vee [\leq(y; \mathrm{lth}(y; c), s_1(; z)) \wedge f_6(z, y; c)] \end{aligned}$$

for $zi \neq 0$, and set

$$\mathrm{comp}_M(y; c) = f_6(y, y; c).$$

Then $\mathrm{comp}_M \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, and, for y, c with $|y| \geq r, |c|$, we have that $\mathrm{comp}_M(y; c) = 1$ exactly if c is the Gödel number of a sequence of codings of configurations $c_0, \ldots, c_k$ of M such that there is a transition from c_j to c_{j+1} for all $j < k$.

4. Let $=(y; a, b) = \leq(y; a, b) \wedge \leq(y; b, a)$ and $\mathrm{cr}(y; w) = \mathrm{seq1}(y; w) \wedge \mathrm{seq4}(y; w)$. Moreover, let I be the set of all i with $1 \leq i \leq 3n + 3m + 1$ such that i is not divisible by 3 if $1 \leq i \leq n$. Define

$$\begin{aligned} \mathrm{init}_M(y; \bar{x}, c) = \mathrm{cf}_M(y; c) \wedge \leq(y; \mathrm{first}(y; c), 0) \wedge \bigwedge_{i \in I} \leq(y; \mathrm{dc}(y, 2^i - 1; c), 0) \\ \wedge \bigwedge_{i=1}^{n} [\mathrm{cr}(y; \mathrm{dc}(y, 2^{3i} - 1; c)) \wedge =(y; \mathrm{nb}(y; \mathrm{dc}(y, 2^{3i} - 1; c)), x_i)]. \end{aligned}$$

Again, $\mathrm{init}_M \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, and, for $y, \bar{x}, c$ with $|y| \geq r, |c|, |x_i|$, for $1 \leq i \leq n$, we have that $\mathrm{init}_M(y; \bar{x}, c) = 1$ just if c is the coding of the start configuration of machine M with respect to input $\bar{x}$.

5. Let

$$\mathrm{acc}_M(y; c) = \mathrm{cf}_M(y; c) \wedge =(y; \mathrm{first}(y; c), 1).$$

Then $\mathrm{acc}_M \in [B_0; \mathrm{SC}, \mathrm{SRN}]$, and, for y, c with $|y| \geq r, |c|$, $\mathrm{acc}_M(y; c) = 1$ exactly if c is the code of an accepting configuration.

6. Finally, we can define the Turing predicate. Set

$$T_M(y; \bar{x}, c) = \mathrm{comp}_M(y; c) \wedge \mathrm{init}_M(y; \bar{x}, \mathrm{first}(y; c)) \wedge \mathrm{acc}_M(y; \mathrm{last}(y; c)).$$

Now we can prove Proposition 3.6. Let $f \in \mathbf{NPMV}$. Then there is some nondeterministic Turing machine M that computes f in time p. Set

$$T(z; \bar{x}, c) = \mathrm{cond}(; T_M(z; \bar{x}, c), 1, 0)$$

and

$$\mathrm{res}(z; b) = \mathrm{nb}(z; \mathrm{dc}(z, 2^{3n+3m}; \mathrm{last}(z; b))).$$

If $f(\bar{x}) \mapsto y$, there exists an accepting computation of M of length at most $p(|\bar{x}|)$ with output y. Let c be its Gödel number, and assume there were numbers z, d with $|z| \geq |c|$ and $d \sqsubset c$ such that $T(z; \bar{x}, d) \bmod 2 = 0$. Then d is the Gödel number of an

accepting computation of M on input $\bar{x}$, which means that the sequences coded by c and d have the same first element. This is impossible since the choice of our sequence encoding $d \sqsubset c$ implies that the sequence coded by d is a proper final segment of the sequence coded by c. Thus $c = \mu^c b.\, T(z; \bar{x}, b) \bmod 2 = 0$ for all z with $|z| \geq |c|$.

During the computation, each head can visit at most $p(|\bar{x}|)$ cells. Thus the length of the code of every configuration in this computation is bounded by $2[2(n + m + 1)p(|\bar{x}|) + 2|r| + 3n + 3m + 1]$. Note that there are $3n + 3m + 1$ commas in each configuration. Moreover, the coding of both the words on the tapes as well as the configuration requires multiplication by 2. Set

$$q(|\bar{x}|) = 4(n + m + 1)p(|\bar{x}|)^2 + (4|r| + 6n + 6m + 3)p(|\bar{x}|).$$

Then $|c| \leq q(|\bar{x}|)$. As has been shown in [1, 3], one can easily construct a function $\mathrm{bnd} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$ such that $q(|\bar{x}|) \leq |\mathrm{bnd}(\bar{x};)|$. It follows that $\mathrm{last}(\mathrm{bnd}(\bar{x};); c)$ is the coding of the last configuration in the computation coded by c, and hence $\mathrm{res}(\mathrm{bnd}(\bar{x};); c)$ is its result.

This shows that

$$f(\bar{x}) \subseteq \mathrm{res}(\mathrm{bnd}(\bar{x}); \mu^c b.\, T(\mathrm{bnd}(\bar{x}); \bar{x}, b) \bmod 2 = 0).$$

The converse inclusion is obvious by our general assumption of M and the construction of the functions T and res.

5. Conclusion. Bellantoni showed in his dissertation [1, 2] that when unbounded minimization is allowed over safe positions in the presence of safe recursion on notation, the resulting functions lie in the polynomial hierarchy. Verifying the minimality of a certain number a requires us to test whether all smaller numbers satisfy the given property. There are exponentially many such numbers with respect to the length of a. Therefore, it is quite obvious that the polytime functions are not closed under this operation.

In the present paper, we modified Bellantoni's minimization operator by requiring a weaker kind of minimality: only the numbers with a binary representation that is a proper prefix of the binary representation of a have to be tested. There are only polynomially many such numbers with respect to the length of a, which implies that the minimality test can be performed in polynomial time. However, now there may be several such minimal numbers a: the resulting function is multivalued.

The characterization derived in the paper is partly based on a normal form theorem which is similar to Kleene's normal form theorem for the computable functions [9]. Here it was shown that the nondeterministic polytime multifunctions can be generated from the deterministic polytime functions by an application of a weakened unbounded minimization operator. Kleene's theorem says that the computable functions can be generated by an application of full unbounded minimization from the primitive recursive ones.

So there is a certain similarity between the computable functions and the nondeterministic polytime functions as well as the primitive recursive functions and the deterministic polytime functions. Danner and Pollett [6] give more examples for this correspondence.

The minimization operator introduced in this paper is a partial operator: it can only be applied to multifunctions that are given together with a companion. To check the property of being a companion seems not to be an easy task, in general. It remains an open question whether the nondeterministic polytime multifunctions can also be generated by using only *total*, fully "implicit" operators.

In order to derive Proposition 3.9, it would be sufficient to require that the condition given in the definition of a companion holds only for those arguments which are used in the verification process. However, then the modified requirement would be part of the minimality condition and hence would have to be tested in polynomial time. It is not clear how this could be done in the case of nondeterministic polytime multifunctions. The situation is similar to the one in classical computability theory, where one has to require that the function minimized over is defined on all arguments used in the minimality verification. Dropping this condition leads to a minimization operator under which the class of computable functions is no longer closed [10].

Appendix. As is well known, there are codings of finite sequences such that the corresponding decodings are in **FP**. By Bellantoni and Cook's theorem, this means that they can be defined from initial functions in B_0 with the help of safe composition and safe recursion on notation. However, in this case, all of their parameters appear in the normal position, whereas here we need them in the safe position. As we will see, this can only be achieved in such a way that, for any input, the entire output of the function does not have the properties we are interested in, but certain of its low-order bits do. Moreover, there will be an additional normal parameter, the length of which is an upper bound on the low-order bits that may be used.

In what follows, we use a coding of finite sequences of numbers, which is essentially due to Buss [4]. The code or *Gödel number* of a sequence $a_1, \dots, a_k$ is constructed as follows. First replace each a_i by $a_i 1$, and write the resulting numbers in binary notation so that we have a string of 0's, 1's, and commas. Then write the string in reverse order, and replace each 0 by "10," each 1 by "11," and each comma by "00." The resulting string of zeros and ones is the binary representation of the Gödel number $\langle a_1, \dots, a_k \rangle$. For example, the Gödel number of 0,1,2 is $(1110110011110011)_2$ or 60,659. The empty sequence has the Gödel number $\langle\rangle = 0$.

Let us show now that the functions testing whether a number is the Gödel number of a finite sequence and, if this is the case, computing the length of the sequence as well as its ith element for given i, are in $[B_0; \mathrm{SC}, \mathrm{SRN}]$. To this end, we need several helper functions.

1. Let the function P be as in Example 2.8, and define

$$\begin{aligned} &\mathrm{rev}(0; a) = 0, \\ &\mathrm{rev}(xi; a) = \mathrm{cond}(; P(x; a), s_0(; \mathrm{rev}(x; a)), s_1(; \mathrm{rev}(x; a))) \qquad \text{for } xi \neq 0. \end{aligned}$$

Then the binary representation of $\mathrm{rev}(y; x)$ corresponds to the $|y|$ low-order bits of x written in reverse order.

2. Let

$$\begin{aligned} f_0(0; a, b) &= 1, \\ f_0(zi; a, b) &= \mathrm{cond}(; f_0(z; a, b), 0, \mathrm{cond}(; p(; f_0(z; a, b)), \\ &\qquad \mathrm{cond}(; P(z; a), \mathrm{cond}(; P(z; b), 1, 3), \mathrm{cond}(; P(z; b), 0, 1)), 3)) \end{aligned}$$

for $zi \neq 0$, and set

$$\leq(z; a, b) = \mathrm{cond}(; f_0(z; \mathrm{rev}(z; a), \mathrm{rev}(z; b)), 0, 1).$$

In case that $|z| \geq |a|, |b|$, we have that $\leq(z; a, b) = 1$ if $a \leq b$ and $\leq(z; a, b) = 0$ otherwise.

3. Set $\hat{s}_1(z;) = s_1(;z)$, let

$$\begin{aligned} f_1(0;w) &= w, \\ f_1(yi;w) &= \mathrm{cond}(;f_1(y;w), \mathrm{cond}(;p(;f_1(y;w)), f_1(y;w), p^2(;f_1(y;w))), \\ &\qquad p^2(;f_1(y;w))) \end{aligned}$$

for $yi \neq 0$, and define

$$\mathrm{truncate}(y;w) = p^2(;f_1(\hat{s}_1(y;);w)).$$

If w is the Gödel number of a sequence $a_1, \dots, a_k$ and $|a_1| \leq |y|$, then $\mathrm{truncate}(y;w) = \langle a_2, \dots, a_k \rangle$.

4. Let $\mathrm{sg}(z;) = {\leq}(z;z,0)$, and set

$$\begin{aligned} \mathrm{tr}(0,y;w) &= w, \\ \mathrm{tr}(zi,y;w) &= \mathrm{cond}(;\mathrm{sg}(z;), \mathrm{truncate}(y;\mathrm{tr}(z,y;w)), w) \qquad \text{for } zi \neq 0. \end{aligned}$$

For z with $|z| > 0$, we have that $\mathrm{tr}(z,y;w) = \mathrm{truncate}^{|z|-1}(y;w)$.

5. Set

$$\begin{aligned} \mathrm{ebit}(x;a) &= \mathrm{cond}(;P(x;P(x;a)),0,1), \\ \mathrm{obit}(x;a) &= \mathrm{cond}(;p(;P(x;P(x;a))),0,1). \end{aligned}$$

Then $\mathrm{ebit}(x;a)$ and $\mathrm{obit}(x;a)$, respectively, are the coefficients of $2^{2|x|}$ and $2^{2|x|+1}$ in the binary expansion of a. Moreover, let $P'(x,y;) = P(x;y)$, and define

$$\begin{aligned} f_2(0,z;w) &= 0, \\ f_2(yi,z;w) &= \mathrm{cond}(;f_2(y,z;w), \mathrm{cond}(;\mathrm{ebit}(y;w), \\ &\qquad \mathrm{cond}(;\mathrm{obit}(y;w), \mathrm{cond}(;\mathrm{sg}(P'(y,z;);),1,0),0),0),1) \end{aligned}$$

for $yi \neq 0$. If w is as above and $|y| > 1 + |a_1|$, then $f_2(y,z;w) = 1$ if $1 + |a_1| < |z|$ and $f_2(y,z;w) = 0$ otherwise.

6. Define

$$\begin{aligned} f_3(0,y;w) &= 0, \\ f_3(zi,y;w) &= \mathrm{cond}(;f_2(y,\hat{s}_1(z;);w), \\ &\qquad \mathrm{cond}(;\mathrm{ebit}(z;w), s_0(;f_3(z,y;w)), s_1(;f_3(z,y;w))), f_3(z,y;w)) \end{aligned}$$

for $zi \neq 0$, and set

$$\mathrm{first}(y;w) = p(;f_3(\hat{s}_1^2(y;),\hat{s}_1^2(y;);w)).$$

If w is as above again and $|y| \geq |a_1|$, then $\mathrm{first}(y;w) = a_1$.

7. Finally, let

$$\begin{aligned} f_4(0,y;w) &= 0, \\ f_4(zi,y;w) &= \mathrm{cond}(;\mathrm{tr}(z,y;w), f_4(z,y;w), \mathrm{cond}(;\mathrm{tr}(\hat{s}_1(z;),y;w),z,1)) \end{aligned}$$

for $zi \neq 0$. Set

$$\mathrm{lth}(y;w) = f_4(\hat{s}_1(y;),\hat{s}_1(y;);w)$$

and

$$\mathrm{dc}(y,z;w) = \mathrm{first}(y;\mathrm{tr}(\hat{s}_1(z;),y;w));$$

then $\mathrm{lth}, \mathrm{dc} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$. Moreover, for w as above and y such that $|y| \geq k, |a_j|$ for $1 \leq j \leq k$, we have that $\mathrm{lth}(y;w) = k$ and $\mathrm{dc}(y,z;w) = a_{|z|+1}$ as long as $|z| < \mathrm{lth}(y;w)$.

If we want to compute the last element of the sequence coded by w, we cannot do this by composing the functions dc and lth because of the restriction in the definition of safe composition. Let, therefore,

$$<(z;a,b) = \mathrm{cond}(;\leq(z;a,b),0,\mathrm{cond}(;\leq(z;b,a),1,0)),$$

and set

$$f_5(0,z;w) = w,$$
$$f_5(yi,z;w) = \mathrm{cond}(;<(z;|s_1(;y)|,\mathrm{lth}(z;w)),f_5(y,z;w),\mathrm{truncate}(z;f_5(y,z;w)))$$

for $yi \neq 0$. Now define

$$\mathrm{last}(z;w) = \mathrm{first}(z;f_5(z,z;w)).$$

Note that

$$|0| = 0, \qquad |xi| = s(|x|;) \qquad \text{for } xi \neq 0,$$

where

$$s(0;) = 1, \qquad s(x0;) = s_1(;x) \quad \text{for } x \neq 0, \qquad s(x1;) = s_0(;s(x;)).$$

Thus $\mathrm{last} \in [B_0; \mathrm{SC}, \mathrm{SRN}]$. Moreover, if w is the Gödel number of a finite sequence and $|z| \geq |w|$, then $\mathrm{last}(z;w)$ is the last element of the sequence coded by w.

Observe that w is the Gödel number of a finite sequence exactly if the binary expansion of w satisfies the following four conditions.

(i) Either $w = 0$ or w is odd.

(ii) For every even j such that the coefficient of 2^{j-1} is not 0, the coefficient of 2^j is 1.

(iii) There is no substring of the form 1000.

(iv) There is no substring of the form 0010.

Set

$$\mathrm{seq1}(0;w) = 1,$$
$$\mathrm{seq1}(zi;w) = \mathrm{cond}(;\mathrm{seq1}(z;w),0,\mathrm{cond}(;\mathrm{ebit}(z;w),1,\mathrm{cond}(;\mathrm{obit}(z;w),0,1)))$$

for $zi \neq 0$,

$$\mathrm{seq2}(0;w) = 1,$$
$$\mathrm{seq2}(zi;w) = \mathrm{cond}(;\mathrm{seq2}(z;w),0,\mathrm{cond}(;P(z;w),\mathrm{cond}(;P(\hat{s}_1(z;);w),$$
$$\mathrm{cond}(;P(\hat{s}_1^2(z;);w),\mathrm{cond}(;P(\hat{s}_1^3(z;);w),1,0),1),1),1))$$

for $zi \neq 0$, and

$$\mathrm{seq3}(0;w) = 1,$$
$$\mathrm{seq3}(zi;w) = \mathrm{cond}(;\mathrm{seq3}(z;w),0,\mathrm{cond}(;P(z;w),\mathrm{cond}(;P(\hat{s}_1(z;);w),1,$$
$$\mathrm{cond}(;P(\hat{s}_1^2(z;);w),\mathrm{cond}(;P(\hat{s}_1^3(z;);w),0,1),1)),1))$$

for $zi \neq 0$. Moreover, let $a \wedge b = \mathrm{cond}(;a,0,\mathrm{cond}(;b,0,1))$, and define

$$\mathrm{seq}(z;w) = \mathrm{cond}(;w,{\leq}(z;w,0),1) \wedge \mathrm{seq1}(z;w) \wedge \mathrm{seq2}(z;w) \wedge \mathrm{seq3}(z;w).$$

Then $\mathrm{seq} \in [B_0;\mathrm{SC},\mathrm{SRN}]$, and for z,w with $|z| \geq |w|$, we have that $\mathrm{seq}(z;w) = 1$ if w is the Gödel number of a finite sequence and $\mathrm{seq}(z;w) = 0$ otherwise.

Acknowledgment. The author is grateful to the referees for a careful reading and the improvement suggestions.

REFERENCES

[1] S. Bellantoni, *Predicative Recursion and Computational Complexity*, Ph.D. thesis, University of Toronto, Toronto, Canada, 1992.
[2] S. Bellantoni, *Predicative recursion and the polytime hierarchy*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, 1995, pp. 15–29.
[3] S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the polytime functions*, Comput. Complexity, 2 (1992), pp. 97–110.
[4] S. R. Buss, *Bounded Arithmetic*, Bibliopolis, Naples, 1986.
[5] A. Cobham, *The intrinsic computational difficulty of functions*, in Logic, Methodology and Philosophy of Science II, Y. Bar-Hillel, ed., North–Holland, Amsterdam, 1965, pp. 24–30.
[6] N. Danner and C. Pollett, *Minimization and NP multifunctions*, Theoret. Comput. Sci., to appear.
[7] D. Leivant, *Subrecursion and lambda representation over free algebras (preliminary summary)*, in Feasible Mathematics, S. Buss and P. Scott, eds., Birkhäuser Boston, Boston, 1990, pp. 281–291.
[8] D. Leivant, *Ramified recurrence and computational complexity* I: *Word recurrence and polytime*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, 1995, pp. 320–343.
[9] P. Odifreddi, *Classical Recursion Theory*, North–Holland, Amsterdam, 1989.
[10] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw–Hill, New York, 1967.
[11] H. Simmons, *The realm of primitive recursion*, Arch. Math. Logic, 27 (1988), pp. 177–188.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1557–1570

TESTING MEMBERSHIP IN LANGUAGES THAT HAVE SMALL WIDTH BRANCHING PROGRAMS*

ILAN NEWMAN†

Abstract. Combinatorial property testing, initiated formally by Goldreich, Goldwasser, and Ron in [*J. ACM*, 45 (1998), pp. 653–750] and inspired by Rubinfeld and Sudan [*SIAM J. Comput.*, 25 (1996), pp. 252–271], deals with the following relaxation of decision problems: Given a fixed property and an input x, one wants to decide whether x has the property or is "far" from having the property.

The main result here is that, if $\mathcal{G} = \{g_n : \{0,1\}^n \to \{0,1\}\}$ is a family of Boolean functions which have oblivious read-once branching programs of width w, then, for every n and $\epsilon > 0$, there is a randomized algorithm that always accepts every $x \in \{0,1\}^n$ if $g_n(x) = 1$ and rejects it with high probability if at least ϵn bits of x should be modified in order for it to be in $g_n^{-1}(1)$. The algorithm makes $(\frac{2^w}{\epsilon})^{O(w)}$ queries. In particular, for constant ϵ and w, the query complexity is $O(1)$.

This generalizes the results of Alon et al. [*Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 1999, pp. 645–655] asserting that regular languages are ϵ-testable for every $\epsilon > 0$.

Key words. property testing, randomized algorithms, branching programs

AMS subject classifications. 68Q15, 68Q10

PII. S009753970038211X

1. Introduction.

Combinatorial property testing, initiated formally by Goldreich, Goldwasser, and Ron in [11] and inspired by Rubinfeld and Sudan [16], deals with the following relaxation of decision problems: Given a fixed property and an input x, one wants to decide whether x has the property or is "far" from having the property. A property here is a set of binary strings (those inputs that have the "property") and is identified with its characteristic function (that is, "1" on all inputs that have the property and "0" elsewhere). Being "far" is measured by the number of bits that need to be changed for an input x in order for it to have the property (i.e., the Hamming distance). A property is said to be (ϵ, q)-testable if there is a randomized algorithm that, for every input, $x \in \{0,1\}^n$ queries at most q bits of x and with probability $2/3$ distinguishes between the case when x has the property and the case when x is ϵn-far from having the property. Varying ϵ and n may result in different algorithms with different query complexity $q = q(\epsilon, n)$ that may depend on both ϵ and n. If, for a fixed $\epsilon > 0$ and every large enough n, a property P is (ϵ, q)-testable with a number of queries q that is independent of the length of the input, n, then we say that P is ϵ-testable. If, for every $\epsilon > 0$, P is ϵ-testable, then P is said to be *testable*.

Apart from being a natural relaxation of the standard decision problem, combinatorial property testing emerges naturally in the context of probably approximately correct (PAC) learning, program checking [10, 6, 16], probabilistically checkable proofs [3], and approximation algorithms [11].

In [11], the authors mainly consider graph properties and show (among other

*Received by the editors December 5, 2000; accepted for publication (in revised form) February 18, 2002; published electronically August 5, 2002. A preliminary version of this paper appeared in *Proceedings of the 41st Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 251–258.

http://www.siam.org/journals/sicomp/31-5/38211.html

†Department of Computer Science, University of Haifa, Haifa 31905, Israel (ilan@cs.haifa.ac.il).

things) the quite surprising fact that the graph property of being bipartite is testable. They also raise the question of obtaining general results identifying classes of properties that are testable. Some interesting examples are given in [11], and several additional ones can be obtained by applying the regularity lemma [1]. Alon et. al. [2] proved that membership in any regular language is testable, hence obtaining a general result identifying a nontrivial class of properties, each being testable. Here we further pursue this direction: We prove that if a language has a (nonuniform) oblivious read-once branching program (BP) of width w, then it is $(\epsilon, (\frac{2^w}{\epsilon})^{O(w)})$-testable. In particular, this shows that every family of functions that can be defined by a nonuniform collection of constant width oblivious read-once BPs is testable. This also generalizes and gives an alternative proof and algorithm for the result of [2], as regular languages can be represented by constant width oblivious read-once BPs. We note, however, that the dependence of the query complexity here is worse than in [2].

A BP of width w is a deterministic *leveled* BP in which every level contains at most w vertices. In what follows, we will be interested in BPs of width w that have the further restriction of being *oblivious read-once*. Namely, every level is associated with a variable (all nodes in a level query the same variable), and each variable appears in at most one level. BPs have been extensively studied as a model of computation for Boolean functions. ([7] contains a survey text; see also [4, 5, 13] for a partial list of different aspects involving BPs and read-once BPs.)

The size of a BP (and a read-once BP) is tightly related to the space complexity of the function it computes: If a language is in $SPACE(s)$, then it has a BP of total size at most $n \cdot 2^{O(s)}$ [8] and also a read-once BP of width $2^{O(2^s)}$ [12]. However, the inverse of the last assertion is not true even for computable languages. The result of [2] and the result here, in its uniform manifestation, may be viewed as asserting that "very small" space functions are "efficiently" testable: All regular languages are in $SPACE(O(1))$ and hence have a read-once BP of $O(1)$ size. What happens for $SPACE(\omega(1))$ functions? It is known that $SPACE(O(1)) = SPACE(o(\log\log n))$ = Regular [12]. Hence the above question is interesting for $SPACE(s)$ with $s = \Omega(\log\log n)$. The result here says nothing directly for $s = \Omega(\log\log n)$. However, we get rid of the strong "uniformity" of the deterministic finite automata (DFAs) used in [2]. In regular languages, the same finite automaton is used to test all the words, even of different lengths. On the other hand, when represented by a family of BPs, each BP computes the characteristic function of the property for a given input length. There are languages of arbitrary complexity that can be represented by $O(1)$-width oblivious BPs. Our results apply to such cases as well. This includes the family of $O(1)$-terms disjunctive normal form (DNF), $O(1)$-clauses conjunctive normal form (CNF), and some other interesting examples (see section 4).

Finally, we note that $SPACE(O(\log n))$ functions are not testable in general; [2, 11, 15] contain lower bounds showing that some functions in $SPACE(O(\log n))$ are not ϵ-testable and sometimes not even (ϵ, n^δ)-testable for some fixed $\epsilon, \delta < 1$. However, the question of whether properties in $SPACE(s)$ for $\log\log n \le s << \log n$ are "efficiently" testable is open. In particular, we do not have any candidate for a $SPACE(O(\log\log n))$ function whose ϵ-testing requires $n^{\Omega(1)}$ queries for some fixed $\epsilon > 0$.

2. Definitions and notation. We identify properties with the collection of their characteristic Boolean functions, namely: A property $\mathcal{P} \subseteq \{0,1\}^*$ is identified with $\{f : \{0,1\}^n \longrightarrow \{0,1\}\}$ so that $f(x) = 1$ if and only if $x \in \mathcal{P}$.

An oblivious leveled BP is a directed graph B in which the nodes are partitioned into levels $L_0, \ldots, L_m$. There are two special nodes: a "start" node belonging to L_0 and an "accept" node belonging to L_m. Edges are going only from a level to nodes in the consecutive level. Each node has at most two out-going edges, one of which is labeled by "0" and the other by "1." In addition, all edges in between two consecutive levels are associated with a member of $\{1, \ldots, n\}$ (a Boolean variable). An input $x \in \{0,1\}^n$ naturally defines a path starting at the *start*-node: At each step, if the edges are associated with i, then the edge with the label identical to the value of x_i is chosen. A leveled BP defines a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$ in the following way: $g(x) = 1$ if the path that x defines reaches *accept*. This definition of BPs is essentially equivalent to what is sometimes called "deterministic" BPs (as each input defines at most one path from each node). However, note that this definition is slightly different from the standard definition of deterministic BPs, in which every vertex has exactly two outgoing edges; one is labeled by "1" and the other by "0." Here, instead, an input x can be "stuck" at an internal node v due to the fact that v has just one outgoing edge that is associated with i and is labeled by a value that is opposite to that of x_i. (This cannot happen in the standard definition.) A leveled BP is of width w (w-width) if its largest level contains w nodes.

An *oblivious read-once* BP computing $g : \{0,1\}^n \longrightarrow \{0,1\}$ is a leveled BP with the additional property that edges ending in distinct levels are labeled with distinct variables. This implies also that there are exactly $n+1$ levels (for a function that depends on all its n variables). We number the levels of the BP from 0 (containing the start s) and on and associate to the edges in between levels the formal Boolean variables $X_1, \ldots, X_n$ consecutively (by possibly renaming the variables). We may assume that the last level is numbered by n.

In what follows, we consider only oblivious read-once BPs. For a given BP, B, and two nodes u, v, we define $B[u : v]$ the (sub) BP for which its start node is u and its accept node is v. If $u \in L_i$ and $v \in L_j$, then $B[u : v]$ computes a Boolean function on the variables $X_i, \ldots, X_j$. The length of $B[u : v]$ in this case is $\nu = j - i$. Such $B[u : v]$, as a subprogram of a read-once oblivious BP, is also read-once oblivious BP. When discussing such a BP $B[u : v]$, we renumber its levels so that its first level, which is level L_i in B, is denoted $L_0(B[u : v])$, and its last level is denoted by $L_\nu(B[u : v])$. When it is clear from the context which is the BP that is considered, we just refer to its first and last levels as L_0, L_ν, respectively (where ν is the length of the corresponding BP).

We will be interested in BPs for which the start and accept nodes are not always defined. Namely, the BP B might have multiple nodes in its first and last levels. For such a BP of length n, any choice of start and accept nodes $(s, t) \in L_0 \times L_n$ defines a different function on n variables. If no path from a node $v \in B$ reaches the last level, then deleting v from B will not change the function that B computes for any choice of start and accept nodes in the first and last levels. Similarly, we may delete every vertex that can be reached from no vertex of the first level. Also, when we talk about $B[u : v]$, for some specific nodes u, v, we may delete any node from $B[u : v]$ that either is not reachable from u or cannot reach v. In particular, this means that u is the only node in $L_0(B[u : v])$, and v is the only node in the last level of $B[u : v]$. Such nodes that can be deleted from the BP are called "unnecessary nodes." In what follows, we always assume that all BPs under discussion contain no "unnecessary nodes."

For integers $a < b$, we denote by $B_{a:b}$ the subprogram of B containing all nodes in levels $L_a, L_{a+1}, \ldots, L_b$. $B_{a:b}$ has undefined source and sink. Note that, if B is an

oblivious read-once BP of width w, then, for any two nodes u, v and any two numbers a and b, $B[u:v]$ and $B_{a:b}$ are oblivious read-once BPs of width at most w. (The width can become smaller as nodes might become "unnecessary.")

Let $x, y \in \{0,1\}^n$; we define $dist(x,y) = hamming(x,y) = |\{i|\ x_i \neq y_i\}|$. Let $g : \{0,1\}^n \longrightarrow \{0,1\}$ such that $g^{-1}(1) \neq \phi$; we define $dist(x,g) = \min\{dist(x,y)|\ y \in g^{-1}(1)\}$. For a BP B and two nodes u and v in levels L_i, L_j, respectively, let $dist(x, B[u:v]) = dist(x[i,j], g')$, where g' is the function computed by $B[u:v]$ on the formal variables $X_i, X_{i+1}, \ldots, X_j$.

Let B be an oblivious read-once BP with fixed start and accept nodes that computes a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$. A randomized algorithm $\mathcal{A}$ is a 1-sided error ϵ-test for B (g) of query complexity $c(\mathcal{A})$ if, for every input $x \in \{0,1\}^n$, it queries at most $c(\mathcal{A})$ queries and

1. for every input $x \in g^{-1}$, the algorithm accepts;
2. for every input $x \in \{0,1\}^n$ for which $dist(x,g) \geq \epsilon n$, the algorithm rejects with probability at least 2/3.

Let $\mathcal{B}_w^n$ be the set of all oblivious read-once BPs of width w and length n. For $B \in \mathcal{B}_w^n$, we denote $\tilde{c}(\epsilon, B) = \min\{c(\mathcal{A}) :\ \mathcal{A}$ is a 1-sided error ϵ-test for $B\}$. Namely, $\tilde{c}(\epsilon, B)$ is the query complexity of the best 1-sided error ϵ-test for B. Let $\tilde{q}(\epsilon, w) = \max\{\tilde{c}(\epsilon, B) :\ B \in \mathcal{B}_w^n\}$. Namely, $\tilde{q}(\epsilon, w)$ is the worst query complexity needed to ϵ-test a w-width BP. Formally, $\tilde{q}(\epsilon, w)$ is a function of n too; however, as we shall see, asymptotically this is not the case.

Finally, in what follows, for ease of notation, we neglect taking $\lceil\ \rceil$ and $\lfloor\ \rfloor$ for numbers, even when they need to be integers, whenever this is clear from the context and has no bearing on the essence of proofs.

3. Results. Our main result is the following.

THEOREM 1. *Let $g : \{0,1\}^n \longrightarrow \{0,1\}$ be computed by an oblivious read-once BP of width w. Then there is an ϵ-test for g that makes $(\frac{2^w}{\epsilon})^{O(w)}$ queries.*

COROLLARY 3.1. *If $g : \{0,1\}^n \longrightarrow \{0,1\}$ has a read-once BP of width $w = O(1)$, then g is testable.*

The proof of Theorem 1 uses several reduction steps in order to reduce testing of a w-width BP to testing of $(w-1)$-width BPs. This approach has prospects since 1-width BPs are testable, as asserted by the following proposition.

PROPOSITION 1. *If $g : \{0,1\}^n \longrightarrow \{0,1\}$ is computable by an oblivious read-once BP of width $w = 1$, then g is $(\epsilon, O(\frac{1}{\epsilon}))$-testable by a 1-sided error algorithm.*

Proof. We assume that g is not identically "0" and not identically "1," as otherwise the test is trivial (with no queries at all). Let B be a BP of width $w = 1$ computing the nonzero function g. It is clear from the definition that g is a one-term DNF. That is, written in formal variables, $X_1, \ldots, X_n$, $g = \Pi_{i=1}^n t_i$, where t_i is either X_i or $\bar{X}_i$. g does not necessarily depend on all of its variables; in this case, we just look at the variables it does depend on. Let x be an input such that $dist(g,x) \geq \epsilon n$ (assuming that there is such x). It is easy to see that, for at least $\epsilon \cdot n$ of the places $\{1, \ldots, n\}$, x_i is not consistent with t_i. Hence sampling $O(\frac{1}{\epsilon})$ bits of an input x and rejecting if x_i is inconsistent with t_i are guaranteed to succeed with probability 2/3 for ϵn-far inputs, and with probability 1 for any input x for which $g(x) = 1$. ∎

For the proof of the case $w \geq 2$, we will need some machinery developed hereafter.

3.1. Main definitions and some intuition. The algorithm for ϵ-testing w-width BPs will be recursive on the width. Namely, our aim is to reduce ϵ-testing of a w-width BP to testing of $(w-1)$-width BPs with possibly a smaller ϵ. Key notions are that of *r-full* levels and decomposable BPs. They are defined below.

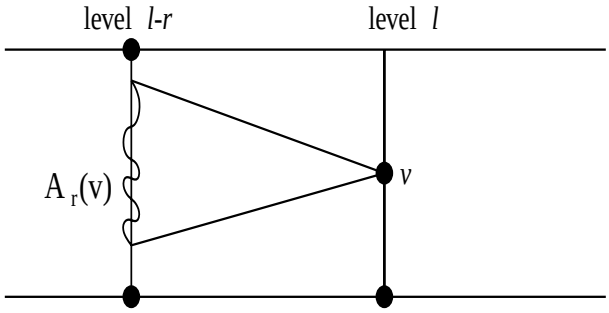


FIG. 1. *$A_r(v)$ is the set of nodes in level L_{l-r} that can reach v. Here v is* not *r-full as not all nodes in level L_{l-r} can reach v.*

For an integer r and a node v in a BP, we denote

$$A_r(v) = \{u | \text{ there is a path of length } r \text{ from } u \text{ to } v\}.$$

See Figure 1.

DEFINITION 3.2. *Let v be a vertex in level L_l of a BP with start and accept nodes that are not necessarily defined. We say that v is* r-full *if $A_r(v)$ contains all nodes of level L_{l-r}. If every vertex in level L_l is r-full, then L_l is said to be r-full.*

Namely, a vertex v is r-full if v is reachable from every vertex of level L_{l-r}; see Figure 1. Note that, for every two nodes u and v of a BP B, v is always 1-full with respect to $B[u : v]$. This is due to the fact that $B[u : v]$ contains no "unnecessary nodes."

FACT 1. *Assume that $v \in L_l$ is r-full for a certain r and l; then*

- *v is r'-full for every $r' > r$;*
- *if $u \in L_{l+1}$ is a neighbor of v, then u is $(r+1)$-full.*

Proof. For the first part, assume that $r' > r$ and $v' \in L_{l-r'}$. Then, as we assume that there are no "unnecessary vertices," v' can reach some vertex $v'' \in L_{l-r}$. In turn, v'' can reach v by the assumption that v is r-full. Hence v' can reach v.

For the second part, if v is r-full, it can be reached from any $w \in L_{l-r}$. Since u is a neighbor of v, it can also be reached by every $w \in L_{l-r}$. □

The following is a crucial ingredient for the rest of what follows.

DEFINITION 3.3. *Let $\delta < 1$. A BP of length ν, with start and accept nodes that are not necessarily defined, is said to be* δ-decomposable *if, for some $\frac{\delta\nu}{20} \le \ell \le \nu - 1$ and $r \le \lfloor \frac{\delta\ell}{10} \rfloor$, L_ℓ is r-full.*

For a given BP, B, the role of the non δ-decomposable subprogram of B is the following: We first show in section 3.2 that, if B' is not δ-decomposable for $\delta < \epsilon$, then ϵ-testing B' can indeed be reduced to testing "narrower" BPs. Then, in section 3.3, we show how a general BP can be decomposed into disjoint nondecomposable subprograms such that testing the BP can be reduced to testing not too many of the nondecomposable parts of it.

3.2. Testing nondecomposable BPs. The following lemma, which is the main technical part of the proof of Theorem 1, relates testing w-width nondecomposable BPs to the test of general $(w-1)$-width BPs.

LEMMA 3.4. *Let $\delta \le \epsilon$, and let B be a non δ-decomposable BP of width w and length n. Then ϵ-testing $B[s : t]$, for any start and accept nodes s and t, requires at most $O(\frac{w^4}{\delta^3}(\log \frac{w^2}{\delta})^2) \cdot \tilde{q}(0.8\epsilon, w-1)$ queries.*

Proof. The idea of the proof is as follows: We fix $O(\frac{1}{\delta^2})$ levels that are equally spaced in B, leaving out enough space in the beginning of B. The assumption that B is not δ-decomposable will imply that, for each of two nodes u, v in the levels we choose, the test of $B[u : v]$ can be reduced to tests of $(w-1)$-width BPs. We then show how to combine the results of the tests on $B[u : v]$ for all such u, v into an ϵ-test for B.

Formally, let $m = \lceil \frac{\delta^2 n}{400} \rceil$. Let $\{l_0, \ldots, l_p\}$ be the set of numbers that are m apart, starting from $\lceil \frac{20m}{\delta} \rceil$ and ending at or before n. Namely, $l_i = \lceil \frac{20m}{\delta} \rceil + i \cdot m$, $i = 0, 1, \ldots, p = \lfloor \frac{n-l_0}{m} \rfloor = O(\frac{1}{\delta^2})$. Let $\mathcal{S} = L_{l_0} \times \cdots \times L_{l_p}$. Our first aim is to show that, for every pair $(u, v) \in L_{l_i} \times L_{l_{i+1}}$, the ϵ_1-test of $B[u : v]$ can be reduced to a small number of general tests of $(w-1)$-width BPs.

We first need the following claims.

CLAIM 3.5. *For every $l \geq l_0$, level L_l is not $(2m)$-full.*

Proof. The proof is immediate from the choice of parameters and the fact that B is not δ-decomposable. ∎

For each l such that $l_i < l \leq l_{i+1}$, let $F(l)$ be the set of all $(l - l_i)$-full vertices in level L_l. In other words, $v \in F(l)$ if it is in the lth level and it is reachable from every vertex of the l_ith level. By our assumption on B, $F(l) \neq L_l$, as otherwise L_l would be $(l - l_{i-1})$-full in contradiction with Claim 3.5.

Hence the above implies the following claim.

CLAIM 3.6. *Let u, v be vertices in levels $L_{l_i}, L_{l_{i+1}}$, respectively, and let l be such that $l_i \leq l \leq l_{i+1}$.*

- *Let u' be in level L_l, and assume that $u' \notin F(l)$; then $B[u : u']$ is of width $w' \leq w - 1$.*
- *Let v' be in level L_l, and assume that $v' \in F(l)$; then $B[v' : v]$ is of width $w' \leq w - 1$.*

Proof. Let $u' \notin F(l)$ be in level L_l. As $u' \notin F(l)$, u' is not $(l - l_i)$-full; then, by Fact 1, it is also not $(l - l')$-full for every $l' > l_i$. Namely, for every intermediate level $L_{l'}$, $l_i < l' < l$, there is a vertex that cannot reach u' and hence can be deleted from $B[u : u']$.

For the second part, assume first that v' is in level L_l for $l > l_i$ and $v' \in F(l)$. Let t be any node at level $L_{l'}$, $l < l' \leq l_{i+1}$, that is reachable from v'. Since $v' \in F(l)$, it follows that t is $(l' - l_i)$-full. Hence not all vertices in level $L_{l'}$ are reachable from v', as otherwise level $L_{l'}$ will be $(l' - l_i)$-full, in contradiction to Claim 3.5. As this is true for every $l < l' \leq l_{i+1}$, it follows that $B[v' : v]$ is of width $w' \leq w - 1$. If v' is in level L_{l_i}, then the same argument for t will work except that t will be $(l' - l_{i-1})$-full. Again, this implies that level $L_{l'}$, $l_i < l' \leq l_{i+1}$, cannot have all of its nodes reachable from v'. Otherwise, it would be $(l' - l_{i-1})$-full, in contradiction to Claim 3.5. ∎

Claim 3.6 asserts that $B[v_i : v_{i+1}]$ is indeed of width of at most $(w-1)$ unless $v_i \notin F(l_i)$ and $v_{i+1} \in F(l_{i+1})$. We still need to deal with the case for which $v_i \notin F(l_i)$ and $v_{i+1} \in F(l_{i+1})$, where the subprogram $B[v_i : v_{i+1}]$ might be of width w. The key observation here is that any path from v_i to v_{i+1} must start at $L_{l_i} - F(l_i)$ (as v_i is such) and end in $F(l_{i+1})$. Hence this path must intersect $F(l)$ for some intermediate level L_l, $l_i < l \leq l_{i+1}$. In addition, by Fact 1, once it intersects $F(l)$, it intersects $F(l')$ for every $l' > l$ (see Figure 2). This suggests the following.

Let $k = \frac{10}{\delta}$; we choose $k+1$ numbers, $p_0, \ldots, p_k$, that are $\frac{m}{k}$ apart in the range $\{l_i, \ldots, l_{i+1}\}$: $p_j = l_i + j \cdot \frac{m}{k}$, $j = 0, \ldots, k$.

CLAIM 3.7. *For every $u \in L_{l_i} - F(l_i)$ and $v \in F(l_{i+1})$, the following hold:*

- *If $y \in \{0,1\}^n$ is such that $dist(y, B[u : v]) = 0$, then, for some $j \in \{1, \ldots, k\}$,*

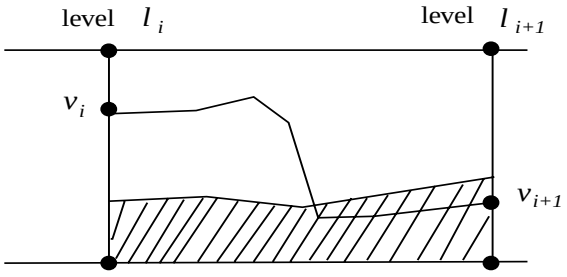


FIG. 2. *Vertices in shadowed area are in $F()$. If a path from v_i to v_{i+1} intersects $F(l)$ at some intermediate level L_l, then it intersects $F()$ for every following level.*

there are some $u' \in L_{p_{j-1}} - F(p_{j-1})$, $v' \in F(p_j)$ so that $dist(y, B[u : u']) = dist(y, B[v' : v]) = 0$.

- *If $y \in \{0,1\}^n$ is such that $dist(y, B[u : v]) \geq (1-\alpha)\epsilon m$ for some $\alpha < 1$, then, for every $j \in \{1, \ldots, k\}$ and for every $u' \in L_{p_{j-1}} - F(p_{j-1})$ and $v' \in F(p_j)$ such that u can reach u', u' can reach v', and v' can reach v,*

$$dist(y, B[u : u']) + dist(y, B[v' : v]) \geq (1-\alpha)\epsilon m - \frac{m}{k} \geq (0.9-\alpha)\epsilon m.$$

Proof. If $dist(y, B[u : v]) = 0$, then, by the discussion above, there is some level $l_i < l \leq l_{i+1}$ so that the path, $Path(y)$, that y defines from u to v intersects $F(l')$ for each $l \leq l' \leq l_{i+1}$ and does not intersect $F(l'')$ for each $l_i \leq l'' < l$. Let j be the smallest such that $p_j \geq l$. Let u' be the vertex that $Path(y)$ intersects in $L_{p_{j-1}}$, and let v' be the vertex that $Path(y)$ intersects in L_{p_j}. Clearly, for these j, u', v', the first part of the claim holds.

For the second part, assume that, for $y \in \{0,1\}^n$, there are $j \in \{1, \ldots, k\}$, $u' \in L_{p_{j-1}} - F(p_{j-1})$, and $v' \in F(p_j)$ such that u can reach u', u' can reach v', and v' can reach v, and such that $dist(y, B[u : v]) < (0.9-\alpha)\epsilon m$. Then, certainly, $dist(y, B[u : v]) < (1-\alpha)\epsilon m$: First, by changing at most $(0.9-\alpha)\epsilon m$ bits of y in the range $\{l_i + 1, \ldots, p_{j-1}\}$ and $\{p_j + 1, \ldots, l_{i+1}\}$, we can get a y' such that its corresponding parts (to the places above) traverse B from u to u' and from v' to v. Then, by changing possibly additional $m/k \leq 0.1\epsilon m$ bits, namely, all bits in the range $\{p_{j-1} + 1, \ldots, p_j\}$, we get a y'' that traverses B from u to v through u' and v'. ∎

We now can present the algorithm that $(1-\alpha)\epsilon$-tests $B[u : v]$ for each $(u, v) \in L_{l_i} \times L_{i+1}$, given that we have a general 1-sided error test for $(w-1)$-width BPs. Note that the length of $B[u : v]$ for any such u and v is m.

Algorithm $A_1((1-\alpha)\epsilon, w, B[u : v])$.
The first parameter is relative distance, the 2nd is width, and $(u, v) \in L_{l_i} \times L_{l_{i+1}}$.

1. If $u \in F(l_i)$ or $v \notin F(l_{i+1})$, then, by Claim 3.6, $B[u : v]$ is already of width $w' \leq w - 1$. This test is done by calling the general $(1-\alpha)\epsilon$-testing procedure for $(w-1)$-width BPs.
2. Otherwise, if $u \notin F(l_i)$ and $v \in F(l_{i+1})$, let $k = \frac{10}{\delta}$ and $\rho = 1 + \frac{\log(kw^2)}{\log 3}$. We choose in the range $\{l_i, \ldots, l_{i+1}\}$ $k+1$ numbers, $p_0, \ldots, p_k$, that are $\frac{m}{k}$ apart: $p_j = l_i + j \cdot \frac{m}{k}$, $j = 0, \ldots, k$.

   For every triplet $(j, u'v')$ such that $j \in \{1, \ldots, k\}$, $u' \in L_{p_{j-1}} - F(p_{j-1})$, $v' \in F(p_j)$, and such that u can reach u', u' can reach v' and v' can reach v, a $(0.9-\alpha)\epsilon$-test is performed ρ independent times on $B[u : u']$ and $B[v' : v]$. This is done by calling the general procedure for testing $(w-1)$-width BPs.
   If there is a triplet $(j, u'v')$ for which all ρ tests pass, then the outcome of A_1 is "Yes." Otherwise, if, for every triplet (j, u', v'), one or more of the ρ tests on either $B[u : u']$ or $B[v' : v]$ answer "No," then the outcome of A_1 is "No."

CLAIM 3.8. *Let B be a w-width BP that is not δ-decomposable, and let m, k be as above. Let $x \in \{0,1\}^n$ be any input; then Algorithm A_1 makes $O(\frac{w^2}{\delta} \cdot \log \frac{w^2}{\delta})$ calls for a general $(1-\alpha)\epsilon$-test of $(w-1)$-width programs on x and*

- *if $dist(x, B[u:v]) = 0$, then A_1 answers "Yes" on x with probability 1;*
- *if $dist(x, B[u:v]) \geq (1-\alpha)\epsilon m$, then the outcome of A_1 on x is "No" with probability at least 2/3.*

Proof. For each triplet (j, u', v') that is relevant to the second case of Algorithm A_1, Claim 3.6 asserts that $B[u:u']$ and $B[v':v]$ are of width at most $(w-1)$. Hence all calls of A_1 are to tests of $(w-1)$-width BPs. There are at most $O(k \cdot w^2) = O(\frac{w^2}{\delta})$ such triplets; hence the claim on the number of calls to $(w-1)$-width tests is obvious.

We assume that the general $(w-1)$-test is a 1-sided error. Let $x \in \{0,1\}^n$ be an input with $dist(x, B[u:v]) = 0$. A "No" result will be obtained if $B[u:v]$ is of width at most $w-1$ and the general $(w-1)$-width test answers "No" (1st case of A_1) or if, for every triplet (j, u', v') as above, one of the tests, to either $B[u:u']$ or $B[v':v]$, answers "No." Both cases occur with probability 0 by Claim 3.7.

Now let $x \in \{0,1\}^n$ be an input for which $dist(x, B[u:v]) \geq (1-\alpha)\epsilon m$. If $B[u:v]$ is of width $(w-1)$, then A_1 answers "Yes" only if the general $(w-1)$-width test errs. This occurs with probability at most 1/3. If $B[u:v]$ is of width w, then, by Claim 3.7, for every triplet (j, u', v') as above, $dist(y, B[u:u']) + dist(y, B[v':v]) \geq (0.9-\alpha)\epsilon m$. However, for each such triplet, either $dist(y, B[u:u']) \geq (0.9-\alpha)\epsilon \cdot \frac{j-1}{k} \cdot m$ or $dist(y, B[v':v]) \geq (0.9-\alpha)\epsilon \cdot (1-\frac{k-j}{k})m$. In any of these cases, a general $(0.9-\alpha)\epsilon$-test to the corresponding $(w-1)$-width BP would erroneously say "Yes" with probability at most 1/3. Since there are ρ such independent tests, all of these tests would err with probability at most $(\frac{1}{3})^\rho \leq \frac{1}{3kw^2}$. This would cause A_1 to erroneously say "Yes" due to this triplet. As there are at most kw^2 possible triplets, A_1 errs with probability at most 1/3. ∎

We now formally end the proof of Lemma 3.4 by presenting the following proposition and the testing algorithm it implies.

PROPOSITION 2. *Let B be a non δ-decomposable BP of width w and length n. Let $m, \{l_0, \ldots, l_p\}$, and $\mathcal{S}$ be as defined above (right after the statement of Lemma 3.4). Let $y \in \{0,1\}^n$; then, for any start and accept nodes $(s,t) \in L_0 \times L_n$, the following hold.*

1. *If $dist(y, B[s:t]) = 0$, then there exists a tuple $(v_0, \ldots, v_p) \in \mathcal{S}$ such that s can reach v_0, v_p can reach t, and $dist(y, B[v_i : v_{i+1}]) = 0$ for $i = 0, \ldots, p-1$.*
2. *Let $dist(y, B[s:t]) \geq \epsilon n$; then, for each $(v_0, \ldots, v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $\Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}]) \geq \epsilon n - l_0 - (n - l_p) \geq 0.9\epsilon n$.*

Proof. If $dist(y, B[s:t]) = 0$, then the path that y takes in B defines the tuple $(v_0, \ldots, v_p) \in \mathcal{S}$ which contains the nodes in which this path intersects L_{l_i}, $i = 0, \ldots, p$, along the way from s to t. This tuple asserts the first item of the proposition.

If $dist(y, B[s:t]) \geq \epsilon n$, then, for any $(v_0, \ldots, v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $dist(y, B[v_0 : v_p]) \geq \epsilon n - l_0 - (n - l_p) \geq 0.9\epsilon n$. However, $dist(y, B[v_0 : v_p]) = \Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}])$. ∎

Proposition 2 defines a way to combine answers to tests on BPs of the form $B[v_i : v_{i+1}]$ into an ϵ-test of B. Intuitively, on an input $x \in \{0,1\}^n$, we just need to check for all tuples $(v_0, \ldots, v_p) \in \mathcal{S}$, and check whether there exists one for which $dist(x, B[v_i : v_{i+1}]) = 0$ for $i = 0, \ldots, p-1$.

Formally, let $x \in \{0,1\}^n$ be the input. The following is an ϵ-test of B for any

start and accept nodes:

Algorithm $A_2(\epsilon, w)$. (B is a non δ-decomposable BP of width w.)

Let m and $\mathcal{S}$ be as above, and let $\nu = 1 + \frac{\log(pw^2)}{\log 3} = O(\log \frac{w}{\delta})$.

1. For each $(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, \ldots, p-1$, call $A_1(0.9 \cdot \epsilon, w, B[u:v])$ (namely, with $\alpha = 0.1$) independently, for ν times. If for (u,v), all of these tests answer "Yes," then define $T(u,v) = 1$. Otherwise, if there is a test out of the ν tests that answers "No" for (u,v), then set $T(u,v) = 0$.
2. Define the following directed graph $G = (V,E)$: $V = L_0 \cup L_n \cup (\cup_{i=0}^{p} L_{l_i})$, and

$$E = \{(s,u) \in L_0 \times L_{l_0} | s \text{ can reach } u \text{ in } B\}$$

$$\cup \{(v,t) \in L_{l_p} \times L_n | v \text{ can reach } t \text{ in } B\}$$

$$\cup \{(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, \ldots, p-1 | \text{ such that } T(u,v) = 1\}.$$

3. Answer "Yes" for $(s,t) \in L_0 \times L_n$ if and only if s can reach t in G.

CLAIM 3.9. *For any $(s,t) \in L_0 \times L_n$ and for every input x, the following hold.*

1. *If $dist(x, B[s:t]) = 0$, then Algorithm A_2 answers "Yes" on (s,t) with probability* 1.
2. *If $dist(x, B[s:t]) > \epsilon n$, then Algorithm A_2 answers "No" on (s,t) with probability at least* 2/3.

Proof. Assume that $dist(x, B[s:t]) = 0$ for an input $x \in \{0,1\}^n$ and $(s,t) \in L_0 \times L_n$. Then, by Proposition 2, there exists a tuple $(v_0, \ldots, v_p) \in \mathcal{S}$ such that s can reach v_0, v_p can reach t, and $dist(y, B[v_i : v_{i+1}]) = 0$ for $i = 0, \ldots, p-1$. By Claim 3.8, Algorithm A_1 answers "Yes" on each of the calls $A_1(0.9 \cdot \epsilon, w, B[v_i : v_{i+1}])$ with probability 1. Hence the path $(s, v_0, \ldots, v_p, t)$ is a valid path in G with probability 1, causing A_2 to answer "Yes" with the same probability.

For the second part, assume that $dist(x, B[s:t]) > \epsilon n$. Then, by Proposition 2, for each $(v_0, \ldots, v_p) \in \mathcal{S}$ such that s can reach v_0 and v_p can reach t, $\Sigma_{i=0}^{i=p-1} dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon n$. However, then, for each such $(v_0, \ldots, v_p) \in \mathcal{S}$, for some $i \leq p-1$, $dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon \frac{n}{p} > 0.9\epsilon m$. Let E' be the set that contains for each $(v_0, \ldots, v_p) \in \mathcal{S}$ a corresponding (v_i, v_{i+1}) for which $dist(x, B[v_i : v_{i+1}]) \geq 0.9\epsilon m$. Note that E' contains an (s,t)-cut in G. Namely, s cannot reach t in $G - E'$.

For each member $(v_i, v_{i+1}) \in E'$, Claim 3.8 asserts that Algorithm A_1 answers "No" on the call $A_1(0.9 \cdot \epsilon, w, B[v_i : v_{i+1}])$ with probability 2/3. Hence it answers erroneously "Yes" on all ν calls for a pair $(u,v) \in E'$ with probability at most $(\frac{1}{3})^\nu \leq \frac{1}{3pw^2}$. Namely, $T(u,v)$ is set erroneously to "1" in step 1 of the algorithm with probability at most $\frac{1}{3pw^2}$. However, there are at most pw^2 possible pairs in E'. This implies that with probability at most 1/3 there exists a pair $(u,v) \in E'$ for which $T(u,v) = 1$. In particular, it follows that s cannot reach t in G with probability at least 2/3. □

CLAIM 3.10. *For any $(s,t) \in L_0 \times L_n$ and for every input x, Algorithm A_2 makes $O(\frac{w^2}{\delta^2} \log \frac{w}{\delta})$ calls to A_1 with distance parameter 0.9ϵ.*

Proof. There are $O(pw^2) = O(\frac{w^2}{\delta^2})$ possible pairs $(u,v) \in L_{l_i} \times L_{l_{i+1}}, i = 0, \ldots, p-1$. For each pair (u,v), there are $\nu = O(\log \frac{w}{\delta})$ calls for A_1. □

COROLLARY 3.11. *Algorithm A_2 provides a 1-sided error ϵ-test for $B[s:t]$ for every start and accept nodes $(s,t) \in L_0(B) \times L_n(B)$, making at most $O(\frac{w^4}{\delta^3}(\log \frac{w^2}{\delta})^2) \cdot \tilde{q}(0.8\epsilon, w-1)$ queries.*

Proof. Claim 3.9 asserts the correction of A_2 as a 1-sided error ϵ-test for $B[s:t]$ for each $(s,t) \in L_0(B) \times L_n(B)$. Observe that the calls for A_1 do not depend on the

choice of s and t. Hence, with the same number of queries as described above for a given choice of s,t, A_2 provides an ϵ-test for every choice of s and t; for each s and t, the outcome has at least probability $2/3$ of being correct.

According to Claim 3.8, each call for A_1 results in possibly $O(\frac{w^2}{\delta}\cdot\log\frac{w^2}{\delta})$ calls to a general 0.8ϵ-test of $(w-1)$-width BPs. Claim 3.10 asserts that there are $O(\frac{w^2}{\delta^2}\log\frac{w}{\delta})$ calls to A_1; hence the claim follows. □

This ends the proof of Lemma 3.4. □

3.3. The general case. In order to test general w-width BPs, it remains to be shown how to reduce testing of decomposable BPs to that of nondecomposable ones. We need the following proposition.

PROPOSITION 3. *For a BP, B, and $t > r$, let t_1, t_2 be r-full vertices in L_t, and let $u \in L_l$ with $l \le t - r$. Then, for every $y \in \{0,1\}^n$,*

$$|dist(y, B[u : t_2]) - dist(y, B[u : t_1])| \le r.$$

Proof. The closest y' to y that traverses B from u to t_1 must intersect $A_r(t_2)$. Hence, by changing only the r last bits of y', we get a y'' that traverses B from u to t_2. □

DEFINITION 3.12. *Let $y \in \{0,1\}^n$ and $0 \le a < b \le n$; we define*

$$dist(y, B_{a:b}) = \min\{dist(y, B[u : v])| \ \ u \in L_a, v \in L_b\}.$$

CLAIM 3.13. *Let $B[s : t]$ be a BP of length ν with start vertex s (the only vertex at level L_0) and accept vertex t (the only vertex at level L_ν). Assume that there are a sequence of numbers $l_0 = 1, \ldots, l_h = \nu$ and a sequence of numbers $r_1, \ldots, r_h$, such that level L_{l_i} is r_i-full for each $i = 1, \ldots, h$. Then, for every $y \in \{0,1\}^\nu$, $\Sigma_i dist(y, B_{l_i:l_{i+1}}) \ge dist(y, B[s : t]) - \Sigma_i r_i$.*

Proof. Let y be such that $\Sigma_1^h dist(y, B_{l_{i-1}:l_i}) = d$. We will show that $dist(y, B[s : t]) \le d + \Sigma_i r_i$, which implies the claim.

Indeed, let $w_i = y[l_{i-1} + 1, : l_i]$, $i = 1, \ldots, h$, be the substring of y that corresponds to the variables of $B_{l_{i-1}:l_i}$. Let y'_i, $i = 1, \ldots, h$, be such that $dist(w_i, y'_i) = d_i$, $dist(y'_i, B_{l_{i-1}:l_i}) = 0$, and so that $\Sigma_1^h d_i = d$. Then for each y'_i, $i = 1, \ldots, h$, let $(u_i, v_i) \in L_{l_{i-1}} \times L_{l_i}$ be such that $dist(y'_i, B_{l_{i-1}:l_i}) = dist(y'_i, B[u_i : v_i]) = 0$. Namely, u_i, v_i are the start and end nodes through which y'_i travels through $B_{l_{i-1}:l_i}$. Then, by Proposition 3, for every $i = 1, \ldots, h$, there is a string z_i such that $dist(y'_i, z_i) \le r_i$ and $dist(z_i, B[u_i : u_{i+1}]) = 0$. However, then, the string $z = z_1 \cdot \ldots \cdot z_h$, which is the concatenation of $z_i, i = 1, \ldots, h$, "travels" in B through all the $u_i, i = 0, \ldots, h$. In particular, $dist(z, B[s : t]) = 0$ (as s and t are the only nodes in levels L_0, L_ν, respectively).

On the other hand, $dist(y, B[s : t]) \le dist(y, z) \le \Sigma_i dist(w_i, z_i) \le \Sigma_i (dist(w_i, y'_i) + dist(y'_i, z')) \le \Sigma_i (d_i + r_i) = d + \Sigma_i r_i$. □

We are ready now to prove Theorem 1.

Proof. Let B be a BP of width w and length n with start and accept nodes $s \in L_0$ and $t \in L_n$, respectively. Let $a_0 = 0$, and let $a_1 = l_1$ be the smallest integer such that level L_{a_1} is r_1-full for $r_1 \le \frac{\epsilon l_1}{20}$. Let l_2 be the smallest integer for which level L_{a_2}, $a_2 = a_1 + l_2$ is r_2-full for $r_2 \le \frac{\epsilon l_2}{20}$, etc. This defines a sequence of numbers $\mathcal{L} = (a_0, a_1, \ldots)$ of which the last may or may not be n. If the last number in $\mathcal{L}$ is not n, then we add n as the last member resulting in a sequence $\mathcal{L}'$; otherwise, we set $\mathcal{L}' = \mathcal{L}$. Assume that $\mathcal{L}' = (a_0 = 0, a_1, \ldots, a_h = n)$. This defines a sequence

of h BPs (with start and accept nodes that are not necessarily defined), $B_1, \ldots, B_h$, $B_i = B_{a_{i-1}:a_i}$ of length $l_i = a_i - a_{i-1}$.

Note that, by our choice, for every $i = 1, \ldots, h$, either $l_i = O(1)$ or B_i is not ϵ_1-decomposable for $\epsilon_1 = 0.5\epsilon$. Moreover, for every $i = 1, \ldots, h-1$, the last level of B_i is r_i-full, and for B_h (with L_n as last level) L_n is either r_i-full if $n \in \mathcal{L}$ or is 1-full if n was added to result in $\mathcal{L}'$ (as t is always 1-full in $B[s:t]$).

An ϵ-test of B is done as follows: For $4/\epsilon$ times, independently an $i \in \{1, \ldots, h\}$ is chosen at random with probability proportional to the length l_i. Let I be the multiset that contains the $4/\epsilon$ chosen i's, possibly with multiplicity. Let T_i be a Boolean flag associated with each $i \in I$. For each $i \in I$, an ϵ_1-test is performed on $B_{a_{i-1}:a_i}$ for every choice of start and accept nodes $(u, v) \in L_{a_{i-1}} \times L_{a_i}$. If for some pair $(u, v) \in L_{a_{i-1}} \times L_{a_i}$, the answer to (u, v) in this test is "Yes" then we mark T_i as "1." Otherwise, if, for all such pairs (u, v), the answer is "No," we mark it as "0."

Finally, if there exists a chosen $i \in I$ for which $T_i = 0$, then the answer to the ϵ-test for B is "No." Otherwise, if for all chosen i's $T_i = 1$, then the answer to the ϵ-test on B is "Yes."

Let us first analyze the query complexity of the above test: As was remarked before, each B_i is either of $O(1)$ length or non ϵ_1-decomposable. Hence, for each chosen i, an ϵ_1-test for each start and accept node $(u, v) \in L_{a_{i-1}} \times L_{a_i}$ can either be done in $O(1)$ queries (in the former case) or it can be done by calling Algorithm A_2 for nondecomposable BPs. Note that, in the latter case, Corollary 3.11 asserts that one call to A_2 provides a test for each start and accept node.

Since there are at most $4/\epsilon$ calls for A_2 (with $\delta = \epsilon_1$), the total complexity is

$$\tilde{q}(\epsilon, w) \leq \frac{4}{\epsilon} \cdot O\left(\frac{w^4}{\epsilon_1^3}\left(\log \frac{w^2}{\epsilon_1}\right)^2\right) \cdot \tilde{q}(0.8\epsilon_1, w-1) = O\left(\frac{w^4}{\epsilon^4}\left(\log \frac{w^2}{\epsilon}\right)^2\right) \cdot \tilde{q}(0.4\epsilon, w-1),$$

which implies that $\tilde{q}(\epsilon, w) = (\frac{2^w}{\epsilon})^{O(w)}$.

Let us check the error probability of this algorithm. If, for an input $x \in \{0,1\}^n$, $dist(x, B[s:t]) = 0$, then, for every $i \in I$ that is chosen in the algorithm above, $dist(x, B[u:v]) = 0$ for some $(u, v) \in L_{a_{i-1}} \times L_{a_i}$. Hence the answer will be "Yes" with probability 1.

For an input x such that $dist(x, B[s:t]) \geq \epsilon n$, by Claim 3.13, $\Sigma_i dist(x, B_{a_i:a_{i+1}}) \geq \epsilon n - \Sigma_i r_i$. However, as $r_i \leq \frac{\epsilon \cdot l_i}{20}, i = 1, \ldots, h-1$, and $r_h \leq \max\{1, \frac{\epsilon \cdot l_h}{20}\}$, we conclude that $\Sigma r_i \leq \frac{\epsilon n}{20} + 1$, and hence $\Sigma dist(x, B_{a_i:a_{i+1}}) \geq \frac{94}{100}\epsilon n$ for large enough n. Thus, by sampling one $i \in \{1, \ldots, h\}$ as above, we get that $dist(x, B_{a_i:a_{i+1}}) \geq \frac{1}{2}\epsilon l_i = \epsilon_1 l_i$ with probability at least 0.44ϵ. To see this, let $D = \{i |\ dist(x, B_{a_{i-1}:a_i}) \geq \frac{1}{2}\epsilon l_i\}$, and let $d_i = dist(x, B_{a_{i-1}:a_i})$; then

$$\frac{94}{100}\epsilon n \ \leq\ \Sigma_{i \in D} d_i + \Sigma_{i \notin D} d_i \ \leq \Sigma_{i \in D} l_i + \frac{1}{2}\epsilon \Sigma_{i \notin D} l_i \ \leq \text{Prob}(i \in D) \cdot n + \frac{1}{2}\epsilon n,$$

which implies that $\text{Prob}(i \in D) \geq 0.44\epsilon$.

Assuming that $i \in D$, for every $u \in L_{a_{i-1}}, v \in L_{a_i}$, $dist(x, B[u:v]) \geq \epsilon_1 l_i$. Thus the success probability for a chosen i is at least $0.44\epsilon \cdot \frac{2}{3}$. Namely, $i \in D$, and the ϵ_1-test on B_i answers "No" as it should for at least one pair of start and accept nodes of B_i. Making $4/\epsilon$ independent, such tests will again reduce the error probability to below $\frac{1}{3}$. □

3.4. Time complexity. We end this section with a note on the total running time of the algorithm. Every fixed BP, B, defines a property $\mathcal{P}_B \subseteq \{0,1\}^n$. We have

presented in the section above an "algorithm scheme." Namely, it produces an ϵ-test for any given ϵ and w-width oblivious read-once BP, B. For the algorithm scheme, the input is ϵ and B, while, for the property tester, the input is $x \in \{0,1\}^n$. These two notions should not be confused. Thus, in analyzing the running time of the ϵ-test of $\mathcal{P}_B$ for a given BP, B, we may assume that we have at hand the decomposition of B into nondecomposable parts for all possible recursion levels. We also assume that we have $F(l)$ for every level l and for every possible subprogram that is considered in any of the recursion levels. We do not discuss how this data is represented or computed, which is out of the scope of this paper. We note, however, that, by computing all-pairs-connectivity, the data above can easily be obtained. Hence the above can be done in polynomial time (in the length of B and $1/\epsilon$).

For an input $x \in \{0,1\}^n$, the operations in a given recursion level involve sampling a decomposable subprogram, calling A_1 and A_2, and processing the return answers of Algorithms A_1 and A_2. Sampling one decomposable program takes $O(\log n)$ steps since there might be $O(n)$ nondecomposable B_i's in the top level. Once all calls to A_1 are done, computing the outcome of Algorithm A_2, for a w-width BP in the top recursion level, is done by forming the graph G and then checking whether s can reach t in G. Given the answers of the calls to A_1, preparing the graph G takes $O(pw^2) = O(\frac{w^2}{\epsilon^2})$ steps. Then, solving the connectivity problem on G takes $O(\frac{w^2}{\epsilon^2})$ steps. Putting this together yields the following recursion for the time $t(\epsilon, w, n)$, where ϵ is the distance parameter, w is the width, and n is the length of the BP:

$$t(\epsilon,w,n) = \frac{4}{\epsilon} \cdot \left[O(\log n) + O\left(\frac{w^2}{\epsilon^2}\right) + O\left(\frac{w^2}{\epsilon^2} \log \frac{w}{\epsilon}\right) \right.$$
$$\left. \cdot O\left(\frac{w^2}{\epsilon} \cdot \log \frac{w^2}{\epsilon}\right) \cdot t(0.4\epsilon, w-1, n) \right].$$

The $\frac{4}{\epsilon}$ term comes from the number of i's chosen in the top level general test. The $\log n$ comes from sampling one i. The $O(\frac{w^2}{\epsilon^2})$ term comes from deciding the connectivity in A_2, and the rest come from A_1 multiplied by the number of calls to it from A_2.

Solving the above yields $t(\epsilon, w, n) = (\frac{2^w}{\epsilon})^{O(w)} \cdot \log n$.

4. Examples of interesting functions and open problems. We present here some examples of functions that have narrow width, read-once BPs and are "efficiently" testable. (Sometimes a direct efficient testing algorithm is obvious.) The first nontrivial such family is of all regular languages with a direct testing algorithm by [2]. We remark here that, for this case, our algorithm is conceptually different than that of [2]. The dependence of the query complexity on w in this case is similar to what would result from [2]. The dependence on ϵ is worse.

Other very simple families are k-term-DNF and k-clauses-CNF, each having 2^k-width oblivious read-once BP. A function $g : \{0,1\}^n \longrightarrow \{0,1\}$ is k-term-DNF if it has a DNF representation (a disjunction of terms where each is a conjunction of literals) with at most k terms. Analogously, a k-clause CNF is defined. Two remarks are due here: For both k-term-DNF and k-clause-CNF, ϵ-tests are known (folklore): k-term DNF is $(\epsilon, O(\frac{k \log k}{\epsilon}))$-testable by testing for each term separately. k-term CNF can be tested by 0 queries for any $\epsilon > \frac{k}{n}$ as such function is either constant or every input has distance at most k to a satisfiable one.

It is also interesting to note that 1-term-DNF includes examples of functions that are, say, in uniform $SPACE(\log\log n)$ but not regular and hence do not belong to $SPACE(o(\log\log n))$. One such interesting example is the following example of Papadimitriou [14]: Let b be a binary string without leading 0's. We denote by $n(b)$ the natural number whose binary representation is b. Let $L = \{b_1\$b_2\$\ldots\$b_k|\ n(b_i) = i\}$. Clearly $L \in SPACE(\log\log n)$. It is also not hard to see that L is not regular. However, as L contains at most one word of each length, it obviously has a BP of width $w = 1$. Note that, although we have here an alphabet of size 3, we may actually encode everything in binary by encoding each symbol with two bits.

In view of Theorem 1, one may ask what is the true dependence of ϵ-testing w-width read-once BPs on w and ϵ. This remains open at this point. Another more puzzling question is whether $SPACE(\log\log n)$ can be "efficiently" testable. (By this we mean with complexity, say, less than n^δ for any $\delta > 0$.) Currently we do not have any candidate for a counterexample to this.

Another issue is how far the current result may be generalized. One restriction that may be considered is being "read-once"—can this be replaced by, say, polynomial total size? To this, the answer is false: Barrington [4] has proved that every NC^1 function has a polynomial length oblivious leveled BP of width 5. However, in [2], examples of such functions that require $\theta(\sqrt{n})$ queries are presented. Hence, instead, one may ask whether constant width linear size BPs are testable. A negative answer is given in [9]: They show that there is a Boolean function $g : \{0,1\}^n \longrightarrow \{0,1\}$ that is computed by a read-twice constant width oblivious BP and that is not ϵ-testable for some fixed $\epsilon > 0$ (a read-k-times BP is a BP where each variable appears in at most k levels).

REFERENCES

[1] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 656–666.

[2] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, *Regular languages are testable with a constant number of queries*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 645–655.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[4] D. M. Barrington, *Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1*, J. Comput. System Sci., 38 (1989), pp. 150–164.

[5] P. Beame, M. Saks, and J. S. Thathachar, *Time-space tradeoffs for branching programs*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 254–263.

[6] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1994), pp. 549–595.

[7] R. B. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, Elsevier, Amsterdam, 1990, pp. 757–804.

[8] A. Cobham, *The recognition problem for the set of perfect squares*, in Proceedings of the 7th IEEE Symposium on Switching and Automata Theory, Berkeley, CA, 1966, pp. 78–87.

[9] E. Fischer and I. Newman, *Functions that have read-twice, constant width, branching programs are not necessarily testable*, to appear in the International Workshop on the Aspects of Complexity and Its Applications, University of Rome La Sapienza, Rome, Italy, 2002.

[10] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 32–42.

[11] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connections to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[12] J. Hartmanis, P. L. Lewis II, and R. E. Stearns, *Hierarchies of memory-limited computations*, in Proceedings of the 6th IEEE Symposium on Switching Circuits and Logic Design, IEEE Computer Society, Los Alamitos, CA, 1965, pp. 179–190.

[13] S. Jukna, A. A. Razborov, P. Savick, and I. Wegner, *On P versus $NP \cap co-NP$ for decision trees and read-once branching programs*, in Mathematical Foundations of Computer Science, Springer-Verlag, Berlin, 1997, pp. 319–326.

[14] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994, exercises 2.8.11 and 1.8.12, pp. 54–55.

[15] M. Parnas, D. Ron, and R. Rubinfeld, *Testing parenthesis languages*, in Proceedings of APPROX/RANDOM 2001, Lecture Notes in Comput. Sci. 2129, Springer-Verlag, New York, 2001, pp. 261–272.

[16] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1571–1595

SELF-STABILIZING SYMMETRY BREAKING IN CONSTANT SPACE*

ALAIN MAYER†, RAFAIL OSTROVSKY‡, YORAM OFEK§, AND MOTI YUNG¶

Abstract. We investigate the problem of self-stabilizing round-robin token management on a bidirectional ring of identical processors. Each processor is an asynchronous probabilistic finite state (i.e., constant space) machine which sends and receives constant-size messages and whose state transition is triggered by the receipt of a message. We also show that this problem is equivalent to symmetry breaking (i.e., leader election).

We justify and suggest a two-layer (hardware and software) solution to the token management problem: The subproblem of reducing an arbitrary but nonzero number of tokens (in an otherwise arbitrary initial system state) to exactly one token (and a legal system state) is solved in hardware and takes only small polynomial time. The detection of a complete lack of tokens (communication deadlock) is done by a software clock. In high-speed networks the hardware layer can be implemented using fast universal switches (i.e., finite state machines) *independent* of the size of the network. We note that randomization is essential, since Dijkstra showed that for arbitrary rings the subproblem does not have a deterministic solution (regardless of the computational power of the identical processors). The use of the software layer (deadlock detection) in our solution is minimized.

Key words. self-stabilization, self-stabilizing protocols, distributed algorithms, token ring protocols, media access protocols

AMS subject classifications. 68W15, 68M14, 68W40, 68M12

PII. S0097539798285997

1. Introduction.

We consider a network whose topology is an arbitrary-size ring of uniform (identical) processors whose size is independent of the ring size. We use asynchronous message-passing. Each processor can change its state and send messages only upon receiving a message from one of its two adjacent neighbors. This is usually called a message-driven model (discussed further in section 2.2).

Our first goal is the design of a *fault-tolerant token management protocol*. A token management (TMA) scheme guarantees that the ring network always contains a predefined number of tokens which circulate in a round-robin fashion (e.g., a token in a token ring, slots in a slotted ring). In addition, any node waiting to see a token should be only minimally delayed. An important concern is to make the TMA scheme fault-tolerant. We aim at making the system able to recover from an arbitrary transient fault which puts the system in any (even maliciously chosen) state. In fact, Dijkstra considered such a fault-tolerant token management scheme in the shared memory model in his seminal paper, which introduced the notion of *self-stabilization*

*Received by the editors June 10, 1998; accepted for publication (in revised form) June 21, 2000; published electronically August 5, 2002. A preliminary version of this work appeared in the Proceedings of the 24th ACM Symposium on the Theory of Computing, Victoria, BC, Canada, 1992.
http://www.siam.org/journals/sicomp/31-5/28599.html

†CenterRun, Inc., Suite 101, 900 Island Drive, Redwood City, CA 94065 (alain_mayer@hotmail.com). The research of this author was partially supported by ONR grant N00014-91-J-1613. Part of this work was performed while the author was at IBM, T.J. Watson Research Center, Yorktown Heights, NY 10598.

‡Telcordia Technologies, Inc., 445 South Street, Morristown, NJ 07960 (rafail@research.telcordia.com). The research of this author was supported by an IBM Graduate Fellowship. Part of this work was done at IBM, T.J. Watson Research Center, Yorktown Heights, NY 10598.

§Synchrodyne Networks Inc., T.J. Watson Center, Yorktown Heights, NY 10598 (ofek@synchrodyne.com).

¶CertCo Inc. and Columbia University, New York, NY 10027 (moti@cs.columbia.edu).

[Dij74]. Informally, the goal of self-stabilization is for the system to reach "normal" operation, starting from an arbitrary initial state (see section 2.1 for more details). Of course, while making the scheme fault-tolerant, we should *not* sacrifice efficiency in both the speed and space requirements. That is, most of the solution should be realized in hardware and should rely as little as possible on a software clock or any other software mechanisms. (See Figure 1 and sections 1.1 and 2.2 for more details.)

Dijkstra's original solution assumed the presence of a leader. However, choosing a leader among identical processors is one of the most basic problems in distributed computing; a leader can coordinate any distributed task. How difficult is it to elect a leader in a self-stabilizing fashion in our model? Our second contribution is a pair of reductions (see section 2.4 for a definition) from leader election to self-stabilizing bidirectional round-robin TMA and vice versa. We show that token management and leader election are equally hard.

1.1. Design of hardware-based protocols. One motivation for considering the above model comes from existing architectures of high-speed communication networks. The bottleneck in these architectures is typically the speed of the processors with which they can process incoming messages. Hence, it is important that essential protocols, such as media access control, can be implemented in simple hardware. This enables a (constant-size) control message to go through many high-speed hardware switches (finite-state machines) and through many communication links during a single software clock-tick.

In many systems (e.g., token rings [BCK+83], MetaNet [OY90]; see section 8), fair access to the network is achieved by passing a single *token* (i.e., a constant-size control signal) in a round-robin fashion on a physical or logical ring. When a network node needs permission for a certain action, it waits for a token, holds it for a single unit of time, and then passes it on to other processors to ensure fairness.

The above approach is efficient only when being designed in hardware. Furthermore, the switching hardware should decide even before the token arrives if there is a need to hold it. This depends only on the local state and the requirements of the network node. For example, let us consider the case when only a single processor on a ring competes for a resource. The fact that the processor must give up a token to ensure fairness (i.e., to check if there is some other processor waiting) *does not* incur a slowdown proportional to the size of the ring. When the processor gives up a token it does not wait for the number of software clock-ticks proportional to the size of the ring: the constant-bit control message makes a full round without substantial delay by going through high-speed switches at each node and does not wait for software clock-ticks at any of them. Moreover, the constant-bit control signal essentially does not decrease the bandwidth of the communication links, making the above scheme useful in practice. The interplay between hardware and software is thus as follows: in normal operation the high-speed switch can work while the software monitors it. This implies the two-layer solution depicted in Figure 1. An efficient solution lets the software intervene only in rare "catastrophic" events (e.g., by employing long time intervals for time-outs).

1.2. Related work. Dijkstra [Dij73] showed that round-robin TMA, even when starting with two or more tokens, is impossible on a ring of unknown size of uniform and deterministic processors. He gave a solution in the presence of a leader. However, the problem of "whether a system of *probabilistic* automata can have a round-robin TMA scheme" remained open.

Israeli and Jalfon showed that if each token executes a symmetric random walk,

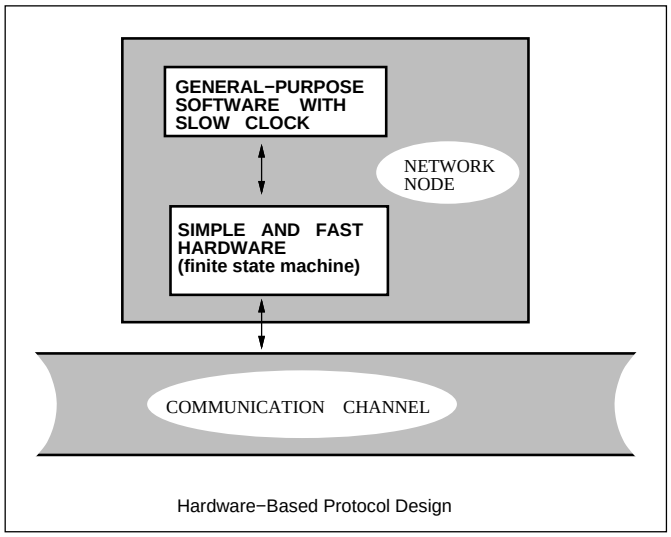


FIG. 1. *Node model.*

such that when two tokens meet one gets eliminated, then eventually only one token will remain [IJ90a]. Notice, however, that their solution gives up the round-robin property. This relaxation implies that its use is mainly for mutual exclusion (to solve contention). It cannot be used for regulation and fair coordination of the network communication medium, which is the usual task of token systems. Indeed, Lamport and Lynch [LL90] mention that "there is one important property that is harder to achieve in coordination problems than in contention—namely self-stabilization." It seems possible to extend the random walk solution to achieve a basic round-robin property: all tokens are sent only in one direction and at each step, each token flips a coin and either stays for a single clock-tick or advances; when two tokens meet, one gets eliminated. However, this solution increases the round-trip delay of a token and thus drastically decreases the throughput of the network, given that the token is used for access control.

Other solutions to self-stabilizing TMA either assume that there is a distinguished node (leader) or make further assumptions about the network size (such as the number of nodes being prime or odd). Leader election is a difficult problem (see section 6) and hence the assumption of having a distinguished node available at all times is very strong. In practice, the size of a token ring being constrained to be prime or odd is rather awkward. Brown, Gouda, and Wu [BGW89] and Afek and Brown [AB89] proposed solutions using a leader. Note that the first solution actually works for more general topologies than rings, while the latter is the only previous solution designed for the message-passing model. Burns and Pachl [BP88] and Herman [Her90] present solutions for special size rings. It is worthwhile mentioning that the first presents a deterministic and uniform solution and the latter is probabilistic and uniform.

There are many papers concerning self-stabilizing leader election (mostly on general topology networks), e.g., [AKY90, APV91, DIM91, KP90, IL94, AO94, ILS95]. All of them either are not message-driven and use the paradigm of local checking (introduced in [AKY90] and [APV91]), perform repeated snapshots [KP90], or use additional restrictions on the size of the ring [ILS95]. In section 2.2 we explain why these paradigms are not suitable for our purpose. Furthermore, most of the previ-

ous work uses logarithmic size memory (to use identification-based techniques). Following the preliminary version of this paper with its two-layer (hardware-software) and small memory approach, subsequent work paid attention to small space restrictions [IL94, AO94, PY94, ILS95]. In this context the work of [MOY96] further restricts this model to *unidirectional* networks in order to cover such architectures as FDDI (see section 8).

Organization of the rest of the paper. In section 2 we present our model and definitions and an overview of our results. In section 3 we present a straightforward modification of a known simple algorithm for the attrition problem on rings to make it "solve" TMA. We show that its fault-tolerance does not satisfy self-stabilization. In section 4 we present our TMA algorithm, and in section 5 we demonstrate correctness and give the time analysis. In sections 6 and 7 we present the reductions between the TMA problem and the leader election problem. In section 8 we present applications.

2. Model and results.

2.1. Self-stabilization. A system S consists of processors and their interconnection-links. The global state of S is the cross-product of the local states of its components. Let P denote a desired predicate over the global state, e.g., there exists a single token in the system.

An algorithm A in system S is self-stabilizing with respect to a predicate P if it satisfies the following two properties (following Schneider's survey [Schn93]):

- Closure: P is closed under the execution of A on S. That is, once P is established in S, A does not falsify it.
- Convergence: Starting from an arbitrary global state, A guarantees to reach a global state in S which st satisfies P.

The above definition covers at least the following: arbitrary inconsistencies when initializing the system S, transmission errors, process failure and its recovery, and memory loss and contamination. Note that this allows us from now on to concentrate on correct executions starting in an arbitrary initial state, rather than trying to cover the above problems case by case.

Arora and Gouda [AG92] introduced the notion of *stabilization* with respect to predicates P and Q as an algorithm which satisfies closure and convergence with respect to P, given that the initial state satisfies Q.

Some of our solutions are randomized. Correctness in this case generalizes the above definition of convergence to occur with probability 1.

2.2. Model of a high-speed ring network. In the following we describe our system S. We restrict ourselves to networks whose topology is a ring. We let n denote the number of nodes in the ring. Each processor (node) in the ring is a probabilistic, finite state machine (PFSM). The PFSM communicates with its two neighbors via asynchronous message-passing. That is, every message sent on a link is eventually received by the respective neighbor. The PFSM has a constant-size buffer for each of its two neighbors. Hence, the PFSM can receive only constant-size (control) messages. The PFSM performs a state transition and sends its own control messages only upon the receipt of such a message. Thus, the processing at a node is done in a message-driven, asynchronous fashion. This is in contrast to other self-stabilizing models, such as local checking, where each node is continuously requesting information from its immediate neighbors. Message-driven computing is better suited for our hardware-based model: There is no need for a notion of time and hence for a clock, which adds to the complexity of the required hardware. It also seems

problematic to find the right trade-off for the time-out value: A small value adds an unacceptable message overhead on each link of the network; a large value delays the stabilization-time. We can assume that the ring is oriented by using a self-stabilizing automata-based orientation protocol as suggested in [IJ90b]. A link between two nodes is modeled as two independent FIFO-channels. We allow neighboring nodes to detect *message collision* as a primitive event. That is, two control messages traveling in opposite directions are said to *collide* if they physically cross each other on the link.[1]

2.3. Self-stabilizing TMA and maintenance. Let predicate P be defined as follows: The set of global states in which there is exactly one token (in a buffer or on a link) in each direction and the state of each PFSM correctly reflects the last token passed through that node. We call a self-stabilizing algorithm for predicate P and system S a TMA scheme.

Let Q be the predicate over the global states in which there is at least one token. Node and token states are still arbitrary. We define *token maintenance* (TM) as the stabilization with respect to P and Q. As mentioned in section 1.2, it is exactly for the problem of TM that Dijkstra has shown that there is no general deterministic solution, namely either lack of stabilization or a deadlock must occur.

We define a TM algorithm to be *fair* if every node always sends a received token to the opposite neighbor within one time unit. This property is important for implementing fair media access control.

We say a TM is *nondeadlocking* if it never reaches a state with no tokens. In our context this is a requirement that prevents systems from relying on the slower (software) layer when not necessary.

2.4. Reduction. Let A be a self-stabilizing algorithm with respect to a predicate P. Let σ be the space requirement of A on each node and let τ be A's stabilization time.

DEFINITION 2.1. *We define Algorithm A' to be a* polynomial time space-preserving self-stabilizing reduction *to A with respect to a predicate P' and a system S if A' is executed concurrently with A in S, it stabilizes to P', its space requirement in addition to A's space on each node is $O(\sigma)$, its messages are of length $O(\sigma)$, and its stabilization time is $O(\tau + p(n))$, where p denotes an arbitrary polynomial.*

Note that the above definition allows Algorithm A' to introduce additional randomness.

2.5. Our results. Given the definition of system S and predicates P and Q in section 2.3, our first contribution is a fair, nondeadlocking stabilizing protocol for S with respect to P and Q (TM) with stabilization time $O(n \log n)$. An attractive feature of such a TM algorithm is that we can choose an *arbitrary* number of PFSMs, plug them together in a ring, and consequently obtain a correct token maintenance scheme. In a second step, our TM algorithm can be augmented to token management via the following two-layer approach: A node times out after it does not see a token for "long enough" and inserts a new token into the system S in an uncoordinated fashion, i.e., without consulting any other node. This time-out has to be dependent

[1] Our links are unit-capacity and so a handshake protocol for the control messages to detect a collision can be implemented. This is similar to [APV91] which showed this to be a necessary condition to avoid an initial state with infinitely flooded channels. Alternatively, if we assume an upper bound on the communication-delay for control messages over a single link, we can detect collision by using a local time-out as is done in practice.

on the ring size. Hence, the corresponding clock needs to be implemented in software. This software time-out needs only be triggered when all tokens in the system are lost within one round-trip delay and hence its use is minimized. Furthermore, if a nondeadlocking TM is used, each node times out at most once during the stabilization period and never times out once S has stabilized. This is in contrast to the type of time-out needed for implementing local checking.

Our second contribution is a pair of polynomial time space-preserving self-stabilizing reductions from TMA to leader election and vice versa. Given the fundamental significance of leader election in distributed computing, this result demonstrates that token management is a basic problem. Furthermore, given the result of Angluin [Ang80] about the impossibility to elect a leader deterministically among identical processors, this result provides an alternative proof that randomization is required for TM in our model.

3. A simple (and failed) approach for TM. Here we demonstrate the difference between self-stabilizing and non-self-stabilizing solutions. [AAGHK89] and [AAHK91] present algorithms for the attrition problem on rings. A procedure solves the attrition problem if, when initiated by every candidate, it eventually takes all but exactly one of these candidates into a permanent state of noncontention. In order to appreciate the inherent difficulty of the self-stabilizing TM problem, we modify an attrition algorithm in order to make it "solve" the TM problem and show that such an approach is not powerful enough.

Consider the code below. Assume that tokens are generated by some time-out mechanism, as advocated in section 2.5. When a node times out, it becomes active and executes the loop below. Inactive nodes simply forward tokens unchanged. The state space of a node consists of the following two local variables: $active \in \{true, false\}$ and $flip \in \{heads, tails\}$. The state space of a token is the variable $value \in \{heads, tails\}$. The complete state space of a node or a link includes the tokens which currently are located at that device.

```
while active do
        flip := random independent uniform coin toss
        value(token) := flip
        send(token)
        receive(next-token)
        if flip = heads and value(next-token) = tails, then active := false
end while
```

Arbitrary initial states in this state space include the following:

1. All nodes are inactive and there is more than one token.
2. There is one token (say, with value = heads) and one active node (say, with flip = tails).
3. There are more tokens than active nodes.
4. There are more active nodes than tokens.

For initial states 1 and 3 the above protocol remains indefinitely in states in which there is more than one token; the protocol does not guarantee convergence. For initial states 2 and 4 it is possible that a state is reached in which there are no tokens, hence the protocol is deadlocking. In general, it is not hard to see that the number of active nodes equals the number of tokens is a necessary condition for nondeadlocking convergence to states in which there is exactly one token circulating. The above algorithm cannot deal with inconsistent initialization, which is an important aspect of self-stabilization (see section 2.1).

4. The TM algorithm *A*. In this section we present first high level ideas and then an exact formulation of the fair and nondeadlocking TM algorithm A on a bidirectional ring. The algorithm stabilizes to a state in which exactly one token circulates in each direction of the ring. Any client program is free to choose one of these tokens as its unique token with respect to the network-function associated with it (e.g., access control). We call tokens circulating in one direction *positive* tokens and the token circulating in the opposite direction *negative* tokens.

4.1. Basic ingredients. The following type of control messages are used:

- Tokens: A token t is a constant-size message which carries a value $V(t) \in \{-3, -2, -1, 1, 2, 3\}$. Tokens which go clockwise always have a positive value, and tokens which go counter-clockwise always have a negative value. When a token passes through a node, it leaves a "footprint" of its value on this node, overwriting any previous "footprint."
- Probes: A probe p is also a constant-size message, but it does not leave any "footprints." Probes also carry a value $V(p) \in \{-3, -2, -1, 1, 2, 3\}$.

Nodes store "footprints" of passing tokens. When we say that node i has a certain "value" $V(i)$, we mean the value of its current "footprint."

Having introduced the basic elements, we now present an overview of the algorithm: We note that line numbers used in this overview refer to lines in the code-like description of the algorithm, which can be found in section 4.3. The most basic operation of a node i is to receive a token t, copy the value (or footprint) of t into its own register ($V(i) := V(t)$), and forward the token (lines B10 and B11). In order to make the system self-stabilizing we need to add at least the following two mechanisms: *token generation* and *token elimination*. It should be noted at this point that both operations use local information only. As we will see later, this implies that these decisions are not always optimal with respect to the global state. In particular, it is possible that they cause the number of tokens in the system to increase temporarily. But overall, the system gets "more organized" with each step and thus convergence to a legal state is assured. In order to implement elimination, we introduce the notion of a *token collision*. Every time a positive and a negative token (t_1 and t_2) meet on the ring, they execute a collision operation. A collision between t_1 and t_2 consists of executing the following operations: if t_1 and t_2 have identical absolute values ($|V(t_1)| = |V(t_2)|$), then the tokens flip a fair coin to agree on a new, distinct value (lines A4, A5) while maintaining their sign (see Figure 2). In other words, they agree on new, up to the sign identical, footprints. In the other case, where their absolute values are different (e.g., $|V(t_1)| < |V(t_2)|$), the token with the smaller absolute value (e.g., t_1) is eliminated (line A2). The intuition behind the collision operation is the following: if two tokens have distinct values and these values originate from a previous collision, then there must be more than one pair of tokens in the system. If the (absolute) values are equal, we still cannot be sure that there is really just one pair, so we must continue to check, i.e., choose a new random value. Note that the above is the only condition on which a token is eliminated and thus the nondeadlocking property of the algorithm follows easily. As mentioned in section 2.2, a collision can be detected by the corresponding endpoints, which in reality are the entities executing the above operations.

Now let us focus on the generation operation: when a token t arrives at node i and sees a footprint identical to its own ($V(t) = V(i)$), it assumes that the footprint at node i was left behind by itself. If this assumption is correct, it implies that neither token t nor node i has seen another token since t left i at its last visit. In that case t

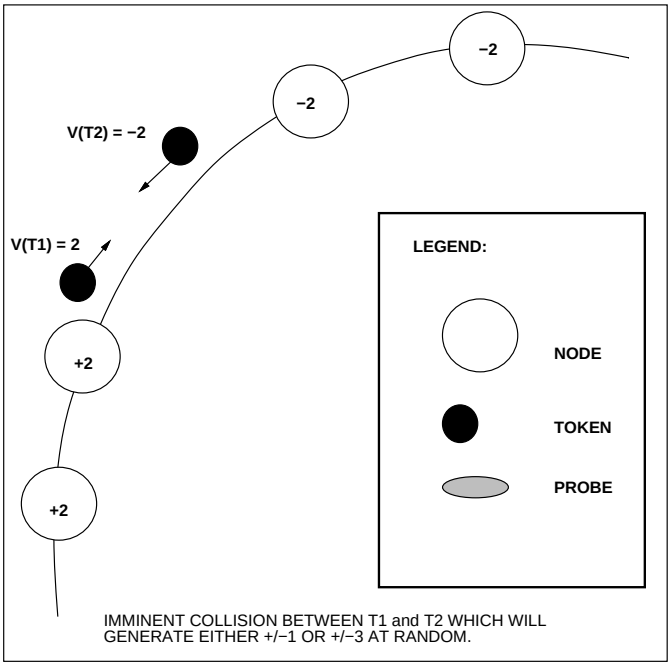


FIG. 2. *Token collision.*

can conclude that it is the only token left in the system and generates a new token of opposite sign (see Figure 3). The idea of token generation based on marking has been used by [Mis83] for unidirectional rings, though his method is different and actually uses $O(\log n)$ space.

So far we have described the collision (and thus the elimination) and the generation operation. Unfortunately, this is not enough to ensure convergence. To see why, consider a state in which all tokens travel in the same direction (we call such a state *skewed*). In such a state there are only positive (or only negative) tokens left; if each pair of neighboring tokens leaves distinct footprints behind, neither a collision nor a generation will ever take place.

To overcome this difficulty we introduce "exploration" by a second kind of message, called a *probe*. As we will show now, probes are used to explore the current state and, if appropriate, turn around a token from positive to negative (or vice versa). Probes are always generated by tokens ("fathers") in pairs ("siblings"). Siblings travel in opposite directions to each other. A probe traveling in the same direction as its father token is called a *forward probe*; analogously, a probe traveling in the opposite direction is called a *backward probe*. Now, every time a token suspects that the system is in a skewed state, it sends a forward and backward probe to "explore" the global situation. The condition to send probes is enabled when a token sees footprints (values) of a token of the same kind (direction) which are larger (in absolute values) than its own. So, for example, a positive token t will send out probes if, upon arriving at some node i, the condition $V(t) < V(i)$ holds (line B6). A probe p is basically carrying the value of its father ($V(p) = V(t)$) and an enabled/disabled bit (initially enabled).

After a probe p has been generated (and p is enabled), one of the following two events will occur: (i) p meets or collides with a token t which has a value of opposite

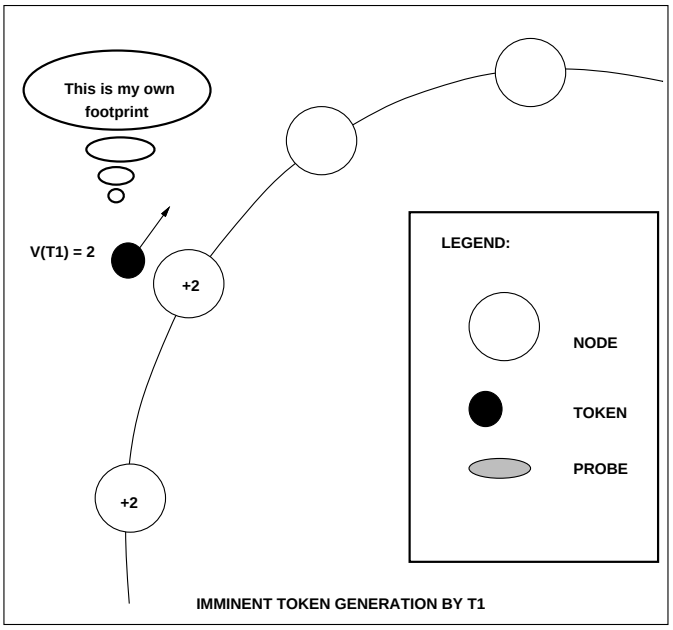


FIG. 3. *Token generation.*

sign ($V(p)V(t) < 0$) or (ii) p collides with a probe p' which has the same value as p and p thus assumes that p' is its sibling. In the first case, p has obtained evidence (i.e., the token t) that the current state is not skewed and thus p will be disabled (lines D1–D2, E1–E2). In the second case, p has not obtained any evidence that the current state is not skewed, so it will rely on the experience of p', since if p' is indeed p's sibling, these two probes have together covered the whole ring. If both are still enabled, then the probes conclude that the current state is indeed skewed, and thus the backward probe, whose next collision is with its father-token, will change the sign (and direction) of its father (line E9). Otherwise, they conclude (correctly) that the current state still contains tokens of opposite sign, and thus they eliminate each other (line F7).

4.2. How to piece the basic operations together. It is easy to reconstruct how a lone token will generate a token of opposite sign. So we provide now an intuition on how the operations described above work together to reduce the number of tokens, if there are more tokens in the system than just one positive and one negative. Consider the following simple observation: Let $r, v \in \{1, 2, 3\}$. Let r be a random value and v be either an arbitrary ("initial") value or another independently obtained random value. Thus $\Pr(r \neq v) > 0$ and in an infinite sequence of comparisons of two such values we get a mismatch with probability 1.

Now, if in a collision at least one of the tokens carries a value obtained in a previous collision (and is thus random), then exactly the comparison operation described above will be executed and a token will be eliminated if there is a mismatch. Thus our strategy should be to ensure that this kind of collision is repeatedly going to take place as long as the system has not stabilized to one pair of tokens. In order to show this, we have to make sure that probes which originate from an "initial" state or have been produced erroneously are not an obstacle by causing tokens to repeatedly

turn and thus preventing these useful collisions. This will be done by again using our simple observation with respect to comparing values when (erroneous) backward probes and tokens collide. Thus we will be able to conclude that despite the presence of arbitrary probes, the system stabilizes to exactly one pair of tokens.

All we have to add to make the algorithm complete are conditions on which a disabled probe who does not have a sibling probe (anymore) can be eliminated from the system. Note that this task is facilitated by the fact that erroneous decisions are not affecting the convergence of the tokens.

It should be mentioned that the condition for a token to send out probes will be true in a skewed state every time it visits a node. This condition can also hold in a nonskewed state and even in a stabilized state. But on the other hand, only one forward and one backward probe of each value is needed and there are only 3 values. Hence, constant buffer size at each node is sufficient (a node receiving a second probe before sending out the first can simply delete the second probe). Given the constant size of probes, the communication-overhead in a stabilized state is minimal.

4.3. Formal description of the algorithm. In the following, we present the different types of messages used in the algorithm:

- *positive tokens*: circulating clockwise on the ring;
- *negative tokens*: circulating counter-clockwise on the ring;
- *probes*: tokens can send out probes to avoid getting stuck in an skewed state. A token will send out probes in both directions. A probe circulating in the same direction as its father token is called a *forward probe*, otherwise it is called a *backward probe*.

Next we present the data-structures used by the algorithm:

- $V(i)$: value of node i ($1 \leq |V(i)| \leq 3$).
- $V(t)$: value of token t ($1 \leq |V(t)| \leq 3$).
- $S(t)$: bit indicating that token t is currently sending out probes. This bit holds from the time a pair of probes is generated until t's next collision or turn. $S(t)$ is used as a guard for the turn operation.
- $L(t)$: value of node that t visited last.
- $V(p)$: value of probe p. $V(p)$ indicates the value of probe p's father-token. The test $V(t) = V(p)$ is used as a guard to turn a token t. The test $V(p_1) = V(p_2)$ is used to detect sibling probes upon a collision of two probes.
- $E(p)$: bit indicating whether the probe p is enabled.
- $T(p)$: if p is a backward probe and has decided to turn its father-token, then this bit is set.

Next we describe the algorithm by stating what is done on each message-driven event.

Positive token t_1 and negative token t_2 collide:
A1: IF $|V(t_1)| \neq |V(t_2)|$, THEN
A2: eliminate token with smaller absolute value
A3: ELSE
A4: $V(t_1) :=$ RANDOM($\{1, 2, 3\} - \{V(t_1)\}$);
A5: $V(t_2) := -V(t_1)$
A6: END;
A7: $S(t_1) :=$ FALSE; $S(t_2) :=$ FALSE;

Node i receives positive token t:
B1: IF $V(i) = V(t)$, THEN
B2: generate token t' (* negative *);
B3: $V(t) :=$ RANDOM($\{1, 2, 3\} - \{V(t)\}$);
B4: $V(t') := -V(t)$; $L(t') := V(i)$;
B5: $S(t) :=$ FALSE; $S(t') :=$ FALSE
B6: ELSIF $V(t) < V(i)$, THEN (* t follows a larger-valued, positive token *)
B7: generate forward probe p_1 and backward probe p_2;
B8: $V(p_{1,2}) := V(t)$; $E(p_{1,2}) :=$ TRUE;
B9: $T(p_{1,2}) :=$ FALSE;
B8: $S(t) :=$ TRUE
B9: END;
B10: $L(t) := V(i)$; $V(i) := V(t)$;
B11: send positive token t (and negative token if one newly generated)

Node i receives negative token t:
symmetric to the receipt of a positive token

Node i receives forward probe p:
C1: send probe p

Node i receives backward probe p:
D1: IF $(V(i)V(p) < 0)$ AND $E(p)$, THEN
D2: $E(p) :=$ FALSE
D3: END;
D4: send probe p

Token t and probe p collide:
E1: IF $V(t)V(p) < 0$, THEN (* p is a forward probe *)
E2: IF $E(p)$, THEN $E(p) :=$ FALSE
E3: ELSE eliminate p
E4: END
E5 ELSE (* p is a backward probe *)
E6: IF $\neg E(p)$, THEN eliminate p
E7: ELSIF $E(p)$ AND $T(p)$, THEN
E8: IF $(V(p) = V(t))$ AND $S(t)$, THEN
E9: turn t;
E10: $V(t) := -L(t)$;
E11: $S(t) :=$ FALSE
E12: END;
E13: eliminate p
E14: END
E15: END

Forward probe p_1 and backward probe p_2 collide:
F1: IF $V(p_1) = V(p_2)$, THEN
F2: IF $E(p_1)$ AND $E(p_2)$, THEN
F3: $T(p_2) :=$ TRUE;
F4: eliminate p_1
F5: ELSE eliminate p_1 and p_2
F6: END
F7: END

Note that we make use of a function RANDOM(S) which returns a random value drawn from the set S. We only use it for sets of size 2 and thus it can be thought to be a fair coin.

5. Analysis. In this section we give a correctness proof of the TM algorithm introduced in the previous sections, and we analyze its stabilization time.

Let $t(n)$ be the expected stabilization time. The stabilization time is the time interval starting at the end of the last transient fault and ending when the system has

reached a legal state. Our main theorem is as follows.

THEOREM. *Algorithm A is a fair and nondeadlocking self-stabilizing TM algorithm for system S with* $t(n) = O(n \log n)$.

Since the proof is quite involved, we will precede its presentation with a subsection introducing all necessary definitions and a subsection giving an overview.

5.1. Definitions. We start with defining a global state of the system and the transitions among those global states. A global state is basically the value of each node, the value of each token, and the positions of the tokens on the ring. A transition from a state to its successor state consists of either a collision among two tokens, or a turn of a token, or a generation of a token. Further we define a local state of a ring segment with respect to the probes present on it. Note that probes are "invisible" in the global state. One can think of a local state as putting a magnifying glass on a segment of the ring.

DEFINITION 5.1 (local and global state, state transition, segment).

1. *The* global state *of the system is defined by*
   - *for each token t, its value* $V(t)$,
   - *the positions of the tokens on the ring,*
   - *for each node i, its value* $V(i)$.
2. *The* (global) state transition *from state s to state* s' *is defined by a* successor *function* $S : s' = S(s)$, *where* s' *results from s by applying one of the following transitions:* {*collision, turn, generation*}. *A node copying a new value from a passing token is strictly speaking also a state transition. But for our purposes this is not an "interesting" operation by itself. Let* $S^j(s)$ *be the* j*th successor of s.*
3. $|s|$ *denotes the number of tokens in state s.*
4. *A sequence of nodes starting at some node i and ending at some node j, such that* $V(i) = V(i+1) = \cdots = V(j)$ *is called a* segment σ *of value* $V(\sigma) = V(i)$. *We will say that a token or a probe is* on *segment* σ *if it is on a node k,* $i \le k \le j$, *or if it is on a link* $(l, l+1)$, $i \le l < j$. *We will use the symbol t alternatively for the token itself or for the segment the token t creates by leaving its footprints behind.*
5. *The* local state *of a segment* σ *in a global state s is defined by*
   - *its value* $V(\sigma)$,
   - *for each probes p on* σ, *its value* $V(p)$ *and its position. Let* $f^\sigma(s)$ $(b^\sigma(s))$ *denote the set of enabled forward probes (backward probes) on* σ.

In terms of the above definition we can now define the correctness condition for TM.

DEFINITION 5.2 (correctness). *For an* arbitrary *state s with* $|s| \ge 1$ *there are infinitely many successor-states and, in particular, there is a state* $S^l(s)$ *such that* $|S^l(s)| = 2$ *and every state* $S^{l'}(s)$ $(l' > l)$ *results from a collision of two tokens with identical absolute value.*

Since "initially" we have no control over the values stored in the nodes or on the tokens, we will define now a predicate over states, which will indicate that, although the system may not have stabilized yet (and indeed might have more tokens than "initially"), we have reached a certain level of order in the system. Intuitively, this means that every node has at least seen one token, and thus the neighborhood of a token is no longer influenced by the "initial" state.

DEFINITION 5.3 (cover). *We say that a state s has the predicate* cover, *i.e., cover(s), if*

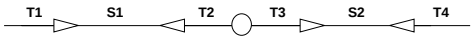


FIG. 4. *Sequence of pairs.*

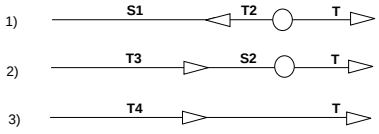


FIG. 5. *"Outgoing" covers.*

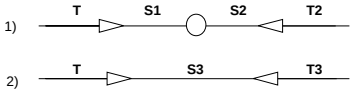


FIG. 6. *"Ingoing" covers.*

- $|s| \geq 2$;
- *the ring is a concatenation of segments, such that a segment boundary is either at the current position of a token or at a previous collision-point. Following the scenario in Figure* 4*, we enumerate all allowable neighborhoods first to the left of a (positive) token* t *and then to the right of* t*. ("$\triangleleft$", "$\triangleright$" denote tokens (and their direction), "$\bigcirc$" denote previous collision-points,* s*'s denote segments, and* t*'s denote tokens and the segment they are creating.) Tokens traveling from left to right are positive tokens and thus their value is always positive. Tokens traveling from right to left are negative tokens and carry a negative value.*
  The values in the scenarios of Figure 5 *are constrained as follows:* $|V(t2)| = |V(t)|$, $|V(s1)| \neq |V(t2)|$; $V(s2) = -V(t)$, $|V(t3)| \neq |V(s2)|$; $V(t4) \neq V(t)$.
  The values in the scenarios of Figure 6 *are constrained as follows:* $|V(s1)| \neq |V(t)|$, $V(s2) = -V(s1) : |V(t2)| \neq |V(s2)|$, $|V(s3)| \neq |V(t)| : |V(s3)| \neq |V(t3)|$.

We conclude this section with the following sequence of simple definitions.

DEFINITION 5.4 (fragment). *A* fragment *is a largest concatenation of segments, such that the structural requirement of a cover is still fulfilled and at least one token is present on this part of the ring.*

DEFINITION 5.5 (block). *We say that a segment* σ *has the predicate* block *in state* s*, i.e., block(*σ*,* s*), if all probes in* $b^{\sigma}(s)$ *have identical values.*

DEFINITION 5.6 (state as a sequence of pairs). *If for state* s *cover(*s*) holds, then we can view* s *alternatively as a sequence of pair of numbers. Every pair represents one token and the segment it is traveling on (through its* token *component and* segment

component).

For example, the cover in Figure 4 corresponds to the sequence $s = (t1, s1)(t2, s1)(t3, s2)(t4, s2)$.

5.2. Overview of the proof. The proof will be structured as follows: For every state s with $s \geq 1$ we define a *subgoal* and show that this subgoal will indeed be reached. For every possible sequence of these subgoals, the last one is always to reach stability. Thus, if the algorithm reaches for each state the desired subgoal, correctness is assured. The following is now a list of different states and their subgoals:

1. For a state s with $|s| \geq 1$ and $\neg cover(s)$ the subgoal is to show that there is a successor state $S^{l_1}(s)$ for which $cover(S^{l_1}(s))$ holds. Note that it is very well possible that $|S^{l_1}(s)| > |s|$ and thus, at first glance, the successor state seems to be farther away from our ultimate goal. However, the predicate cover implies that we have reached a higher level of order in the system, since as soon as cover holds, no generation transition can take place anymore and cover is a stable predicate. Thus, it guarantees that the system does not move away anymore from its goal state (by generating additional tokens). Another important fact is that any state with a cover has a successor state. If the state is skewed, this property holds thanks to our probes scheme.
2. For a state s with $cover(s)$, $|s| > 2$, and $|s|$ being odd, the subgoal is to show that there is a successor state $S^{l_2}(s)$ such that either $|S^{l_2}(s)| < |s|$ or $\Pr(|S^{l_2}| < |s|) > 0$. If the first disjunctive part of our goal holds, then we have made direct progress towards the goal state; otherwise, we must have at least reached a state in which we could have made progress, i.e., there was a nonzero probability to reduce the number of tokens in the transition to this state. This probability follows from the fact that in a state with an odd number of tokens, there will be always at least one token which collides consecutively with at least two other tokens. Since in the first collision it picked up a random value independent of the rest of the system, the subgoal follows.
3. For a state s with $cover(s)$, $|s| > 2$, and $|s|$ being even, the subgoal is to show that either $|S^{l_3}(s)| < |s|$ or $\Pr(|S^{l_3}(s)| < |s|) > 0$ or there is a pair of tokens (t_1, t_2) of the following form $(t_1, \sigma_1)(t_2, \sigma_2)$ such that $\Pr(block(\sigma_2, S^{l_3}(s))) > 0$. Thus, in the case in which there is an even number of tokens, we restrict ourselves to an even more modest goal. This is necessary since tokens can (at least temporarily) form pairs such that every token collides only with its partner and then turns and repeats this cycle. However note that in order to turn, a token needs to be supplied with probes. Now if a *block* of probes is on the adjacent segment of a token, then the token, which carries a random value independent of the rest of the system, has a nonzero probability to mismatch these probes and will break out of its local cycle.
4. For a state s with $|s| = 2$, $cover(s)$, the subgoal is to show that the leftover probes will eventually be removed from the system. This task is considerably simplified by the fact that as long as the system is in a state with more than two tokens, we are allowed to make "mistakes" with respect to this task, since we have shown convergence to a state with two tokens for arbitrary configurations of probes.
5. For a state s with $s = (t_1, \sigma_1)(t_2, \sigma_2)$, we show that if there are no leftover probes, we have reached stability.

The rest of the proof basically ties all the above subgoals together and shows that in an infinite sequence of states, the desired stable state is reached with probability 1.

5.3. Correctness proof. In order to prove our main theorem, we will introduce a series of lemmas. The first lemma addresses the nondeadlocking property of the algorithm.

LEMMA 5.7. *For every state s for which $|s| \geq 1$, $\forall l > 0$, $|S^l(s)| \geq 1$.*

Proof. The only condition on which a token is eliminated is the collision of two tokens (line A1 in the algorithm). Thus, every time a token is eliminated another token survives. □

The following lemma proves that we reach our first subgoal, a state s for which $cover(s)$ holds.

LEMMA 5.8. *For every state s, for which $|s| \geq 1$, there exists a finite $l > 0$ such that $cover(S^l(s))$ holds.*

Proof. First we need to show that every state s, for which $\neg cover(s)$ holds, has a successor state: (i) s is skewed, $|s| \geq 2$: Assume that no transition takes places. Then eventually $s = (t_1, \sigma_1)(t_2, \sigma_2) \dots (t_{|s|}, \sigma_{|s|})$, where all $V(t_i)$ are either positive or negative and $\sigma_i = t_{i+1}$. Thus, as can be easily seen, $cover(s)$ holds. (ii) s is not skewed, $|s| \geq 2$: There are always two opposite tokens facing each other with no other token in between. Thus, either a turn or a collision transition is imminent. (iii) $|s| = 1$: If the token does not turn, then a generation transition will take place after at most one round.

Consider a link $(i, i+1)$ as a pair of values of its end-nodes $(V(i), V(i+1))$ together with the values of tokens which might be in transit on this link. Based on this information, we can decide for every link whether it could be a part of a fragment, i.e., the link is *consistent*.

First note that if a token has crossed a link and updated the value of the destination-node, the link will become consistent. Thus, a token can enlarge a fragment by crossing the first inconsistent link outside of its fragment. In that case, we say that the token is at the *border* of its fragment. Further, every generation, collision, and turn transition preserves the fragment in which (or at whose border) it occurs (see lines A1–A6, B2–B4, E9–E10). Hence the consistency of a link is a stable property.

Consequently, it remains to show that eventually every link will be crossed by a token and that at least two tokens will be present: By Lemma 5.7, all successor states have at least one token and thus at least one fragment. Assume that there is a fragment which does not grow and assume that the token t at the border of this fragment is moving away from the border (if not it will either turn and thus move away or enlarge the fragment). At some point it will either turn or collide with another token t' (otherwise t would enlarge the fragment on the other border). Assume without loss of generality that in the latter case t' survives the collision and thus is now moving towards the border of the fragment. t left footprints of opposite sign and thus t' cannot turn until it reaches a node which t never visited. Thus, we have shown a contradiction.

If a state is reached with a single fragment spanning the whole ring and just one token is alive, then after at most one round a second token will be generated (condition on line B1), and thus a cover will have been reached. □

The next few lemmas show that the cover predicate is stable and well defined with respect to the successor function. We also show that no generation can take place in a state in which the cover predicate holds.

LEMMA 5.9. *Every state s, for which $cover(s)$ holds, has a successor state.*

Proof.

- s is skewed: If no transition takes place then eventually $s = (t_1, \sigma_1)(t_2, \sigma_2) \dots$

$(t_{|s|}, \sigma_{|s|})$, where $V(t_i)$ are either all positive or all negative and $\sigma_i = t_{i+1}$. Thus, for at least one token t_i the condition to send out probes (line B7 or the symmetric case) will be true every time t_i visits a node. t_i sends forward probes on σ_i and backward probes on t_i. Given the structure of s, these probes cannot be disabled (i.e., conditions on lines D1, C1, and E1 cannot become true). Thus, two enabled siblings (or probes from two tokens with the same value) will meet and thus the T-bit on the backward probe is set and a token (which still has the S-bit set) is turned.

- s is not skewed and consequently there is a fragment $(t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1) > 0$, $V(t_2) < 0$. Thus, a turn transition or a collision transition is imminent. □

LEMMA 5.10. *For every state s for which cover(s) holds, $|S(s)| \leq |s|$.*

Proof. By definition of a cover, the only condition on which the algorithm generates a new token (line B1 in the algorithm) cannot be enabled in s. □

LEMMA 5.11. *For every state s for which cover(s) holds, cover($S(s)$) holds.*

Proof. By Lemma 5.10, there is no generation transition leading out of s. Given that $|s| > 2$, every possible collision transition preserves the cover (see lines A1–A6). In every turn transition, the new absolute value of the turned token is the same as the value of the token it was following, and thus the result is identical to a collision (see lines E10 and B10). Also, if $|s| = 2$ and cover(s) holds, then, by definition of a cover, all collisions will take place between tokens of equal absolute value and so $|s|$ will always remain two. □

LEMMA 5.12. *For every state s, for which cover(s) holds, there exists a finite $l > 0$ such that $S^l(s)$ results from a collision transition.*

Proof. By Lemma 5.9, we have always a "next" transition, and by Lemma 5.10, there is no generation transition. If all successor states of s were results of turn transitions, then the sum of the absolute values of all tokens would steadily increase. Remember that the new value of a token t after its turn is set to $-L(t)$, where $L(t) = V(i)$ for i being the last node visited. And $S(t)$ is set only if $V(i) > V(t)$. Also, since cover(s) holds and t does not collide, it remains on the same segment from the time $S(t)$ is set until it turns. See lines (B10, B6, B7, E10). Since this sum is bounded, this is not possible. □

LEMMA 5.13. *Starting in any state s for which cover(s) holds, no token will be in two or more consecutive turn transitions.*

Proof. After a token t has been in a turn transition, its absolute value ($|V(t)|$) has been increased. Furthermore, it is traveling on a segment whose absolute value is t's old value (see the definition of *cover*), and thus $S(t)$ is false. Hence, as long as it is on this segment, t will not turn again. t can leave this segment only through a collision-transition. The lemma follows. □

LEMMA 5.14. *For every state s for which cover(s) holds, and for every token t in s, there exists a finite $l_t > 0$ such that $S^{l_t}(s)$ results from a collision transition involving token t.*

Proof. By induction on $|s|$, we have the following:

Basis: $|s| = 2$: follows immediately from Lemma 5.12.

Hypothesis: $|s| = n$.

Step: $|s| = n + 1$. Assume that token t will never be involved in a collision transition. Then by Lemma 5.13, t can be involved in only one turn transition. Thus, token t will be neither in a turn transition nor in a collision transition for infinitely many successor states. By definition of a cover, token t will eventually be in the

following fragment: $(t, \sigma_1)(t', \sigma_2)$, where $V(t)$ and σ_1 are either both positive or both negative and $t' = \sigma_1$. Now, if t' turns, then t is in a collision. If t' is in a collision where one token is eliminated, then we can apply the hypothesis. If no token is eliminated, then t is in a collision. Thus t' cannot be in a turn transition nor in a collision transition. It will eventually be in a analogous fragment as t. Then we can look at t''s right neighbor, etc. Finally, we will get a contradiction with Lemma 5.9 and thus our assumption that t will never be involved in a collision transition was wrong. ∎

The next lemma shows that any token which is involved in at least two consecutive collisions creates a nonzero probability that a token will be eliminated.

LEMMA 5.15. *The following two statements concern probabilistic elimination of tokens and hold in any state s with cover(s):*

(1) *If, starting in state s, there is at least one token which consecutively collides with at least two opposite tokens (let $S^k(s)$ be the state resulting from the second collision), then* $\Pr(|S^k(s)| < |s|) > 0$.
(2) *If* $\Pr(|S^k(s)| < |s|) = 0$, *then every token has collided with at most one opposite token during all transitions leading from s to $S^k(s)$.*

Proof. (1) The value the token carries after a first collision is random and independent of any other value in the system. Thus, there is a nonzero probability that in a second consecutive collision the tokens carry distinct absolute values, and consequently there is a nonzero probability that one of the tokens involved in the collision will be eliminated (see lines A1–A2). In addition, no new tokens are generated (by Lemma 5.10). (2) follows from the contraposition of (1) and Lemma 5.13. Note that a token needs to turn at least once between any two collisions to ensure that $\Pr(.) = 0$, and by Lemma 5.13 a token can turn at most once. ∎

The next lemma shows that our second subgoal will be reached.

LEMMA 5.16. *For every state s with odd$(|s|)$ and cover(s), there exists a finite $l > 0$, such that* $(|S^l(s)| < |s|) \vee (\Pr(|S^l(s)| < |s|) > 0)$.

Proof. By Lemma 5.14, every token t present in s will be eventually involved in a collision transition with another token t'. If one of these tokens is eliminated, then we are done. Otherwise the fragment of these two tokens right after the collision looks as follows: $(t, \sigma_1)(t', \sigma_2)$, where $V(t), V(\sigma_2) < 0$ and $V(t'), V(\sigma_1) > 0$. Since the number of tokens in s is odd, there will be a token t which, after its first collision with t', will be involved in a collision with a different token t'' ($t'' \neq t'$). By Lemma 5.13, there cannot be two or more consecutive turns by a token. Thus t must have been involved in two consecutive collisions. Now we can use Lemma 5.15 and we are done. ∎

DEFINITION 5.17 (partner, pair, local cycle, neighbor).

- *Starting in some state s for which cover(s) and even$(|s|)$ hold, $|s| > 2$, we call two tokens involved in their first collision* partners *with respect to state s, and the function partner$_s(t)$ yields the partner of token t with respect to state s. By Lemma* 5.14*, this function is well defined. Both partners together form a* pair.
- *A* local cycle *of a pair is a sequence of the three transitions which brings both tokens back to the same relative position: Each token is involved in a turn transition and in a collision transition with its partner. Note that if both tokens started out with the same value, this is exactly the scenario for which the probability of decreasing the number of tokens is zero (Lemma* 5.15*).*
- *The two tokens on the left and on the right (which might coincide) of a pair are called* neighbors *of that pair.*

The next few lemmas provide useful facts about pairs of tokens and the local state of adjacent segments.

LEMMA 5.18. *Let s be a state, such that cover(s) holds, $|s| > 2$, and even($|s|$) holds. If in s the neighbors of a pair are not a pair, or if a pair does not correspond to one of the following fragments, then there exists a finite $l > 0$, such that $(|S^l(s)| < |s|) \vee (\Pr(|S^l(s)| < |s|) > 0)$. For any pair the fragment is in one of the following scenarios. $(t, \sigma_1)(partner_s(t), \sigma_2)$ with the following restrictions:*

1. *Pair is about to collide: $V(t) > 0$, $V(partner_s(t)) < 0$, $|V(t)| = |V(partner_s(t))|$, $\sigma_1 = \sigma_2$.*
2. *Pair right after collision: $V(t) < 0$, $V(partner_s(t)) > 0$, $|V(t)| = |V(partner_s(t))|$.*
3. *Both tokens in positive direction: $V(t) > 0$, $V(partner_s(t)) > 0$, $|V(t)| > |V(partner_s(t))|$, $|V(\sigma_2)| = |V(t)|$.*
4. *Both tokens in negative direction: $V(t) < 0$, $V(partner_s(t)) < 0$, $|V(t)| < |V(partner_s(t))|$, $|V(\sigma_1)| = |V(partner_s(t))|$.*

Proof. If the neighbors are not a pair, an argument similar to the proof of Lemma 5.16 shows that a token must be involved in two consecutive collisions. If a pair is not in one of the scenarios above, a collision transition with an elimination is imminent. □

LEMMA 5.19. *If in state s (cover(s), $|s| > 2$, even($|s|$) both hold) a pair is in scenario 2, then for every probe $p \in b^{\sigma_1}(s)$ (where $V(\sigma_1) < 0$) the value $V(p)$ is mutually independent from $V(t)$ (and symmetrically for $p \in b^{\sigma_2}(s)$).*

Proof. For every probe $p \in b^{\sigma_1}(s)$ we can conclude that token t cannot have produced p since its last collision because its partner creates a segment of opposite sign (cf. scenario 2). The lemma follows now since the value $V(t)$ is random and mutually independent from every other value in the system. □

LEMMA 5.20. *If in state s (cover(s), $|s| > 2$, even($|s|$) both hold) a pair is in scenario 2, and for at least one of t or $partner_s(t)$ the next transition is not a turn, then there exists a finite $l > 0$, such that $(|S^l(s)| < |s|) \vee (\Pr(|S^l(s)| < |s|) > 0)$.*

Proof. Assume without loss of generality that t's next transition is not a turn. By Lemma 5.14, every token will be involved in a collision leading into a successor state (and thus every token is involved in a future transition). From this it follows that t will be involved in a future transition and the only transition which is possible is a collision. If the other token involved in this particular transition is not $partner_s(t)$, then we can apply Lemma 5.15; if this token is indeed $partner_s(t)$, then, since $|s| > 2$, $partner_s(t)$ must have been in at least two consecutive collisions, and thus we can also apply Lemma 5.15 with respect to $partner_s(t)$. □

The next lemma shows that our third subgoal is reached.

LEMMA 5.21. *If the system is in state s, cover(s) holds, $|s| > 2$, even($|s|$) holds, and a pair is in scenario 2, where $V(\sigma_2) > 0$, then there exists a finite $l > 0$, such that either $(|S^l(s)| < |s|) \vee (\Pr(|S^l(s)| < |s|) > 0)$ or $\Pr(block(\sigma_2, s^l)) > 0$.*

Proof. If either t's or $partner_s(t)$'s next transition is not a turn, then we can apply Lemma 5.20. Otherwise the transitions of t and $partner_s(t)$ form a local cycle. In order to maintain this cycle, both tokens must continuously turn and thus use probes in $|b^{\sigma_1}(s)|$ and $|b^{\sigma_2}(s)|$.

So let us consider how $|b^{\sigma_2}(S^k(s))|$ can change for $k > 0$: First note that the neighbors also will form a local cycle (otherwise we can again invoke Lemma 5.20). This implies that there will always be new "incarnations" of σ_2 with initially $|b^{\sigma_2}(S^k(s))| = 0$. Note that all probes entering σ_2 are either generated by the neighbors or passed

through the neighbors, and thus their fragment is in scenario 3 (otherwise the probes would get disabled). In order to pass through this fragment, the incoming backwards probes must mismatch all probes in $f^{\sigma_4}(s)$, all of which originated at token $partner_s(t')$. If incoming backwards probes originated with $partner_s(t')$, then they obviously match, otherwise they match with nonzero probability, since the forward probe carries a random value independent of the value of the backward probe (by Lemma 5.19). Now if the first incoming probe matches, only probes generated by $partner_s(t')$ will be on $partner_s(t')$ and consequently on σ_2. Thus for some finite $l > 0$, $\Pr(block(\sigma_2, s^l)) > 0$, or, in other words, the neighboring pair acts as a *probabilistic filter* with respect to incoming backwards probes.

Note that if there are any disabled probes on σ_2 which match any of the enabled probes, then they will merely eliminate the enabled probes and thus not interfere with the block predicate. □

LEMMA 5.22. *For every state s ($cover(s)$, $|s| > 2$, $even(s)$ both hold), there exists a finite $l > 0$, such that either* $(|S^l(s)| < |s|) \vee (\Pr(|S^l(s)| < |s|) > 0)$.

Proof. By Lemma 5.14, there is a successor state in which a pair is guaranteed to be in scenario 2, where $V(\sigma_1) < 0$, $V(\sigma_2) > 0$. By Lemma 5.21, if $\neg((|S^l(s)| < |s|) \vee (Pr(|S^l(s)| < |s|) > 0))$, then there is a nonzero probability that there is a block on at least one of σ_1 or σ_2. By Lemma 5.19, the value of the block is independent of the token's value. Thus, there is a nonzero probability that the block has a different value than the token, and thus the next transition of the token is not a turn. Using Lemma 5.20 and noting that the probability for a block on σ_2 and the probability that there will be an elimination in the subsequent collision (second in a row) are independent, we are done. □

The last few lemmas concern the last two subgoals, i.e., states with two tokens.

LEMMA 5.23. *For every state s with $cover(s)$, $|s| = 2$, there is a finite $l > 0$, such that $S^l(s) = (t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1), V(\sigma_2) > 0$, $V(t_2), V(\sigma_1) < 0$, and $V(t_1) < V(\sigma_2)$.*

Proof. Lemma 5.12 guarantees that there will always be a state s' as described above except maybe for the condition $V(t_1) < V(\sigma_2)$. If this condition does not hold, then by simply following the algorithm, we can conclude that the full condition holds in either $S(s')$ or $S^2(s')$. □

LEMMA 5.24. *If there are no enabled probes at the time of a transition into a state s of the form of Lemma* 5.23*, then the system has reached stability.*

Proof. Remember that $s = (t_1, \sigma_1)(t_2, \sigma_2)$, where $V(t_1), V(\sigma_2) > 0$, $V(t_2), V(\sigma_1) < 0$, and $V(t_1) < V(\sigma_2)$. Assume without loss of generality that just before the next transition, $s = (t_1, \sigma_2)(t_2, \sigma_2)$. Thus t_1 is sending out probes ($S(t_1) = \text{TRUE}$). All of its forward probes are in $f^{\sigma_2}(s)$ and all of its backward probes are in $b^{t_1}(s)$. Following the algorithm, we get $S(s) = (t_2, \sigma_3)(t_1, \sigma_4)$, where $V(\sigma_3) > 0$, $V(\sigma_4) < 0$. For all probes which were in $f^{\sigma_2}(s)$, the conditions on lines E1 and E2 became true before the transition and thus have been disabled. All these forward probes are now on σ_4. All enabled backward probes are now in $b^{\sigma_3}(S(s))$. Thus before the next transition, all backward probes will have been in a collision with a disabled forward probe, and thus both are eliminated (line F7). □

LEMMA 5.25. *For every state s as in the form of Lemma* 5.23*, with initial enabled probes, there is* (i) *a probability of at least* 1/2 *that those enabled probes cause one token to turn, in which case the system will be stabilized by $S^3(s)$, or else* (ii) *all enabled initial probes will be disabled.*

Proof. Trivially, $|b^{t_1}(s)| = |f^{t_1}(s)| = |b^{t_2}(s)| = |f^{t_2}(s)| = 0$. Without loss

of generality let us assume that just before the next transition $s = (t_1, \sigma_2)(t_2, \sigma_2)$. Thus we concentrate on $b^{\sigma_2}(s)$. If before t_1 reached σ_2, $|b^{\sigma_2}(s)| \neq 0$, then there is a probability of 1/2 for each of these probes that it will match t_1's forward probes (since $V(t_1)$ is random and independent). If indeed there is a matching probe, then $S(s) = (t_1, t_2)(t_2, t_1)$, and all probes which were in $b^{\sigma_2}(s)$ are disabled. By Lemma 5.9 $S^2(s)$ is reached by a turn of t_2. It is easy to see then that $S^3(s)$ is reached through a collision between t_1 and t_2 and there are no initial enabled probes.

Furthermore, it is easy to see that all initial probes (which we just showed are disabled by S^2) are removed from the system by $S^3(s)$. □

We now get to our first main theorem which ties all the previous lemmas together and shows the correctness of Algorithm A.

THEOREM 5.26. *Algorithm A is a correct self-stabilizing solution for* fair nondeadlocking round-robin TM *on system S. Furthermore, the lifetime of any control-signal, except for one pair of a positive and a negative token, is finite.*

Proof. The following chain of arguments proves that starting the system in an arbitrary state s, $|s| > 0$, will lead with probability 1 to a stable state $S^l(s)$ with $|S^l(s)| = 2$ $(l \geq 0)$.

By Lemma 5.8, $cover(S^{l_1}(s))$ holds for a finite l_1. If $|S^{l_1}(s)| > 2$, then by Lemma 5.16 and Lemma 5.22 there exists a finite l_2, such that $(|S^{l_2}(s)| < |S^{l_1}(s)|) \vee (\Pr(|S^{l_2}(s)| < |S^{l_1}(s)|) > 0)$. Also, by Lemma 5.10 and by Lemma 5.11, $|S^k(s)| \leq |S^{l_1}(s)|$ $\forall k > l_1$. Suppose now we have an infinite sequence of states such that $|S^m(s)| = |S^{l_1}(s)|$ $(\forall m \geq 0)$. The number of different states in the system is finite. So some state s' for which $|\Pr(|S(s')| < |s|) > 0$ is visited infinitely often. Therefore, the probability of the computation we have supposed is zero. This argument can be repeated until for some l_3 we have $|S^{l_3}(s)| = 2$. By Lemma 5.24, Lemma 5.23, and an argument similar to the above, there is now an l_4 for which $S^{l_4}(s)$ is stable. Using Lemma 5.25 together with the argument above, we can conclude that the lifetime of every signal except for one positive and one negative token is indeed finite. □

5.4. Stabilization-time. The following sequence of lemmas will show that the following theorem holds.

THEOREM 5.27. *Algorithm A has* $t(n) = O(n \log n)$.

LEMMA 5.28. *For every s a state* $S^l(s)$ *with* $cover(S^l(s))$ *is reached for the expected* $l = O(n)$.

Proof. From the proof of Lemma 5.8, it follows that the maximum time to enlarge a fragment is the time it takes for a token to traverse it. Also note that an initial probe either causes a token to turn within n time units or gets disabled. This holds since a probe which fails to turn a token will reach the opposite end of a fragment. If there is no change in sign of the nodes outside the fragment, the tokens on that fragment will not turn before a cover is reached. Also, if there is a change in sign, the probe gets disabled. □

LEMMA 5.29. *For every state s with* $cover(s)$*, there is an l whose expected worst case bound is* $O(n \log n)$*, such that* $|S^l(s)| = 2$.

Proof. Let $k = |s|$. If s is skewed, then after at most $O(n/k)$ time units a turn operation will take place. This holds by Lemma 5.9 and the fact that the maximum distance between two equal-valued tokens is $O(n/k)$ (important for nonconstant k). Now for every state which is not skewed a useful collision takes place within $O(n/k)$ expected time units (by Lemmas 5.16 and 5.22 and the fact that in a nonskewed state there are tokens going in opposite directions at distance at most $O(n/k)$). Thus, the expected time to reach a state with two tokens can be expressed as $\sum_{k=1}^{n} n/k =$

$O(n \log n)$. ∎

LEMMA 5.30. *For a state s, such that cover(s) holds and $|s| = 2$, the system stabilizes in $O(n)$ expected time.*

Proof. If there are initial probes, then after an expected constant number of rounds there will be a turn and then Lemma 5.25 applies, or all initial probes are disabled before such a turn can take place. ∎

6. Reduction from leader election (L) to TMA. Leader election (symmetry breaking) is an important procedure in many parallel and distributed computing scenarios (see, e.g., [AAGHK89, AAHK86, AB89, AM89, Ang80, Dij74, It90, IR81, FL84, PKR82, RL81, SS89]). This is especially true in our model, where the nodes in the ring are identical finite state machines. In this section we present an algorithm which, given a correct TMA-algorithm, achieves self-stabilizing leader election. We start with the appropriate definition.

DEFINITION 6.1. *A self-stabilizing leader election algorithm (L) is a self-stabilizing algorithm with respect to predicate P = "there exists a single leader" and, furthermore, after stabilization, the leader node remains unchanged.*

The control messages used by the L-algorithm are the tokens of the TMA-algorithm. These tokens are augmented essentially by a new data-field which indicates whether the token has seen (i) zero, (ii) one, or (iii) more than one leader since its last collision. If the system has stabilized, then, upon a collision, it will always be the case that one token will be in state (i) and one token will be in state (ii) (since together they have covered the ring exactly once since their last collision). If the system has not yet stabilized, it is possible that (a) both tokens are in state (i), in which case there is no leader, or (b) that both tokens are either in state (ii) or state (iii), which indicates that there are multiple leaders. For the L-algorithm we assume that a collision is always associated with a node. (The precise definition depends on the implementation of a collision.) Thus in case (a) a new leader is created on the node associated with the collision and in case (b) we also create a new leader and then put both tokens in a "delete mode" until their next collision. If a token passes through a node in delete mode, it will remove the leader-predicate from that node.

We use the tokens of our TMA-algorithm as control messages. We augment the messages and nodes with the following data-structures:

- $L(i)$: the bit indicating that node i is a leader.
- $L(t)$: the leader-value of token t $(0 \le L(t) \le 2)$.
- $D(t)$: the bit indicating that token t is in delete mode.

Tokens t_1 and t_2 collide on node i:

```
L1:   IF (L(t1) + L(t2) = 1) OR D(t1) OR D(t2), THEN
L2:     L(t1) = L(t2) := 0; D(t1) = D(t2) := FALSE
L3:   ELSIF L(t1) + L(t2) = 0, THEN
L4:     L(i) := TRUE;
L5:     L(t1) := 1; L(t2) := 0
L6:   ELSIF (L(t1) + L(t2) > 1), THEN
L7:     L(i) := TRUE;
L8:     D(t1) = D(t2) := TRUE;
L9:     L(t1) := 1; L(t2) := 0
      END IF
```

Token t arrives at node i:

```
D1:   IF D(t) AND L(i), THEN
D2:     L(i) := FALSE
D3:   END IF
```

THEOREM 6.2. *Algorithm L is a correct polynomial time space-preserving self-stabilizing reduction from TMA to leader election. Furthermore,* $p(n) = O(n)$.

Proof. We show that at the third collision there is exactly one node i for which $L(i)$ holds and no node j will change its value $L(j)$ from that point on: Note first that at the second collision either none or both of the tokens are in the delete mode, since the tokens agreed on the mode in their first collision (lines L2, L8). If they are in the delete mode, then they created a new leader at their first collision (line L7) and removed every other leader on the ring (line D2). Now assume that none is in the delete mode. Then, their L-values correctly reflect the number of leaders in the ring. Thus, if there is no leader, then one will be created (line L4), and if there is more than one leader, then one new leader is created (line L7) and all others will be removed by the time of the third collision. □

7. Reduction from TMA to L. In this section we present an algorithm which, given a correct L-algorithm, achieves self-stabilizing TMA.

The algorithm is simple. We maintain one bit at the leader and one bit on each token. If a token arrives at the leader and its bit mismatches the leader bit, the token is removed; otherwise the leader chooses a new bit at random and the token copies it. Every time a token is removed the leader generates a counter-token in the opposite direction. This token is used to find out whether the leader just removed the last token from the system. (This can happen as initially the bits on the token and leader are arbitrary.) If a counter-token collides with a token, it is removed, and if a counter-token arrives at the leader, the leader will generate a new token (and remove the counter-token).

We use tokens (circulating in one direction) and counter-tokens (circulating in the opposite direction) as control messages and we use the leader of our L-algorithm. We augment the messages and the leader node with the following data-structures:

- $B(l)$: the bit at leader (l) indicating the current value of the token.
- $B(t)$: the value of token t.

Token t arrives at leader l:
T1: IF $B(l) \neq B(t)$, THEN
T2: eliminate t;
T3: generate counter-token;
T4: ELSE
T5: $B(l) :=$ random$(0, 1)$; /* fair coin */
T6: $B(t) := B(l)$

Counter-token t arrives at leader l:
T7: generate token t;
T8: $B(t) := B(l)$

Token t and counter-token t' collide:
T9: delete t'.

The above algorithms establishes the following theorem.

THEOREM 7.1. *Algorithm TMA is a correct polynomial time space-preserving self-stabilizing reduction from leader election to TMA. Furthermore,* $p(n) = O(n \log n)$.

Proof. First note that after n time units all present counter-tokens have actually been generated by the leader, and thus a token is generated if and only if there are no tokens in the system (since a counter-token is eliminated upon a collision with a token (line T9) and if a counter-token arrives at the leader, a token is generated (line T7)).

Let k be the number of tokens in the system at this point. From the randomization used by the leader (line T4), it is easy to see that n time units later, the expected number of tokens in the system is $k/2$. Finally, note that the last token will not be removed as its bit always matches the bit of the leader (line T1). □

8. Applications. Our work is derived from the needs of actual systems. The main goals are the development of self-stabilizing mechanisms for high-speed local area ring networks (LANs) and embedded rings on general topology networks. Note that many existing and suggested LAN architectures, such as FDDI [R86], MetaRing [CO90], Cambridge LAN [HN88], Magnet [LTG90], and ATM-ring [OMS89], have ring-based topologies. Also, a recent suggestion for control on a general topology network is to embed a virtual ring to support on-line high-speed control mechanisms [OY90]. Having self-stabilizing mechanisms can prevent potentially costly centralized and duplicated monitoring and recovery protocols. (For the high cost of such mechanisms, see, e.g., [BCK+83].) As explained in section 1.1, for such architectures, e.g., [R86, CO90, OY90], we need hardware-oriented algorithms to support fast on-line processing at a low-level protocol.

In particular, *TMA* is an important regulation control task in well established (e.g., token ring) and more recently suggested (e.g., MetaRing [CO90]) LANs. MetaRing employs a variant of a token, called the *SAT-token*. The SAT-token distributes transmission permits or quotas to the active nodes, and as a result, multiple nodes can access the network at the same time. In this new scheme a node will hold the SAT-token only if it is not satisfied. (It could not send the quota given to it by the SAT-token during its previous visit.) It has been shown that in high-speed implementations, the efficiency and effectiveness of this new fairness scheme increase as the transfer delay of the SAT-token decreases. Since the Meta-Ring (unlike token ring) allows concurrent transmission, it has been further shown that when the SAT-token is used, the fairness and correct operation are preserved even during stabilization time, when there are several SAT-tokens in the system, as long as the TMA scheme is round-robin and fair.

Acknowledgments. We are happy to thank Yehuda Afek, Baruch Awerbuch, Amotz Bar-Noy, Shay Kutten, Sergio Rajsbaum, and Baruch Schieber for helpful discussions. We are also grateful to the anonymous referees for their help in greatly improving the exposition of this paper.

REFERENCES

[AAGHK89] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick, *The bit complexity of randomized leader election on a ring*, SIAM J. Comput., 18 (1989), pp. 12–29.

[AAHK86] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick, *Probabilistic solitude verification on a ring*, in Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, 1986.

[AAHK91] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick, *Probabilistic leader election on rings of known size*, in Lecture Notes in Comput. Sci. 519, Springer-Verlag, New York, 1991, pp. 481–495.

[AB89] Y. Afek and G.M. Brown, *Self-stabilization over unreliable communication media*, Distrib. Comput., 7 (1993), pp. 27–34.

[AG92] A. Arora and M. Gouda, *Closure and convergence: A foundation for fault-tolerant computing*, in Proceedings of the 22nd International Conference on Fault-Tolerant Computing Systems, IEEE, 1992.

[AM89] Y. Afek and Y. Matias, *Simple and efficient election algorithms for anonymous networks*, in Proceedings of the 3rd International Workshop on Distributed Algorithms, Vol. 393, Springer-Verlag, New York, 1989, pp. 183–194.

[AKY90] Y. Afek, S. Kutten, and M. Yung, *Memory-efficient self-stabilization on general networks*, in Proceedings of the 4th International Workshop on Distributed Algorithms, Vol. 486, Springer-Verlag, New York, 1990, pp. 15–28.

[Ang80] D. Angluin, *Local and global properties in networks of processors*, in Proceedings of the 12th ACM Symposium on Theory of Computing, 1980.

[AO94] B. Awerbuch and R. Ostrovsky, *Memory efficient and self stabilizing network reset*, in Proceedings of the 14th ACM Symposium on Principles of Distributed Computing, 1994.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilization by local checking and correction*, in Proceedings of the 33nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 268–277.

[BGW89] G.M. Brown, M.G Gouda, and C.L. Wu, *Token systems that self-stabilize*, IEEE Trans. Comput., 38 (1989), pp. 845–852.

[BP88] J.E. Burns and J. Pachl, *Uniform self-stabilizing rings*, ACM TOPLAS, 11 (1989), pp. 330–344.

[BCK+83] W. Bux, F.H. Closs, K. Kümmerle, H.J. Keller, and H.R. Müller, *Architecture and design of a reliable token-ring network*, IEEE J. Selected Areas Comm., 1 (1983), pp. 756–765.

[CO90] I. Cidon and Y. Ofek, *MetaRing - A full-duplex ring with fairness and spatial reuse*, IEEE Trans. Comm., 41 (1993), pp. 110–120.

[Dij73] E.W. Dijkstra, *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, New York, 1982, pp. 41–46.

[Dij74] E.W. Dijkstra, *Self-stabilizing systems in spite of distributed control*, Comm. ACM, 17 (1974), pp. 643–644.

[DIM91] S. Dolev, A. Israeli, and S. Moran, *Uniform dynamic self-stabilizing leader election*, in Proceedings of the 5th International Workshop on Distributed Algorithm, Vol. 579, Springer-Verlag, 1991, pp. 167–180.

[FL84] G.N. Frederickson and N. Lynch, *The impact of synchronous communication on the problem of electing a leader*, J. Assoc. Comput. Mach., 34 (1987), pp. 98–115.

[Her90] T. Herman, *Probabilistic self-stabilization*, Inform. Process. Lett., 35 (1990), pp. 63–67.

[HN88] A. Hopper and R. M. Needham, *The Cambridge fast ring networking system*, IEEE Trans. Comput., 37 (1988), pp. 1214–1223.

[IJ90a] A. Israeli and M. Jalfon, *Token management schemes and random walks yield self stabilizing mutual exclusion*, in Proceedings of the 9th ACM Symposium on Principles of Distributed Computing, 1990.

[IJ90b] A. Israeli and M. Jalfon, *Self-stabilizing ring orientation*, in Proceedings of the 4th International Workshop on Distributed Algorithms, Vol. 484, Springer-Verlag, 1990, pp. 1–14.

[IR81] A. Itai and M. Rodeh, *Symmetry breaking in distributive networks*, Inform. and Comput., 88 (1990), pp. 60–87.

[It90] A. Itai, *On the computational power needed to elect a leader*, in Proceedings of the 4th International Workshop on Distributed Algorithms, Vol. 484, Springer-Verlag, 1990, pp. 29–40.

[IL94] G. Itkis and L. Levin, *Fast and lean self-stabilizing asynchronous protocols*, in Proceedings of the 35th IEEE Annual Symposium on Foundation of Computer Science, 1994.

[ILS95] G. Itkis, C. Lin, and J. Simon, *Determinstic, constant space, self-stabilizing election on uniform rings*, in Proceedings of the 9th International Workshop on Distributed Algorithms, 1995.

[KP90] S. Katz and K. J. Perry, *Self-stabilizing extensions for message-passing systems*, Distrib. Comput., 7 (1993), pp. 17–26.

[LL90] L. Lamport and N.A. Lynch, *Distributed computing: Models and methods*, in Handbook on Theoretical Computer Science, Vol. B, Elsevier, Amsterdam, 1990, pp. 1157–1199.

[LTG90] A.A. Lazar, A.T. Temple, and R. Gidron, *MAGNET* II: *A metropolitan area network based on asynchronous time sharing*, IEEE J. Selected Areas Comm., 8 (1990), pp. 1582–1594.

[Mis83] J. Misra, *Detecting termination of distributed computations using markers*, in Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, 1983.

[MOY96] A. Mayer, R. Ostrovsky, and M. Yung, *Self-stabilizing algorithms for synchronous unidirectional rings*, in Proceedings of the Seventh Annual ACM-SIAM

Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1996, pp. 564–573.

[OY90] Y. Ofek and M. Yung, *Principles for high-speed network control*, in Proceedings of the 9th ACM Symposium on Principles of Distributed Computing, 1990.

[OMS89] H. Ohnishi, N. Morita, and S. Suzuki, *ATM ring protocol and performance*, in Proceedings of the IEEE Conference on Communications, 1989.

[PKR82] J. Pachl, E. Korach, and D. Rotem, *A technique for proving lower bounds for distributed maximum-finding algorithms*, in Proceedings of the 14th ACM Symposium on Theory of Computing, 1982.

[PY94] G. Parlati and M. Yung, *Non-exploratory self-stabilization for constant-space symmetry breaking*, in Algorithms - ESA'94, Lecture Notes in Comput. Sci. 855, J. van Leeuwen, ed., Springer-Verlag, Berlin, 1994, pp. 183–201.

[RL81] M.O. Rabin and D. Lehmann, *On the advantage of free choice: A symmetric solution to the dining philosophers problem*, in Proceedings of the ACM Symposium on Principles of Programming Languages, 1981.

[R86] F.E. Ross, *FDDI - A tutorial*, IEEE Communication Magazine, 24 (1986), pp. 10–17.

[Schn93] M. Schneider, *Self-stabilization*, ACM Comput. Surveys, 25 (1993), pp. 45–67.

[SS89] B. Schieber and M. Snir, *Calling names on nameless networks*, in Procedings of the 8th ACM Symposium on Principles of Distributed Computing, 1989.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1596–1607

APPROXIMATING THE LONGEST CYCLE PROBLEM IN SPARSE GRAPHS*

TOMÁS FEDER†, RAJEEV MOTWANI‡, AND CARLOS SUBI§

Abstract. We consider the problem of finding long paths and cycles in Hamiltonian graphs. The focus of our work is on sparse graphs, e.g., cubic graphs, that satisfy some property known to hold for Hamiltonian graphs, e.g., k-cyclability.

We first consider the problem of finding long cycles in 3-connected cubic graphs whose edges have weights $w_i \geq 0$. We find cycles of weight at least $(\sum w_i^a)^{\frac{1}{a}}$ for $a = \log_2 3$. Based on this result, we develop an algorithm for finding a cycle of length at least $m^{(\log_3 2)/2} \approx m^{0.315}$ in 3-cyclable graphs with vertices of degree at most 3 and with m edges. As a corollary of this result, for arbitrary graphs with vertices of degree at most 3 that have a cycle of length l (or, more generally, a 3-cyclable minor with degrees at most 3 and with l edges), we find a cycle of length at least $l^{(\log_3 2)/2}$.

We consider the graph property of 1-*toughness* that is common to Hamiltonian graphs and 3-connected cubic graphs, and we try to determine if 1-toughness implies the existence of long cycles. We show that 2-connectivity and 1-toughness, for constant degree graphs, may give cycles that are only of logarithmic length. However, we exhibit a class of 3-connected 1-tough graphs with degrees up to 6, where we can find cycles of length at least $m^{\log_3 2}/2$.

Key words. long paths and cycles, Hamiltonian graphs, approximation algorithms

AMS subject classifications. 68R10, 68W25

PII. S0097539701395486

1. Introduction.

Over the last decade, there has been tremendous progress in the area of approximation algorithms. For most canonical NP-hard problems, either dramatically improved approximation algorithms have been devised, or strong negative results have been established, leading to a substantially improved understanding of the approximability of these problems. However, there is one problem which has resisted all attempts at devising either positive or negative results—longest paths and cycles in undirected graphs. Essentially, there is no known algorithm which guarantees an approximation ratio better than $n/\text{polylog}(n)$, and there are no hardness of approximation results that explain this situation. This is true even for the problem of finding long paths and cycles in bounded-degree Hamiltonian graphs, and indeed it has been conjectured that this special case is already very hard to approximate.

We examine the following question: If a graph has a Hamiltonian cycle, what is the longest cycle that can be found in polynomial time? Our approach is to select properties of Hamiltonian graphs and use them to construct long cycles. For instance, Hamiltonian graphs are k-cyclable for all k; i.e., any k vertices lie on a common cycle. For 3-cyclable graphs with vertices of degree at most 3, we find a cycle of length at least $m^{(\log_3 2)/2}$. Before describing these and other results in greater detail, we briefly review the previous work in this area.

*Received by the editors September 23, 2001; accepted for publication (in revised form) April 17, 2002; published electronically August 5, 2002. A preliminary version of this paper has appeared as an extended abstract in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, ACM, New York, 2000, pp. 524–529.

http://www.siam.org/journals/sicomp/31-5/39548.html

†268 Waverely Street, Palo Alto, CA 94301 (tomas@theory.stanford.edu).

‡Department of Computer Science, Stanford University, Gates Building 4B, Stanford, CA 94305 (rajeev@ cs.stanford.edu). This author's research was supported in part by NSF grant IIS–0118173, an Okawa Foundation research grant, and Veritas.

§738 Cowper Street, Palo Alto, CA 94301 (carlos_subi@hotmail.com).

Previous work. The first approximation algorithm was due to Monien [11], who gave a polynomial-time algorithm for finding paths of length $\Omega(\log n/\log\log n)$ in Hamiltonian graphs. Fürer and Raghavachari [7, 8] and Karger, Motwani, and Ramkumar [10] improved this to find paths of length $\Omega(\log n)$ in Hamiltonian graphs. Karger, Motwani, and Ramkumar also pointed out that the Hamiltonian cycle problem appears difficult even in *cubic* graphs and devised an algorithm for finding paths of length $\Omega(\sqrt{n}\log n)$ in *random* cubic Hamiltonian graphs. Alon, Yuster, and Zwick [1] presented an algorithm for finding paths of length $c\log n$ for any constant c as long as the graph had such a path (no assumption of Hamiltonicity). A recent result of Vishwanathan [14] gives an algorithm for finding paths of length $\Omega((\log n/\log\log n)^2)$ in Hamiltonian graphs.

The situation from the point of view of hardness results is equally dismal. Karger, Motwani, and Ramkumar [10] showed that, unless P = NP, it is impossible to find paths of length $n - n^\epsilon$ in Hamiltonian graphs for any ϵ. They also established that there is no constant-factor approximation algorithm for the longest path problem (unless P = NP) and that, if there is a polynomial-time algorithm with ratio $2^{O(\log^{1-\epsilon} n)}$, then NP $\subseteq$ DTIME$(2^{O(\log^{1/\epsilon} n)})$. They extend the latter result to bounded-degree graphs and conjecture that the problem is as hard even in bounded-degree Hamiltonian graphs. More recent work, of Bazgan, Santha, and Tuza [3], establishes that the longest path problem cannot be approximated within constant factors in *cubic* Hamiltonian graphs. They also showed that, for any $\epsilon > 0$, this problem is not approximable within ratio $2^{O(\log^{1-\epsilon} n)}$ unless NP $\subseteq$ DTIME$(2^{O(\log^{1/\epsilon} n)})$.

Overview of results. The focus of our work is on sparse graphs, e.g., cubic graphs, that satisfy some property known to hold for Hamiltonian graphs, e.g., k-cyclability. It was conjectured by Bondy and Simonovits [4] that every 3-connected cubic graph has a cycle of length at least n^c for some constant $c > 0$. This was verified by Jackson [9], who showed that 3-connected cubic graphs have cycles of length at least n^c for $c = \log_2(1+\sqrt{5}) - 1 \approx 0.69$. A key tool in our work is a constructive and generalized version of this result. In section 2, we consider the problem of finding long cycles in 3-connected cubic graphs whose edges have weights $w_i \geq 0$. We find cycles of weight at least $(\sum w_i^a)^{\frac{1}{a}}$ for $a = \log_2 3$; the exponent $\frac{1}{a} \approx 0.63$ is smaller than that of Jackson, but the proof is much simpler and constructive and also gives the first result for the weighted case. (We note that the best upper bound known [4] for 3-connected cubic graphs has exponent $\log 8/\log 9 \approx 0.96$.)

Based on our algorithm for weighted cycles in 3-connected cubic graphs, together with an algorithm given in section 3 for finding a cycle through three given edges in 3-connected cubic graphs when such a cycle exists, we describe in section 4 an algorithm for finding cycles of length at least $m^{(\log_3 2)/2}$ in 3-cyclable graphs with vertices of degree at most 3. As a corollary, we find a cycle of length at least $l^{(\log_3 2)/2}$ in any graph with vertices of degree at most 3 that has a 3-cyclable minor with degrees at most 3 and with l edges. (Note that a cycle of length l is such a minor.) A similar approximation for the longest path problem in graphs with vertices of degree at most 3 follows from this result.

In section 5, we consider the graph property of 1-*toughness* that is common to Hamiltonian graphs and 3-connected cubic graphs, and we try to determine if 1-toughness implies the existence of long cycles. We show that 2-connectivity and 1-toughness, for constant degree graphs, may give cycles that are only of logarithmic length and only slightly longer paths. However, we exhibit a class of 3-connected 1-tough graphs with degrees up to 6 where we can find cycles of length at least $m^{\log_3 2}/2$.

2. Weighted 3-connected cubic graphs. In this section, we establish the following result.

THEOREM 2.1. *Let $G(V, E)$ be an n-vertex, 3-connected, cubic graph. Suppose that each edge $e_i \in E$ has associated with it a weight $w_i \geq 0$. Then G has a cycle of total weight at least*

$$L_a(w) = \left(\sum {w_i}^a\right)^{\frac{1}{a}}$$

for $a = \log_2 3 \approx 1.58$, and such a cycle can be found in polynomial time. In the unit weight case, this gives a cycle of length at least $(\frac{3}{2}n)^d$ for $d = \log_3 2 \approx 0.63$.

The rest of this section is devoted to the proof of this theorem. We first prove some relevant lemmas. The bound will follow from combining and dropping weights as follows.

LEMMA 2.2. *Consider a collection of weights $w_i \geq 0$. Suppose we repeatedly perform one of the following operations: either select two weights and replace them with their sum or select three weights, drop the smallest one, and replace the other two with their sum. In the end, when only one weight remains, it will be at least $L_a(w) = (\sum {w_i}^a)^{\frac{1}{a}}$ for $a = \log_2 3$.*

Proof. We show that the quantity $\sum {w_i}^a$ is not reduced by either operation. That is, if $0 \leq x \leq y \leq z$, then $x^a + y^a + z^a \leq (y + z)^a$. We can assume $x = y$. If $z = y$, then $3y^a = (2y)^a$. As z increases, $\frac{d}{dz} 2y^a + z^a = az^{a-1} \leq \frac{d}{dz}(y + z)^a = a(y + z)^{a-1}$. ∎

We shall simplify the graph by removing and combining edges so that the weights are treated as in Lemma 2.2. We define an operation on *cubic graphs* which selects an edge h, removes it from the graph, and then collapses the two resulting paths of length 2 into single edges. We will refer to this as *the H-transform applied at edge h*; the transformation is illustrated in Figure 2.1. Note that, when we apply the H-transform to a cubic graph G with n vertices and $3n/2$ edges, we will obtain a cubic graph G' with $n - 2$ vertices and $(3n/2) - 3$ edges.

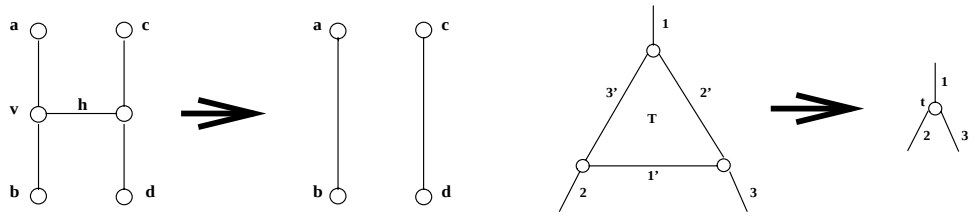


FIG. 2.1. *An illustration of the H-transform and the T-transform.*

LEMMA 2.3. *Let G be a 3-connected cubic graph with no triangles. Then there exists a vertex v such that applying the H-transform at any of the three edges incident on v will result in a new graph G', which is also a 3-connected cubic graph.*

Proof. We claim that applying the H-transform at an edge h of a 3-connected cubic graph G with no triangles results in a new graph G' which is also 3-connected and cubic if and only if the graph G does not have two other edges e, f, such that e, f, h are not all three incident to the same vertex, and removing e, f, h disconnects G.

The "only if" part is immediate since, if G has two such edges e, f, then, in G', the two edges e, f, or possibly edges obtained from them after collapsing paths of length 2 into single edges, will disconnect G', contrary to 3-connectivity.

For the "if" part, suppose that G' is not 3-connected. Then G' has two edges l, m that disconnect it. If l is a contracted path of length 2 from G, then the middle vertex of this path, on which h is incident, together with m disconnect G, contrary to the 3-connectivity of G. So h, l, m are edges in G that separate it, not all three incident to the same vertex, so these are the required edges e, f, h, completing the proof of the "if" part.

We can assume that G has three edges e, f, g whose removal disconnects it into two components K and K' having at least two vertices each. (Otherwise, any v will have the desired property, by the preceding claim.) Choose such a configuration with K minimal. Since K is not a triangle, it has some vertex v on which none of e, f, g is incident. Let h be an edge incident on v, drop h, and combine each of the two resulting paths of length 2 into single edges to obtain G'. Suppose G' is not 3-connected. Then, by the claim, G has two edges l, m such that h, l, m are edges in G that separate it, not all three incident to the same vertex. If either K or K', say, K', has exactly one of h, l, m in it, say, l, then l would have to separate one of e, f, g from the other two, say, e, but then e, l would separate G, contradicting 3-connectivity. So we can assume that none of h, l, m is in K'. However, this contradicts the minimality of K. □

While Lemma 2.3 assumed the absence of triangles, we must be able to handle triangles. To this end, we define an operation for removing a triangle $T = \{u, v, w\}$ which involves replacing the three vertices in T by a "supervertex" t whose three incident edges are the three edges incident on u, v, and w that were *not* a part of the triangle T (see Figure 2.1). It is obvious that applying this transformation to a cubic graph G will result in a cubic graph G'. Moreover, since any two edges separating G' would also separate G, this transformation also preserves 3-connectivity.

LEMMA 2.4. *Let $G \neq K_4$ be a 3-connected cubic graph having at least one triangle T. If T is replaced by a supervertex, then the resulting graph G' is also 3-connected and cubic.*

We now provide the proof of Theorem 2.1. Let G be a 3-connected cubic graph. If G has no triangle, then we select v as in Lemma 2.3, and, if the three edges incident on v are 1, 2, and 3 of weights $w_1 \leq w_2 \leq w_3$, we remove edge 1 and combine edges 2 and 3. (The two edges at the other end of 1 also get combined.) Notice that the combination of weights is as in Lemma 2.2.

If $G \neq K_4$ has triangles, we can replace a triangle with a supervertex as in Lemma 2.4. We must decide what to do with the weights inside the triangle. For a triangle T with incident edges 1, 2, and 3, we can label the edges of T with $1'$, $2'$, and $3'$, where, if 1 was incident on a vertex of T, then label $1'$ goes to the edge joining the other two vertices of T, and similarly for 2 and 3 (see Figure 2.1). The weights of $1'$, $2'$, and $3'$ are then added to 1, 2, and 3, respectively. Notice that, if 1 is later removed by an application of Lemma 2.3, then also removing $1'$ gives a path 2, $3'$, $2'$, and 3, thus justifying the way the weights are handled. This works recursively for supervertices consisting of triangles whose vertices are themselves supervertices consisting of triangles, and so on.

Finally, if $G = K_4$, then label disjoint pairs of edges $1, 1'$, $2, 2'$, and $3, 3'$. Then removing $1, 1'$ gives a cycle $2, 3', 2', 3$, so we can remove the pair of disjoint edges of least weight and apply Lemma 2.2. This completes the proof.

3. Cycle through three edges. In this section, we establish the following result.

PROPOSITION 3.1. *Let G be a 3-connected cubic graph, and let e, f, g be three edges in it. Then G has a cycle through e, f, g if and only if removing e, f, g does not*

disconnect G. Such a cycle can be found in polynomial time.

If removing e, f, g disconnects G into two components K and K', then the cycle would have to go between K and K' an even number of times and so could not include all three edges. This proves the "only if" part.

The proof of the "if" part is based on reducing the graph to a K_4 by repeatedly applying the H-transform and the T-transform. If e, f, g are three edges in K_4 that do not disconnect it, then they are not all three incident to the same vertex, so they either form a triangle or a path of length 3 that can be completed to a cycle of length 4. This produces the desired cycle.

We observe that we can find a cycle through two given edges l, m in G by selecting an endpoint u of l and an endpoint v of m, finding three disjoint paths from u to v, and observing that some two of these three paths form the desired cycle.

We denote by G^* the graph obtained from G by removing e, f, g. By assumption, G^* is connected.

Suppose first that G has no triangles. Then we can find a vertex v such that applying the H-transform at any of the three edges h_1, h_2, h_3 incident on v will result in a new graph G' which is also a 3-connected cubic graph. There are then three cases.

Case 1. None of e, f, g is incident on v. If removing h_1 from G^* does not disconnect G^*, then applying the H-transform at h_1 will result in a graph G' such that the three edges e, f, g, or edges obtained from them by contracting paths of length 2, do not disconnect it. Otherwise, removing h_1 from G^* disconnects it, creating, in particular, a component C_1 not containing v. Similarly, removing h_2 or h_3 from G^* creates components C_2, C_3 not containing v, respectively. Since G is 3-connected, the three edges e, f, g must go from C_1 to C_2, from C_1 to C_3, and from C_2 to C_3, respectively. However, then h_1, e, f separate G, and we could not have applied the H-transform at h_1 unless C_1 consists of a single vertex. In this last case, e, f get collapsed to a single edge l so that we just need to find a cycle through l, g in G', and this can be done as observed above.

Case 2. Precisely one of e, f, g is incident on v. Say $g = h_3$. As before, if removing h_1 from G^* does not disconnect G^*, then we can apply the H-transform at h_1. Otherwise, removing h_1 or h_2 from G^* creates components C_1, C_2 not containing v, respectively. The edge $g = h_3$ goes from v to either C_1 or C_2, say, C_2. The edges e, f must go from C_1 to C_2 since G is 3-connected. However, then h_1, e, f separate G, and, as before, we would not have applied the H-transform at h_1 unless C_1 consists of a single vertex, in which case e, f get collapsed to a single edge l, and we can find a cycle through l, g in G'.

Case 3. Precisely two of e, f, g are incident on v. Say $f = h_2$ and $g = h_3$. Then the H-transform at h_1 collapses f, g to a single edge l, and we can find a cycle through l, e in G'.

It is not possible that all three of e, f, g are incident on v since they would disconnect G. This completes the study of the H-transform. We consider next the T-transform, with four cases.

Case 1. None of e, f, g are inside the triangle T. Then after replacing T with a supervertex t to obtain G', the three edges e, f, g will still not disconnect G'.

Case 2. Precisely one of e, f, g is inside the triangle T. Say e is in T. Then, after replacing T with a supervertex t, it is sufficient to find a cycle in G' through f, g, and t; we can always ensure that e gets visited inside T. If either f or g is incident on t, then we just need to find a cycle through f, g, and this is done as observed before.

Otherwise, to ensure that t is visited, we can let e' be one of the three edges $1, 2, 3$ incident on T and find a cycle through e', f, g in G', provided that these three edges do not disconnect G'. If they do, then each of the edges $1, 2, 3$ disconnects G^* (even if e is included in G^*), creating components C_1, C_2, C_3 not containing T, respectively. Then adding two edges f, g to G^* to obtain G will not make it 3-connected, which is a contradiction.

Case 3. Precisely two of e, f, g are inside the triangle T. Say e, f are the edges $1', 2'$ in T. Let e', f' be the two corresponding edges $1, 2$ incident to T. Clearly, e', f', g disconnect G if and only if e, f, g disconnect G, so we can assume they do not. A cycle entering and leaving T at e', f' can be made to go through e, f inside T; we can thus carry out the T-transform.

Case 4. The three edges e, f, g constitute the triangle T. Then T is the desired cycle.

This completes the proof.

4. 3-cyclable graphs. A graph G is said to be *k-cyclable* if, for any choice of k vertices, there is a cycle in G containing all k of them. Clearly, if a graph has a Hamiltonian cycle, it is k-cyclable for all k. If k is a constant, then, given k vertices, one can find in polynomial time a cycle containing all k of them if one exists. This requires finding k disjoint paths connecting the vertices. A polynomial-time algorithm for the disjoint paths problem with a constant number of paths was given by Robertson and Seymour [12, 13] in their work on graph minors. For 3-cyclable graphs with vertices of degree at most 3, however, finding a cycle through three vertices will require only the result in section 3. Note that, in particular, 3-connected cubic graphs are 3-cyclable.

In the rest of this section, we will establish the following result (and its corollary).

THEOREM 4.1. *Let G be a 3-cyclable graph with vertices of degree at most 3 and with m edges. Then we can find, in polynomial time, a cycle in G of length at least $m^{\frac{1}{2a}}$, where $a = \log_2 3$.*

Proof. We will need three properties of the 3-cyclable graph G.

1. G is 2-connected since it is 2-cyclable.
2. If there is an edge $f = uv$ such that the subgraph $G - \{u, v\}$ on the vertices other than u, v has two or more connected components, then we can remove the edge f from G and still have a 3-cyclable graph. For, if a cycle uses f, it could use instead a path from u to v in the connected component of $G - \{u, v\}$ not visited by the original cycle. (The justification for counting edges so removed will be given in the following corollary.)
3. For any two vertices u, v, the graph $G - \{u, v\}$ on the remaining vertices has at most two connected components. For, if it has at least three components, then three vertices in the three components cannot be on a cycle, contradicting 3-cyclability.

Suppose there are two edges e_1, e_2 that separate G into two components K and K'. If K has no edge separating the two vertices incident on e_1, e_2, then we say that K is a 2-*component*. The *graph* of a 2-component K is obtained from K by connecting the two vertices incident on e_1, e_2 by an edge e and replacing all maximal 2-components contained in it by single supervertices. The graph of a 2-component is a 3-connected graph with edges replaced by paths. This follows from the fact that, if we replace all maximal paths whose vertices other than the starting and ending vertices have degree 2 by single edges and add the edge e, then there will be no parallel edges, by properties (1), (2), and (3). The resulting graph must be 3-connected since, otherwise, we could

identify a new 2-component contained in it, contrary to maximality. At the top level of G, if G is 3-connected, apply Theorem 2.1. Otherwise, select a 2-component K, adjacent via two edges e_1, e_2 to its complement K', which has some $r \geq 0$ edges, each separating the two vertices incident on e_1, e_2, thus decomposing K' into $(r+1)$ 2-components; so G is a cycle of $(r+2)$ 2-components.

So suppose G is not 3-connected, and perform the above decomposition into 2-components. The 2-components so selected have a tree structure: at the top level, we have a cycle of 2-components, which are viewed as the children of the root; for a 2-component in this tree, its children are the maximal 2-components contained in it.

We assign a weight w_i to each 2-component in the tree so that, if the 2-component has m_i edges, then $w_i \geq {m_i}^d$, where $d = \frac{1}{a} = \log_3 2$. We do this inductively, starting at the leaves. So suppose, for a given 2-component, we have assigned weights to the maximal 2-components contained in it. The graph of the 2-component is a 3-connected graph with edges replaced by paths. To every such path, assign a weight equal to the sum of the weights of the supervertices in it plus the number of edges connecting these supervertices. The special edge e is given weight 0. Apply then Theorem 2.1 to the 3-connected graph, obtaining a cycle of weight $w \geq (\sum w_i^a)^{\frac{1}{a}} \geq (\sum m_i)^{\frac{1}{a}} = m^d$. Assign weight w to the 2-component, completing the induction. At the top level, we have a cycle of 2-components, so we add their weights plus the number of edges in the cycle.

Unfortunately, the cycles that we selected for each 2-component to define the weights of the 2-component do not necessarily connect since the cycle for a 2-component may not go through the special edge e that links it to its parent in the tree. We shall define cycles that do go through e so that, if a 2-component has weight w, then the cycle through e visits at least $w^{\frac{1}{2}} \geq m^{\frac{1}{2a}}$ edges other than e, and this will complete the proof.

Consider a 2-component whose weight w was defined by finding a cycle C of weights w_i, and suppose, for each i, we have found a path inside the corresponding maximal 2-component of length at least ${w_i}^{\frac{1}{2}}$. We write $w_i = \epsilon_i w$, with $\sum \epsilon_i = 1$. Clearly, if C goes through e, we can just select C and have a cycle through e of length at least $\sum {w_i}^{\frac{1}{2}} \geq w^{\frac{1}{2}}$ since $\sum {\epsilon_i}^{\frac{1}{2}} \geq 1$, where the sums are over the weights in C.

Suppose C does not go through e. By 3-connectivity, we can join an endpoint of e to C with three disjoint paths and, in particular, obtain two disjoint paths joining the two endpoints of e to C, respectively. The two disjoint paths enter C at two distinct vertices, thus decomposing the weights of C into two subsets S and T delimited by these two vertices. If $\sum {\epsilon_i}^{\frac{1}{2}} \geq 1$ when the sum is taken over the weights in either S or T, then we are done. If this is not the case, then we will show that the largest weight $w_1 = \epsilon_1 w$ in S and the largest weight $w_2 = \epsilon_2 w$ in T satisfy ${\epsilon_1}^{\frac{1}{2}} + {\epsilon_2}^{\frac{1}{2}} > 1$. If $w_2 = 1$, then the cycle through e and S goes through the edge of weight w_1 and at least one other edge (of weight at least $w_2 = 1$). So we can assume $w_1, w_2 > 1$. It will then be sufficient to make the cycle through e go through the edges corresponding to the two weights w_1, w_2. Each of these two edges is a path with a maximal 2-component in it; then we can select two representative vertices from the two maximal 2-components, plus a vertex from the complement of the 2-component to represent edge e; a cycle through these three vertices must exist by 3-cyclability; this implies that there is a cycle through the three selected edges in the graph of the 2-component, and since this graph is a 3-connected cubic graph, such a cycle can be found in polynomial time by Proposition 3.1.

It remains to prove the claim about the sum of the $\epsilon_i^{\frac{1}{2}}$. Notice that, if $w_i \geq w_{i'}$

and $0 \le \delta \le w_{i'}$, then ${w_i}^{\frac{1}{2}} + {w_{i'}}^{\frac{1}{2}} \ge (w_i + \delta)^{\frac{1}{2}} + (w_{i'} - \delta)^{\frac{1}{2}}$. For the argument, we modify the weights by choosing the appropriate values δ as follows. We can ensure that S will have at most one nonzero weight smaller than w_1 and, similarly, that T will have at most one nonzero weight smaller than w_2. Thus S has $s \ge 1$ weights equal to w_1 and one weight $0 \le w_1' < w_1$; similarly, T has $t \ge 1$ weights equal to w_2 and one weight $0 \le w_2' < w_2$.

We thus have $s\epsilon_1 + \epsilon_1' + t\epsilon_2 + \epsilon_2' = 1$, with $s\epsilon_1^{\frac{1}{2}} + {\epsilon_1'}^{\frac{1}{2}} < 1$ and $t\epsilon_2^{\frac{1}{2}} + {\epsilon_2'}^{\frac{1}{2}} < 1$. We write $\epsilon_1' = \lambda\epsilon_1$ with $0 \le \lambda < 1$ and let $s' = s + \lambda$. Similarly, we write $\epsilon_2' = \mu\epsilon_2$ with $0 \le \mu < 1$ and let $t' = t + \mu$. Then $s'\epsilon_1 + t'\epsilon_2 = 1$. Also, $s'\epsilon_1^{\frac{1}{2}} \le (s + \lambda^{\frac{1}{2}})\epsilon_1^{\frac{1}{2}} < 1$, and similarly $t'\epsilon_2^{\frac{1}{2}} < 1$. However, then $s'\epsilon_1 < \epsilon_1^{\frac{1}{2}}$ and $t'\epsilon_2 < \epsilon_2^{\frac{1}{2}}$, so $\epsilon_1^{\frac{1}{2}} + \epsilon_2^{\frac{1}{2}} > 1$. This completes the proof. □

COROLLARY 4.2. *Let G be a graph with vertices of degree at most 3. Let l be the maximum number of edges in a 3-cyclable minor of G with degrees at most 3. (So l is at least the length of the longest cycle in G.) Then we can find, in polynomial time, a cycle of length at least $l^{(\log_3 2)/2}$.*

Proof. Note that a 3-cyclable minor K with degrees at most 3 is obtained from a subgraph H of G by skipping some vertices of degree 2 in H so that an edge in K corresponds to a path with internal vertices of degree 2 in H.

We can assume G is 2-connected. Otherwise, some edge f separates G into two components, and the 3-cyclable minor must be contained in one of these two components. We may thus remove f; the problem decomposes into 2-connected subproblems.

The structure of the graph as a tree of 2-components is still the same as before, even if the graph is not 3-cyclable. The only difference is that we cannot avoid parallel edges in the graph of a 2-component. Parallel edges occur only in one new kind of 2-component: Let P_3 be the graph consisting of three parallel edges joining two vertices; the graph of a 2-component may now also be P_3, with one of the three parallel edges being the special edge e linking to the parent in the tree, and one or both of the other two parallel edges f_1, f_2 being a path of vertices corresponding to 2-components that are children of this 2-component in the tree.

We proceed to simplify the graph as follows, from the leaves of the tree up to the root. For the special P_3 2-component, we remove one of the two parallel edges f_1, f_2—the one corresponding to a subgraph of smaller size. Thus we have no parallel edges anymore, as before.

As we go up the tree, we must also make sure that we will be able to obtain, from a cycle in a 2-component, a new cycle containing the special edge e. This is not possible if the two edges f_1 and f_2 with weights w_1 and w_2, together with e, separate the 3-connected graph of the 2-component. All such possible pairs (f_1, f_2) which together with e separate the 3-connected graph must be disjoint. We replace one of f_1, f_2, the one corresponding to a subgraph of smaller size, with a single edge (of weight 1).

If the 3-cyclable minor of size l reaches the root of the tree of 2-components, then the simplified graph will have $m \ge l$ edges, by induction from the leaves up to the root of the tree of 2-components. The reason is that, when f_1 or f_2 is removed or reduced to a single edge, the 3-cyclable minor of size l necessarily does have vertices in one of the two subgraphs corresponding to f_1 and f_2. We then get a cycle of length at least $m^{(\log_3 2)/2}$ with the previous algorithm.

It may seem that one should not count the edges removed to satisfy property 2 above. However, when a single edge f_1 is removed from P_3, at least one edge

corresponding to e is later added, and one can count both edges since $x^{\frac{1}{2a}} + 1 \geq (x+2)^{\frac{1}{2a}}$ for $a, x \geq 1$.

If the 3-cyclable minor of size l does not reach the root of the tree of 2-components, consider the 2-component closest to the root that it reaches. The 3-cyclable minor does not have vertices corresponding to the special edge e connecting this 2-component to its parent. On this 2-component, we do not simplify the graph for pairs of edges f_1, f_2 which together with e disconnect the graph of the 2-component (neither in the case of P_3 nor in the case of a 3-connected graph); we assign to e weight 1. As before, we have $m \geq l$ edges by induction from the leaves of the tree up to this 2-component and get a cycle of length at least $m^{(\log_3 2)/2}$ with the previous algorithm. □

An algorithm that approximates the longest cycle problem can be used to approximate the longest path problem by guessing and adding the edge that would complete the long path into a cycle and then removing this edge after a long cycle is found and getting back a path. In our case, since adding an edge could create two vertices of degree four, we can instead add the edge between two new vertices inserted in the middle of two existing edges.

5. Long cycles and toughness. A graph G is said to be 1-*tough* if it has the following property: *For every set S of vertices, the graph $G - S$ on the remaining vertices has at most $|S|$ components.* There has been interest in devising algorithms for long paths/cycles in graphs that are 1-tough [5, 10, 14] because Hamiltonian and weakly Hamiltonian graphs are 1-tough. There has been some work that shows that, for graphs with large degree sums, 1-toughness implies the existence of long cycles [2]. In the above results, 3-connected cubic graphs are 1-tough since every component is delimited by at least three separating vertices, while each separating vertex delimits at most three components. The graphs in Theorem 4.1 are not necessarily 1-tough. However, all 2-connected graphs with vertices of degree at most 3 can be made 1-tough without significantly altering the structure of the graph, namely, by replacing each vertex of degree 3 with a triangle; then each vertex of S delimits at most two components of $G - S$, and each component of $G - S$ is delimited by at least two vertices of S, giving 1-toughness.

We show that there are 1-tough cubic graphs that are only 2-connected, and where the length of the longest cycle is only logarithmic in the number of vertices, thus giving an exponential gap with respect to the 3-connected case for cubic graphs.

THEOREM 5.1. *There are arbitrarily large 1-tough, 2-connected, n-vertex k-regular graphs for $k \geq 3$ whose longest cycle has length $O(k \log_k n)$ and whose longest path has length $O(k \log_k^2 n)$. The graphs for $k = 3$ are planar.*

Proof. We give an inductive construction, and, for the induction to go through, we will need a condition that is slightly stronger than 1-toughness. We say that a graph with some chosen special edges is *extra 1-tough* if, when counting the number of components of $G - S$, we also count as components any special edges both of whose endpoints are in S; the total is still at most $|S|$. Notice that K_{k+1}, with all edges incident to one chosen vertex v as special, is extra 1-tough.

Suppose we have constructed a 2-connected k-regular graph G with some special edges, starting with K_{k+1} as just described, and this graph is extra 1-tough. We select a special edge e and replace it with $H = K_{k+1}$ as follows. Break one of the special edges (v, w) in H, and replace e with H by connecting v and w to the vertices to which e is incident. The resulting graph is still 2-connected, k-regular, and extra 1-tough. To see this last property, notice that the t vertices in S not in H still give at most t components, while the $|S| - t$ vertices in H subdivide the component corresponding

to e as follows. The vertices v and w may each create at most one extra component, while the remaining vertices in H create no component except for the special edge (v, u) for each u chosen if v is also chosen. This proves extra 1-toughness.

We apply this construction inductively. Starting with K_{k+1}, we replace the k special edges with H, where each copy of H brings in $k-1$ special edges. We then replace the $k(k-1)$ special edges with a copy of H again, so we now have $k(k-1)^2$ special edges, and we continue in this fashion. The resulting graph will have the structure of two balanced trees of degree k connected at the leaves, where one tree corresponds to the subdivision into special edges at v, while the other tree brings the special edges back together into the remaining vertices of $H-v$. The trees have height $O(\log_k n)$, and the longest cycle will go from one root to the other and back, visiting the k vertices in $H-v$ for each vertex in the second tree. The cycle will thus have length at most $O(k \log_k n)$. The longest path has a middle section that goes from one root v_1 to the other root v_2, then back to a child of v_1, then back to a child of a child of v_2, and so on. This gives the $O(k \log_k^2 n)$ bound for paths. □

It remains open whether 3-connected graphs always have cycles or paths of length n^c for some $c < 1$ by also assuming the graph to be planar or of bounded degree. It was shown by Karger, Motwani, and Ramkumar [10] that there are 3-connected graphs that are 1-tough, have a 2-factor, and are 3-cyclable, with longest path of length $O(\log n)$; the examples, however, have two vertices adjacent to all vertices.

THEOREM 5.2. *Let G be a 3-connected graph with maximum degree at most 6 such that every vertex of degree 4 belongs to at least one triangle, every vertex of degree 5 belongs to at least two edge-disjoint triangles, and every vertex of degree 6 belongs to at least three edge-disjoint triangles. Then G is 1-tough. Let T be a minimal set of triangles satisfying the conditions on vertices of degree 4, 5, and 6. If no two vertices of degree 5 or 6 are adjacent, then no triangle in T shares two different edges with two other triangles of T. If this condition on the triangles of T holds, then we can find in polynomial time a cycle of length $r^{\log_3 2}$, where r is the number of edges of G minus 2/3 the number of edges belonging to triangles of T.*

Proof. The triangles that are assumed to exist for vertices of degree 4, 5, and 6 imply that the neighborhood of every vertex contains at most three independent vertices. As a consequence, if we select a set S of vertices, each vertex of S will be adjacent to at most three components of $G-S$. On the other hand, each component of $G-S$ is delimited by at least three vertices of S by 3-connectivity. It follows that $G-S$ has at most $|S|$ components; that is, G is 1-tough.

Suppose now that the vertices of degree 5 or 6 are independent; thus their neighbors are of degree 3 or 4. Since degree 4 vertices require only one triangle, it follows that a vertex of degree 6 will not have any triangle in T other than the three edge-disjoint triangles required to be in T. If a triangle t_1 in T shares two distinct edges with two triangles t_2 and t_3 in T, then some vertex v of t_1 must be of degree 5; say, it is a vertex v common to t_1 and t_2. If the vertex of t_2 not on t_1 and the vertex of t_3 not on t_1 are the same, then we can drop t_1 or t_2 from T. Otherwise, either v also belongs to t_3, in which case t_2 and t_3 can be used as the two edge-disjoint triangles for v and we can drop t_1, or v does not belong to t_3. In this last case, either the triangle t_4 which is edge-disjoint from t_1 is also edge-disjoint from t_2, in which case we can drop t_1, or it shares an edge with t_2, in which case we can drop t_2.

So we assume that every triangle of T shares at most one edge with other triangles of T. We first consider the case where all triangles of T are edge-disjoint. It is known that the operation of replacing a vertex u of degree at least 4 with two adjacent vertices

v and w each having at least two neighbors, the neighborhoods of v and w being disjoint and giving the neighborhood of u when combined, preserves 3-connectivity. We give weight $1/2$ to the edges of T and weight 1 to the remaining edges. We replace each vertex of degree 4 with two vertices as just explained, with one of the two vertices incident to a triangle in T, and give weight 0 to the edge joining the two new vertices. For degree 5 vertices, we replace them with a path of length 2 of weight 0 edges, and distribute the adjacencies to make the new vertices have degree 3, in such a way that the first and last vertex of the weight 0 path are incident to two edge-disjoint triangles in T. For degree 6 vertices, we replace them with a star of degree 3 consisting of three weight 0 edges, so that the three leaves are incident to three edge-disjoint triangles in T. By Theorem 2.1, we can find a cycle of weight at least $r^{\log_3 2}$ in the resulting cubic graph. This cycle, however, might visit some vertex of the original graph more than once. However, then it must visit two edges of the same selected triangle in T, and these two edges of weight $1/2$ can be replaced with a single direct edge, so we avoid such conflicts and obtain a cycle of length at least $r^{\log_3 2}$ in the original graph.

Notice that a vertex of degree 6 belongs only to three edge-disjoint triangles in T, while a vertex of degree 5 can have at most two triangles sharing an edge in T. So only two or three triangles can share an edge e. When this happens, we cannot preserve all two or three triangles when introducing weight 0 edges, but we can preserve one of them, say, t, and introduce single weight 0 edges at the endpoints of e separating t from the other one or two triangles that used to share e. Notice that, if there are two separated triangles, they cannot both give a conflict at their vertices not on e, so we can restrict our attention to the case of only one separated triangle. We thus have a gadget with five edges of weight $1/2$, at most two edges of weight 0, and at most four incoming edges. This gadget is traversed by at most one or two paths. We can verify that these paths can be modified within the gadget so as to avoid conflicts and in such a way that the resulting path lengths are at least the sum of the original path weights. This completes the proof. □

A graph is said to be α-tough if, for every set S of vertices that leaves a graph $G - S$ on the remaining vertices with $k \geq 2$ connected components, we have $|S| \geq \alpha k$. It has been conjectured that there is a constant α such that all α-tough graphs have a Hamiltonian cycle. It is known that it is not enough to take $\alpha = 1.5$: Consider a 3-connected cubic graph that does not have a Hamiltonian cycle, and replace every vertex with a triangle; since every vertex of the resulting graph has at most two independent neighbors, it follows that every vertex of S is adjacent to at most two components of $G - S$, while every component of $G - S$ is delimited by at least three vertices of S by 3-connectivity, giving 1.5-toughness. In this direction, we would like to know if $(1 + \epsilon)$-tough graphs with $\epsilon > 0$, which are, in particular, 3-connected, always have cycles of length at least n^c for some $c = c(\epsilon) < 1$.

Acknowledgment. We are grateful to Sudipto Guha for several helpful discussions.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–233.

[2] D. Bauer, A. Morgana, E. F. Scheichel, and H. J. Veldman, *Long cycles in graphs with large degree sums*, Discrete Math., 79 (1989), pp. 59–70.

[3] C. Bazgan, M. Santha, and Z. Tuza, *On the approximation of finding a(nother) Hamiltonian cycle in cubic Hamiltonian graphs*, J. Algorithms, 31 (1999), pp. 249–268.

[4] J. A. Bondy and M. Simonovits, *Longest cycles in* 3*-connected cubic graphs*, Canad. J. Math., 32 (1980), pp. 987–992.
[5] V. Chvatal, *Hamiltonian cycles*, in The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E. L. Lawler et al., eds., Wiley, Chichester, UK, 1985, pp. 402–430.
[6] T. Feder, R. Motwani, and C. Subi, *Finding long paths and cycles in sparse Hamiltonian graphs*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 524–529.
[7] M. Fürer and B. Raghavachari, *Approximating the minimum degree spanning tree to within one from the optimal degree*, in Proceedings of the Third Annual ACM–SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 317–324.
[8] M. Fürer and B. Raghavachari, *Approximating the minimum-degree Steiner tree to within one of optimal*, J. Algorithms, 17 (1994), pp. 409–423.
[9] B. Jackson, *Longest cycles in* 3*-connected cubic graphs*, J. Combin. Theory Ser. B, 41 (1986), pp. 17–26.
[10] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.
[11] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.
[12] N. Robertson and P. D. Seymour, *An outline of a disjoint paths algorithm*, in Paths, Flows, and VLSI-Layout, N. Korte et al., eds., Springer-Verlag, Berlin, 1990, pp. 267–292.
[13] N. Robertson and P. D. Seymour, *Graph minors* XIII. *The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.
[14] S. Vishwanathan, *An approximation algorithm for finding a long path in Hamiltonian graphs*, in Proceedings of the Eleventh Annual ACM–SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 680–685.

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1608–1623

IMPROVED APPROXIMATION ALGORITHMS FOR THE VERTEX COVER PROBLEM IN GRAPHS AND HYPERGRAPHS*

ERAN HALPERIN†

Abstract. We obtain improved algorithms for finding small vertex covers in bounded degree graphs and hypergraphs. We use semidefinite programming to relax the problems and introduce *new* rounding techniques for these relaxations. On graphs with maximum degree at most Δ, the algorithm achieves a performance ratio of $2-(1-o(1))\frac{2\ln\ln\Delta}{\ln\Delta}$ for large Δ, which improves the previously known ratio of $2-\frac{\log\Delta+O(1)}{\Delta}$ obtained by Halldórsson and Radhakrishnan. Using similar techniques, we also present improved approximations for the vertex cover problem in hypergraphs. For k-uniform hypergraphs with n vertices, we achieve a ratio of $k-(1-o(1))\frac{k\ln\ln n}{\ln n}$ for large n, and for k-uniform hypergraphs with maximum degree at most Δ the algorithm achieves a ratio of $k-(1-o(1))\frac{k(k-1)\ln\ln\Delta}{\ln\Delta}$ for large Δ. These results considerably improve the previous best ratio of $k(1-c/\Delta^{\frac{1}{k-1}})$ for bounded degree k-uniform hypergraphs, and $k(1-c/n^{\frac{k-1}{k}})$ for general k-uniform hypergraphs, both obtained by Krivelevich. Using similar techniques, we also obtain an approximation algorithm for the weighted *independent set* problem, matching a recent result of Halldórsson.

Key words. vertex cover, semidefinite programming, approximation algorithms

AMS subject classification. 68W25

PII. S0097539700381097

1. Introduction.

Given a hypergraph $H=(V,E)$, a *vertex cover* of H is a set $C\subset V$ such that, for every $e\in E$, there exists $v\in e\cap C$. The *vertex cover* problem is the problem of finding a vertex cover of minimum size. The weighted version of the problem is obtained when every $v\in V$ has a nonnegative weight $w(v)$ and the goal is to find a vertex cover of H of minimum total weight.

The *vertex cover* problem in hypergraphs is equivalent to the *set cover* problem. An instance of the *set cover* problem is a set S and a collection C of subsets of S. A cover of S is a subset $X\subseteq C$ such that every $s\in S$ is contained in one of the sets of X. The goal is to find a cover of minimal size. It is clear that this is another formulation for the *vertex cover* problem in a hypergraph, by identifying every edge with an element of S, and every vertex with the subset that contains all the edges containing it.

The greedy algorithm gives a $(\ln n+1)$-approximation ratio for the *set cover* problem [5], where $|V|=n$. Moreover, the *set cover* problem cannot be approximated within a factor of less than $\ln n$ unless $NP\subseteq DTIME[n^{\log\log n}]$ [6].

As the approximation ratio for the *vertex cover* problem in general hypergraphs cannot be improved, we consider some of its subproblems. We first consider the *vertex cover* problem in bounded degree graphs. The *vertex cover* problem in graphs is known to be APX-complete [22], and furthermore, it cannot be approximated within a factor of $\frac{7}{6}\approx 1.1666$ [15]. A 2-approximation algorithm for the *vertex cover* problem

*Received by the editors November 14, 2000; accepted for publication (in revised form) February 23, 2002; published electronically August 15, 2002. This work is part of the author's Ph.D. thesis, prepared at Tel-Aviv University under the supervision of Professor Uri Zwick. A preliminary version of the paper appeared in *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms,* San Francisco, CA, 2000.

http://www.siam.org/journals/sicomp/31-5/38109.html

†Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (heran@post.tau.ac.il).

in a graph is trivial, one has to find a *maximal* matching in the graph and take all its vertices as a cover. The best algorithms known for the problem were found by Bar-Yehuda and Even [2] and by Monien and Speckenmeyer [21]. These algorithms achieve a ratio of $(2 - \frac{\ln\ln n}{2\ln n})$, where n is the number of vertices in the graph.

In graphs with maximal degree Δ, Hochbaum [17] shows how to approximate the *vertex cover* problem within a factor of $(2 - \frac{2}{\Delta})$. Halldórsson and Radhakrishnan [13] improve this and give a $(2 - \frac{\log\Delta + O(1)}{\Delta})$-approximation algorithm for the problem. We improve this result and give a $(2 - (1 - o(1))\frac{2\ln\ln\Delta}{\ln\Delta})$-approximation algorithm using semidefinite programming relaxations and new rounding techniques for the relaxed solution. Notice that $\frac{\ln\ln\Delta}{\ln\Delta} \gg \frac{\ln\Delta}{\Delta}$, and thus our algorithm substantially improves the known algorithms.

We next consider the vertex cover problem in hypergraphs with edges of size at most k for a given k. It is equivalent to the set cover problem restricted to instances in which every element is found in at most k sets. For simplicity we consider k-uniform hypergraphs, which are hypergraphs in which all edges are of size k. All our results can be easily extended to the case where each edge is of size at most k. A k-approximation algorithm is straightforward using a maximal matching. For any fixed k, the best algorithm known [20] achieves a ratio of $k(1 - c/n^{\frac{k-1}{k}})$. For bounded degree k-uniform hypergraphs, the best algorithm known [20] achieves a ratio of $k(1 - c/\Delta^{\frac{1}{k-1}})$. In this paper we present an algorithm which, for a fixed k, achieves a ratio of $(k - (1 - o(1))k(k-1)\frac{\ln\ln\Delta}{\ln\Delta})$ for k-uniform hypergraphs, where Δ is the maximal degree in the hypergraph. This result can be extended to the case where k is allowed to grow slowly with Δ. For general k-uniform hypergraphs with n vertices, our algorithm achieves a ratio of $(k - (1 - o(1))\frac{k\ln\ln n}{\ln n})$. These results apply to the set cover problems in which every element is contained in at most k sets. The methods used in [2, 21] for graphs cannot be extended easily to hypergraphs, and thus our techniques establish a uniform approach for the vertex cover problem in graphs and in hypergraphs.

Finally, we show that our techniques can be easily used in other problems, and we consider as an example the *independent set* problem in bounded degree graphs. Given a graph $G = (V, E)$, an independent set $I \subseteq V$ is a set of vertices satisfying that no two vertices in I are adjacent. The *independent set* problem is the problem of finding a maximum size independent set in G. If the vertices of G are weighted, then the weighted version of the problem is finding a maximum weight independent set in G. This problem was studied extensively by many researchers [17, 13, 4, 3, 11, 12, 25, 10], and it is known to be hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, unless NP-hard problems have randomized polynomial time algorithms [14]. For bounded degree graphs, Halldórsson and Radhakrishnan [12] found the first nontrivial approximation algorithm which achieves a ratio of $O(\Delta/\ln\ln\Delta)$, where Δ is the maximal degree in the graph. Vishwanathan [25] found a polynomial $O(\Delta\ln\ln\Delta/\ln\Delta)$-approximation algorithm for the unweighted case. This algorithm uses an algorithm of Alon and Kahale [1] as a black box, which by itself uses an algorithm of Karger, Motwani, and Sudan [18] as a black box. We present an algorithm which uses a more direct approach. Furthermore, our result extends to the weighted case. Recently, a similar result for the weighted case was obtained independently by Halldórsson [10].

To sum up, the following are the main improvements achieved in this paper:

- A polynomial time randomized algorithm for the weighted *vertex cover* problem in graphs which achieves a ratio of $2-(1-o(1))\frac{2\ln\ln\Delta}{\ln\Delta}$ for large Δ, where

Δ is the maximal degree in the graph (see Theorem 2.1). This substantially improves the previous ratio of $2 - \frac{\log \Delta}{\Delta}$ [13]. In particular, for large n, this gives a ratio of $2-(1-o(1))\frac{2\ln\ln n}{\ln n}$ for general graphs, where n is the number of vertices in the graph, which slightly improves the known $2 - \frac{\ln\ln n}{2\ln n}$ ratio [2, 21].

- For every fixed k, a polynomial time randomized algorithm for the weighted *vertex cover* problem in k-uniform hypergraphs which achieves a ratio of $k - (1 - o(1))\frac{k(k-1)\ln\ln\Delta}{\ln\Delta}$, for large Δ, where Δ is the maximal degree in the hypergraph (see Theorem 4.1). In general k-uniform hypergraphs, the algorithm achieves a ratio of $k - (1 - o(1))\frac{k\ln\ln n}{\ln n}$, where n is the number of vertices in the hypergraph and n is large. (Note that the maximum degree in this case is at most n^{k-1}.) These results substantially improve the former ratios of $k(1 - c/\Delta^{\frac{1}{k-1}})$ and $k(1 - c/n^{\frac{k-1}{k}})$ [20].
- In section 5 we introduce a randomized $\Omega(\frac{\log\Delta}{\Delta\log\log\Delta})$-approximation algorithm for the independent set problem in weighted graphs, matching a recent bound of Halldórsson [10].

The rest of the paper is organized as follows. In section 2 we describe an approximation algorithm for the weighted *vertex cover* problem in bounded degree graphs, p-claw free graphs, and sparse graphs. In section 4 we describe an approximation algorithm for the weighted *vertex cover* problem in bounded degree hypergraphs, and in section 5 we describe an approximation algorithm for the weighted *independent set* problem in bounded degree graphs.

2. Vertex cover in graphs. We first need some notations and definitions. Let $G = (V, E)$ be a weighted graph with n vertices and m edges. Assume $V = \{1, \ldots, n\}$ and that for every $i \in V$ we have a positive weight w_i. For a set $S \subseteq V$, let $e(S)$ be the set of edges in the subgraph induced by S. Let $w(S)$ be the total weight of S, i.e., $w(S) = \sum_{i\in S} w_i$. Let Δ denote the maximal degree in G. A p-claw in G is an induced subgraph of G isomorphic to $K_{1,p}$. G is a p-claw free graph if G does not contain a p-claw. Let $vc(G)$ denote the size of the minimal vertex cover in G.

2.1. Relaxations of the vertex cover problem. The minimum vertex cover can be formulated using the following integer linear program:

$$\begin{array}{lll} \text{Minimize} & \sum w_i \cdot x_i, & \\ & x_i + x_j \geq 1 \quad , & (i,j) \in E, \\ & x_i \in \{0,1\} \quad , & i \in V. \end{array} \tag{1}$$

As solving integer programs is known to be NP-hard, this formulation of the problem cannot help us directly in finding a good approximation algorithm. On the other hand, a 2-approximation algorithm is obtained by using the following relaxation to the integer program (1):

$$\begin{array}{lll} \text{Minimize} & \sum w_i \cdot x_i, & \\ & x_i + x_j \geq 1 \quad , & (i,j) \in E, \\ & 0 \leq x_i \leq 1 \quad , & i \in V. \end{array} \tag{2}$$

Let us denote the optimum of relaxation (2) by $lp(G)$. It is clear that $lp(G) \leq vc(G)$, and it is easy to see that the set $S = \{i : x_i \geq \frac{1}{2}\}$ is a cover for G and that its size is no more than $2 \cdot lp(G)$. Solving relaxation (2) can be done in polynomial time

using the ellipsoid method [8, 23], and therefore solving relaxation (2) and taking S as a cover is a polynomial 2-approximation algorithm for the problem [16].

For the complete graph of n vertices with all weights equal to 1, the integrality ratio of relaxation (2) is $(2-\frac{2}{n})$, i.e., the ratio $vc(K_n)/lp(K_n)$ is at least $(2-\frac{2}{n})$, and this ratio is attained when all variables are equal to $\frac{1}{2}$. In order to eliminate this case, we define the following semidefinite relaxation:

$$\text{(3)} \qquad \begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} w_i \cdot \frac{1+v_0 \cdot v_i}{2}, \\ & v_i \in \mathbb{R}^n, \|v_i\| = 1 \quad , \quad i \in V, \\ & (v_j - v_0) \cdot (v_i - v_0) = 0 \quad , \quad (i,j) \in E. \end{aligned}$$

Let us denote the optimum of relaxation (3) by $sd(G)$. In [19] it is proved that this is indeed a relaxation to the vertex cover problem. Moreover, they show that the integrality gap of relaxation (3) is at least $2-\epsilon$ for every constant $\epsilon > 0$; i.e., for every $\epsilon > 0$ there is a graph G_ϵ such that $vc(G_\epsilon)/sd(G_\epsilon)$ is at least $2-\epsilon$. They also show that $sd(G) \leq lp(G)$ for every G, and therefore relaxation (3) is tighter.

By assigning every dot product $v_i \cdot v_j$ a variable y_{ij}, relaxation (3) can be formulated as minimizing a linear function of the variables y_{ij}, subject to linear constraints on the variables and the constraint that the matrix $Y = \{y_{ij}\}_{1 \leq i,j \leq n}$ is positive semidefinite; i.e., for every $v \in \mathbb{R}^n$, $v^t Y v \geq 0$. For every $\epsilon > 0$, this program can be solved within an additive error of ϵ in polynomial time in $\log \frac{1}{\epsilon}$ and n using the ellipsoid method [8, 23]. Using Choleski decomposition, one can find the vectors v_i from the matrix Y in polynomial time.

Using relaxation (3), one can easily attain a 2-approximation algorithm for the *vertex cover* problem by solving the relaxation and considering the set $S = \{i \in V | v_0 \cdot v_i \geq 0\}$ as the cover. It can be easily verified that this is indeed a 2-approximation algorithm, and that its worst case occurs when all vectors are orthogonal to v_0.

The intuition for our algorithm is that whenever two vectors corresponding to adjacent vertices are very close to being orthogonal to v_0, they are close to being opposite to each other and therefore a large independent set can be found among those vertices. Thus, subtracting the independent set from the set of those vertices gives a relatively small vertex cover.

2.2. The approximation algorithm. We are now ready to describe an approximation algorithm for the minimum vertex cover. The input of the algorithm is a graph $G = (V, E)$ with $|V| = n$, and $|E| = m$, and a weight function w. Let x be a small positive number to be determined later. In Figure 1, we give the flow of the algorithm. The algorithm first solves the semidefinite program (3), then partitions the vertices of the graph into three parts, finds a large independent set in one of them, and outputs a result using this partition.

We now give an intuition as to why the algorithm finds a large independent set inside S_2. I' is the set of vertices whose vectors are relatively close to r. In section 2.3 we show how to choose c such that the expectation of the sum $\sum_{(i,j) \in e(I')} (w_i + w_j)$ is less than a constant factor of the expected total weight of I'. We thus do not have many vertices in I' which are not isolated in the subgraph induced by I', and therefore, if we remove the nonisolated vertices from I', we are left with an independent set I of total weight which is a constant factor of $w(I')$. We choose c such that $w(I')$ is maximal and thus get a large independent set.

It is easy to see that the algorithm finds a vertex cover. To show this, we have to show that, for every $(i,j) \in E$, i or j are in the cover. If $i \in S_1$ or $j \in S_1$, we are

Algorithm. VC-G

Input: A weighted graph $G = (V, E)$.
Output: A vertex cover of G.

1. Solve relaxation (3).
2. Let $S_1 = \{i \mid v_0 \cdot v_i \geq x\}$, $S_2 = \{i \mid -x \leq v_0 \cdot v_i < x\}$.
3. Find a large independent set I in S_2, using the following procedure (see [18]). Let c be a positive number to be determined later.
   (a) Choose an n-dimensional random vector r, and let $I' = \{i \in S_2 \mid v_i \cdot r \geq c\}$.
   (b) Remove vertices from I' which are nonisolated in the subgraph induced by I'. The remaining vertices form an independent set I.
4. Output $S_1 \cup (S_2 \setminus I)$.

FIG. 1. *Algorithm* VC-G *for finding a small vertex cover in a graph.*

done. If $i \in V \setminus (S_1 \cup S_2)$, then $v_0 \cdot v_i < -x$ and therefore

$$v_j \cdot v_0 = 1 + v_i \cdot v_j - v_0 \cdot v_i \geq x.$$

Thus, $j \in S_1$ and is therefore in the cover. We are left with the case that $i, j \in S_2$. However, as I is independent, at least one of i and j is not in I and is therefore in the cover.

2.3. Analysis of the algorithm. The main result we prove in this section is the following theorem.

THEOREM 2.1. *Let $G = (V, E)$ be a weighted graph with maximal degree at most Δ. The randomized algorithm* VC-G *achieves a performance ratio of*

$$2 - \frac{2 \ln \ln \Delta}{\ln \Delta} + o\left(\frac{\ln \ln \Delta}{\ln \Delta}\right).$$

In order to prove Theorem 2.1, we need some known results about n-dimensional probability space. Recall that $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ is the density function of a standard normal random variable. Let $N(x)$ be the tail of the standard normal distribution, i.e., $N(x) = \int_x^\infty \phi(x)dx$. We will use the following bounds on $N(x)$ (see Lemma VII.2 of Feller [7]).

LEMMA 2.2. *For every $c \geq 1$, $\phi(c)(\frac{1}{c} - \frac{1}{c^3}) < N(c) < \frac{\phi(c)}{c}$.*

A random vector $r = (r_1, \ldots, r_n)$ is said to have the *n-dimensional standard normal distribution* if the components r_i are independent random variables, each one having the standard normal distribution. In this paper a random n-dimensional vector will always denote a vector chosen from the n-dimensional standard distribution. We will use the following (see Theorem IV.16.3 in [24]).

LEMMA 2.3. *Let u be any unit vector in $\mathbb{R}^n$. Let r be a random n-dimensional vector. The projection of r on u, given by the product $r \cdot u$, is distributed according to the standard normal distribution.*

We now prove a lemma which bounds the size of the independent set I found in algorithm VC-G. This lemma is derived from Lemma 7.1 of [18].

LEMMA 2.4. *Let $G = (V, E)$ be a weighted graph with maximal degree Δ and weight function w. Let x be a number such that $x = \Theta(\frac{\ln\ln\Delta}{\ln\Delta})$. Assume $V = \{1, \ldots, n\}$ and that for every $i \in V$ there corresponds an n-dimensional unit vector v_i such that for every $(i, j) \in E$, $v_i \cdot v_j \leq -1 + 2x$. Then there is a constant C such that, for Δ sufficiently large, an independent set of total weight at least $C \cdot \frac{w(V)}{\Delta^x \cdot \sqrt{x \cdot \ln\Delta}}$ can be found by a randomized algorithm in polynomial time.*

Proof. As explained in section 2.2, we let c be a positive number to be determined later, and we let r be a random n-dimensional vector. Let $I' = \{i \in V \mid v_i \cdot r \geq c\}$. By the linearity of the expectation,

$$E(w(I')) = \sum_{i \in V} w_i \cdot N(c) = w(V) \cdot N(c).$$

Let $X = \sum_{(i,j) \in e(I')} (w_i + w_j)$. We upper bound the expectation of X as follows. By the linearity of the expectation, the expectation of X is

$$\sum_{(i,j) \in E} (w_i + w_j) \cdot Pr(i, j \in I').$$

Thus, in order to upper bound the expectation of X, it is sufficient to bound the probability that a given edge will be in the subgraph induced by I'.

By the terms of the lemma, for every $(i, j) \in E$, $v_i \cdot v_j \leq -1 + 2x$, and hence $\|v_i + v_j\| \leq 2\sqrt{x}$. We now bound the probability that i and j are both in I'.

$$\begin{aligned} &Pr(v_i \cdot r \geq c \wedge v_j \cdot r \geq c) \\ &\leq Pr((v_i + v_j) \cdot r \geq 2c) \\ &= Pr\left(\frac{v_i + v_j}{\|v_i + v_j\|} \cdot r \geq \frac{2c}{\|v_i + v_j\|}\right) \\ &= N\left(\frac{2c}{\|v_i + v_j\|}\right) \leq N\left(\frac{c}{\sqrt{x}}\right). \end{aligned}$$

We therefore get that the expectation of X is at most

$$N\left(\frac{c}{\sqrt{x}}\right) \cdot \sum_{(i,j) \in E} (w_i + w_j) \leq N\left(\frac{c}{\sqrt{x}}\right) \cdot w(V) \cdot \Delta.$$

We are now ready to determine c. We need the following claim.

CLAIM 2.5. *Let $c = \sqrt{\frac{2x}{1-x} \cdot \ln\Delta}$. If $x = \omega(\frac{1}{\ln\Delta})$, then, for sufficiently large Δ, the expectation of X is at most $(1 + o(1))\sqrt{x}$ fraction of the expected total weight of I'.*

Proof. As $x = \omega(\frac{1}{\ln\Delta})$, we have that $c = \omega(1)$, and therefore by Lemma 2.2 we get

$$\begin{aligned} \frac{N\left(\frac{c}{\sqrt{x}}\right)}{N(c)} &\leq \frac{\phi\left(\frac{c}{\sqrt{x}}\right)\sqrt{x}}{\phi(c)(1 - \frac{1}{c^2})} \\ &\leq \frac{(1 + o(1))\sqrt{x}\phi\left(\frac{c}{\sqrt{x}}\right)}{\phi(c)} = \frac{(1 + o(1))\sqrt{x}}{\Delta}. \end{aligned}$$

Thus, we get that

$$N\left(\frac{c}{\sqrt{x}}\right) \leq \frac{(1 + o(1))\sqrt{x}N(c)}{\Delta}.$$

As the expectation of X is at most $w(V) \cdot \Delta \cdot N(\frac{c}{\sqrt{x}})$, and the expected total weight of I' is $w(V) \cdot N(c)$, the claim follows. □

Let $\alpha w(V)$ be the expectation of $w(I') - X$. By Claim 2.5, $\alpha \geq N(c)(1 - 2\sqrt{x})$. Furthermore, $w(I') - X \leq w(V)$. Consider the variable $w(V) - w(I') + X$. It is nonnegative with expectation $(1 - \alpha)w(V)$, and therefore, by applying the Markov inequality to $w(V) - w(I') + X$, we get

$$Pr(w(I') - X \geq \frac{\alpha w(V)}{2}) \geq \frac{\alpha}{2 - \alpha}.$$

For each $i \in I'$ let $d_{I'}(i)$ be the degree of i in the subgraph induced by I'. In the induced subgraph by I' we get that

$$\sum_{d_{I'}(i)>0} w_i \leq \sum_{(i,j)\in e(I')} (w_i + w_j) = X.$$

Therefore, if we take $I = I' \setminus \{i : d_{I'}(i) > 0\}$ we are left with an independent set of weight at least $w(I') - X$, which is at least $\frac{\alpha w(V)}{2}$ with probability at least $\frac{\alpha}{2-\alpha}$. By Lemma 2.2 we have

$$\alpha \geq \frac{N(c)}{2} \geq \frac{C'}{\Delta^{\frac{x}{1-x}} \cdot \sqrt{x \cdot \ln \Delta}} \geq \frac{C}{\Delta^x \cdot \sqrt{x \cdot \ln \Delta}}, \tag{4}$$

where C and C' are some constants. The last inequality holds since $x = \Theta(\frac{\ln \ln \Delta}{\ln \Delta})$. It is easy to see that $\frac{2-\alpha}{\alpha}$ is only polylogarithmic in Δ, and therefore, if we repeat this process a polylogarithmic number of times, with very high probability, one of those times, we will get an independent set of size at least $\frac{\alpha w(V)}{2}$, and by (4) the lemma is proved. □

We are now ready to prove Theorem 2.1. Let

$$x = \frac{\ln \ln \Delta - \frac{3}{2} \ln \ln \ln \Delta + \ln(C/2)}{\ln \Delta}.$$

Clearly, $x = \frac{\ln \ln \Delta}{\ln \Delta}(1 - o(1))$. For $i = 1, 2$, let $value(S_i) = \sum_{i \in S_i} w_i \frac{1 + v_0 \cdot v_i}{2}$; that is, $value(S_i)$ is the contribution of the vertices in S_i to the semidefinite program. Clearly, it is sufficient to show that

$$\frac{w(S_1) + w(S_2) - w(I)}{value(S_1) + value(S_2)} \leq 2 - \frac{2 \ln \ln \Delta}{\ln \Delta} + o\left(\frac{\ln \ln \Delta}{\ln \Delta}\right).$$

As $value(S_1) \geq w(S_1)(1 + x)/2$, then

$$\frac{w(S_1)}{value(S_1)} \leq \frac{2}{1 + x} = 2 - 2x + o(x). \tag{5}$$

Moreover, as $x = \Theta(\frac{\ln \ln \Delta}{\ln \Delta})$, then by Lemma 2.4 we have that

$$\begin{aligned} w(S_2) - w(I) &\leq w(S_2)\left(1 - \frac{C}{\Delta^x \sqrt{x \ln \Delta}}\right) \\ &= w(S_2)\left(1 - \frac{2 \ln \ln \Delta}{\ln \Delta}\right) = w(S_2)(1 - 2x + o(x)), \end{aligned}$$

and, as $value(S_2) \geq w(S_2)(1-x)/2$, we get that

$$\frac{w(S_2) - w(I)}{value(S_2)} \leq \frac{2 - 4x + o(x)}{1-x} = 2 - 2x + o(x). \tag{6}$$

By (5) and (6), we get that the performance ratio of the algorithm is at most $2 - 2x + o(x)$, which proves Theorem 2.1. □

3. Vertex cover in *p*-claw free graphs. In this section we show that the same algorithm gives a better performance for p-claw free graphs.

We first note that in the proof of Theorem 2.1, Δ can be replaced by the maximal degree in S_2, since we needed only an upper bound on the degrees of S_2 to obtain the bound on X. We thus have the following claim.

CLAIM 3.1. *Let $G = (V, E)$ be a weighted graph. Let $\Delta(S_2)$ be the maximal degree of the subgraph induced by S_2 in algorithm* VC-G *running on G. Algorithm* VC-G *achieves a ratio of*

$$2 - \frac{2 \ln \ln \Delta(S_2)}{\ln \Delta(S_2)} + o\left(\frac{\ln \ln \Delta(S_2)}{\ln \Delta(S_2)}\right).$$

We now prove that our algorithm gives a better ratio for p-claw free graphs.

THEOREM 3.2. *Let $G = (V, E)$ be a weighted $(p+1)$-claw free graph. Algorithm* VC-G *achieves a performance ratio of*

$$2 - \frac{2 \ln \ln p}{\ln p} + o\left(\frac{\ln \ln p}{\ln p}\right).$$

Proof. We first show that S_2 contains no triangles. Assume the opposite. Let $i, j, k \in S_2$ be a triangle in S_2. By definition of S_2, we have

$$0 \leq (v_i + v_j + v_k)^2 \leq -3 + 12x < 0.$$

Therefore, the set of neighbors of every vertex in S_2 is an independent set. As G is a $(p+1)$-claw free graph, we have that the largest degree in S_2 cannot exceed p. Thus, by Claim 3.1 we get the desired ratio. □

4. Vertex cover in hypergraphs. For a given k, let $H = (V, E)$ be a hypergraph, with m edges and with n vertices, in which every edge is of size at most k. Assume $V = \{1, \ldots, n\}$ and that for every vertex i there is a positive weight w_i. We consider the minimum *vertex cover* problem in H. Again, let $vc(H)$ denote the weight of the minimum vertex cover in H. Let Δ be the maximal degree in H. For a set $S \subseteq V$, let $e(S)$ denote the set of edges in the subhypergraph induced by S, and let $w(S)$ denote the total weight of S. We assume for simplicity that the hypergraph is k-uniform. Our results can be easily extended to the nonuniform case.

4.1. Relaxations for the vertex cover problem in hypergraphs. Consider the next semidefinite relaxation to the problem:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n} w_i \cdot \frac{1 + v_0 \cdot v_i}{2}, \\
& \sum_{i,j \in e, i<j} v_i \cdot v_j \leq \sum_{i \in e} v_0 \cdot v_i + \frac{(k-4)(k-1)}{2} + k - 2 \quad , \quad e \in E, \\
& \sum_{i \in e} v_0 \cdot v_i \geq -k + 2 \quad , \quad e \in E, \\
& v_i \cdot v_i = 1 \quad , \quad i \in V.
\end{aligned} \tag{7}$$

Algorithm. VC-H

Input: A weighted graph $G = (V, E)$.
Output: A vertex cover of G.

1. Solve the semidefinite program (7).
2. Let $S_1 = \{i \mid v_0 \cdot v_i \geq b\}$ and $S_2 = \{i \mid a \leq v_0 \cdot v_i < b\}$.
3. Project all the vectors in S_2 to the space orthogonal to v_0 and normalize the vectors. For every $i \in S_2$, let v_i' be the projected vector.
4. Choose a random n-dimensional vector r, and let $I' = \{i \in S_2 \mid v_i' \cdot r \geq c\}$.
5. Remove vertices with positive degree from I' until there is no edge in I'. Let I be the remaining independent set.
6. Output $S_1 \cup (S_2 \setminus I)$.

FIG. 2. *Algorithm* VC-H *for finding a small vertex cover in a hypergraph.*

We first claim that this is indeed a relaxation; i.e., for every vertex cover C of H, there are vectors satisfying the constraints of relaxation (7) which also satisfy $\sum_{i=1}^{n} w_i \cdot \frac{1+v_0 \cdot v_i}{2} = w(C)$. Let $C \subseteq V$ be a vertex cover of H. Let v_0 be an arbitrary unit vector. Assign v_i to be v_0 if $i \in C$ and $-v_0$ otherwise. It is clear that $\sum_{i=1}^{n} w_i \cdot \frac{1+v_0 \cdot v_i}{2} = w(C)$. We still have to show that the constraints of the relaxation are satisfied. Consider an edge $e \in E$. By the definition of the vectors, not all $i \in e$ equal $-v_0$, and therefore the second constraint holds trivially; i.e., we have that $\sum_{i \in e} v_0 \cdot v_i \geq -k + 2$. It is easy to see that if not all vectors in e equal v_0, we have

$$\sum_{i,j \in e, i<j} v_i \cdot v_j \leq \frac{(k-4)(k-1)}{2},$$

and therefore

$$\sum_{i,j \in e, i<j} v_i \cdot v_j \leq \sum_{i \in e} v_0 \cdot v_i + \frac{(k-4)(k-1)}{2} + k - 2.$$

If all vectors in e equal v_0, the above trivially holds. Therefore all constraints are satisfied and this is indeed a relaxation.

4.2. The approximation algorithm. We are now ready to describe our algorithm. The algorithm is an extension of algorithm VC-G given in Figure 1. The input of the algorithm is $H = (V, E)$, and the weight function is w. Let x be a small positive number to be determined later. Let c be a positive number to be determined later. Moreover, let $a = --1 + \frac{2}{k} - (k-1)x$ and $b = -1 + \frac{2}{k} + x$. The algorithm is given in Figure 2.

It is easy to see that $S_1 \cup (S_2 \setminus I)$ is a cover for the following reason: Assume the opposite. Let $e \in E$ be an edge which is not covered by $S_1 \cup (S_2 \setminus I)$, i.e., $e \subseteq I \cup (V \setminus (S_1 \cup S_2))$. If there exists $i \in (V \setminus (S_1 \cup S_2)) \cap e$, then there is $j \in S_1 \cap e$; otherwise, we have

$$\sum_{j \in e} v_0 \cdot v_i < a + (k-1) \cdot b = -k + 2,$$

which is a contradiction. We can therefore assume that all $i \in e$ are in I, but I is independent, which is a contradiction. Therefore, algorithm VC-H finds a cover to the hypergraph.

4.3. Analysis of the algorithm. In this section we prove the following theorem:

THEOREM 4.1. *Let k be an integer such that $k^3 = o(\frac{\ln\ln\Delta}{\ln\ln\ln\Delta})$. Let $H = (V, E)$ be a weighted k-uniform hypergraph with maximal degree Δ. There is a proper choice of x and c such that the randomized algorithm* VC-H *achieves a performance ratio of*

$$k - k(k-1)\frac{\ln\ln\Delta}{\ln\Delta} + o\left(\frac{\ln\ln\Delta}{\ln\Delta}\right).$$

In order to prove Theorem 4.1, we first have to bound the weight of I. We need the following lemma, which bounds the probability that a given edge in the subhypergraph induced by S_2 will be in I'.

LEMMA 4.2. *Let $e \in E$ be an edge in the subhypergraph induced by S_2. If $x = \Theta(\frac{\ln\ln\Delta}{\ln\Delta})$, then, for sufficiently large Δ, the probability that e will be an edge in the subhypergraph induced by I' is at most $N(\frac{c}{\sqrt{d_k \cdot x}})$, where $d_k = \frac{k}{2(k-1)}$.*

Proof. For a given $i \in e$, let u_i be the projection of v_i to the space orthogonal to v_0. Let $a_i = v_0 \cdot v_i$. We therefore have that $v_i = a_i \cdot v_0 + u_i$. As $i \in S_2$, we know that $a \le a_i < b$, and thus $a^2 \ge a_i^2 > b^2$. Let $v_i' = \frac{u_i}{\|u_i\|}$ be the normalized projection of v_i. By the definition of a_i, u_i,

$$u_i \cdot u_j = v_i \cdot v_j - a_i \cdot a_j$$

and

$$\|u_i\| = \sqrt{1 - a_i^2} \ge \sqrt{1 - a^2}.$$

Now, by the constraints of relaxation (7) we have

$$\sum_{i,j \in e, i<j} v_i \cdot v_j \le \frac{(k-4)(k-1)}{2} + kx.$$

Considering all the above we get

$$\begin{aligned}
\sum_{i,j\in e,i<j} v_i' \cdot v_j' &= \sum_{i,j\in e,i<j} \frac{u_i \cdot u_j}{\|u_i\| \cdot \|u_j\|} \\
&= \sum_{i,j\in e,i<j} \frac{v_i \cdot v_j - a_i \cdot a_j}{\sqrt{(1-a_i^2)(1-a_j^2)}} \\
&\le \frac{\sum_{i,j\in e,i<j}(v_i \cdot v_j - a_i \cdot a_j)}{1-a^2} \\
&\le \frac{(k-4)(k-1)/2 + kx - \binom{k}{2}b^2}{1-a^2} \\
&= -\frac{k}{2} + \frac{k^2 \cdot d_k}{2} \cdot x - \frac{k^4 x^2}{4(k-1)(2+kx)} \le -\frac{k}{2} + \frac{k^2 \cdot d_k}{2} \cdot x.
\end{aligned}$$

The last equality follows by assigning $a = -1 + \frac{2}{k} - (k-1)x$, $b = -1 + \frac{2}{k} + x$, and using some simple, though cumbersome, calculations. We can therefore bound the

norm of the sum of the projected vectors v'_i for $i \in e$:

$$\left\| \sum_{i \in e} v'_i \right\|^2 \leq \frac{k^3 \cdot x}{2(k-1)} = k^2 \cdot d_k \cdot x.$$

The probability that all $i \in e$ will be in I' is bounded by

$$Pr\left(\left(\sum_{i \in e} v'_i \right) \cdot r \geq k \cdot c \right) = N\left(\frac{k \cdot c}{\| \sum_{i \in e} v'_i \|} \right) \leq N\left(\frac{c}{\sqrt{d_k \cdot x}} \right),$$

and the lemma follows. □

For every vertex, the probability that this vertex will be in I' is $N(c)$. Therefore, the expected total weight of I' is $w(S_2) \cdot N(c)$. Let $X = \sum_{e \in e(I')} \sum_{i \in e} w_i$. We want to find c such that X will be smaller than $w(I')$ by at least a constant factor.

LEMMA 4.3. *Let* $c = \sqrt{\frac{2d_k \cdot x}{1 - d_k \cdot x} \cdot \ln \Delta}$. *If* $x = \Theta(\frac{\ln \ln \Delta}{\ln \Delta})$, *then, for sufficiently large* Δ, *the expectation of* X *is at most* $\sqrt{2x}$ *fraction of the expected weight of the vertices in* I'.

Proof. As $x = \omega(\frac{1}{\ln \Delta})$, we have that $c = \omega(1)$, and therefore by Lemma 2.2 we have that

$$\begin{aligned} N\left(\frac{c}{\sqrt{d_k \cdot x}} \right) &\leq \phi\left(\frac{c}{\sqrt{d_k \cdot x}} \right) \cdot \frac{\sqrt{d_k \cdot x}}{c} \\ &= \phi(c) \cdot \frac{\sqrt{d_k \cdot x}}{\Delta \cdot c} \\ &\leq \frac{N(c)}{\Delta} \cdot \frac{\sqrt{d_k \cdot x}}{1 - 1/c^2} \leq \frac{\sqrt{2x} N(c)}{\Delta}. \end{aligned} \tag{8}$$

Now, the expected total weight of I' is $w(S_2) \cdot N(c)$. By the linearity of the expectation and by Lemma 4.2, the expectation of X can be bounded as follows:

$$\begin{aligned} E(X) &= \sum_{e \in E(S_2)} \sum_{i \in e} w_i \cdot Pr(e \in e(I')) \\ &\leq N\left(\frac{c}{\sqrt{d_k \cdot x}} \right) \cdot \sum_{e \in E} \sum_{i \in e} w_i \\ &\leq N\left(\frac{c}{\sqrt{d_k \cdot x}} \right) \cdot \sum_{i \in V} \Delta \cdot w_i \\ &= N\left(\frac{c}{\sqrt{d_k \cdot x}} \right) \cdot \Delta \cdot w(S_2). \end{aligned}$$

Thus, by (8), the lemma follows. □

Let $\alpha_k w(S_2)$ be the expectation of $w(I') - X$. $w(I') - X \leq w(S_2)$, and therefore, by the Markov inequality applied to $w(S_2) - w(I') + X$, the probability that $w(I') - X$ is at least $\frac{\alpha_k w(S_2)}{2}$ is lower bounded by $\frac{\alpha_k}{2 - \alpha_k}$. By the definition of α_k, for some constant C,

$$\alpha_k \geq (1 - \sqrt{2x}) N(c) \geq \frac{2C}{\Delta^{d_k \cdot x} \sqrt{d_k x \ln \Delta}}. \tag{9}$$

We are now ready to prove Theorem 4.1. Once again, for $i = 1, 2$, let $value(S_i) = \sum_{i \in S_i} \frac{1+v_0 \cdot v_i}{2}$. We first determine x. Set

$$x = \frac{\ln \ln \Delta - \frac{3}{2} \ln \ln \ln \Delta + \ln(Cd_k) - \ln(k^2)}{d_k \ln \Delta}.$$

Notice that, by the terms of the lemma, k is very small, and

$$x \leq \frac{\ln \ln \Delta}{d_k \ln \Delta}, \tag{10}$$

$$k^3 x = o(1). \tag{11}$$

By the definition of x,

$$\Delta^{d_k x} = \frac{d_k \ln \Delta}{k^2 \sqrt{\pi} (\ln \ln \Delta)^{3/2}},$$

and thus by (9) and (10)

$$\frac{\alpha_k}{2} = \frac{k^2 x}{2} \left(\frac{\ln \ln \Delta}{x d_k \ln \Delta} \right)^{3/2} \geq \frac{k^2 x}{2}.$$

Therefore, as $\frac{2-\alpha_k}{\alpha_k}$ is only logarithmic in Δ, if we repeat the process a polylogarithmic number of times, then with high probability, in at least one of these times, we will get an independent set I of total weight at least $\frac{k^2 x w(S_2)}{2}$. Therefore, in this case

$$w(S_2) - w(I) \leq w(S_2) \left(1 - \frac{k^2 x}{2} \right).$$

On the other hand, $value(S_2) \geq w(S_2)(\frac{1}{k} - \frac{(k-1)x}{2})$. Combining these facts with (11), we get

$$\frac{w(S_2) - w(I)}{value(S_2)} \leq \frac{1 - \frac{k^2 x}{2}}{\frac{1}{k} - \frac{(k-1)x}{2}} = k - \frac{k^2 x}{2} + \frac{k^3 x^2}{2} = k - \frac{k^2 x}{2} + o(x). \tag{12}$$

From (10) and (11), and as $value(S_1) \geq w(S_1)(\frac{1}{k} + \frac{x}{2})$, we get that

$$\frac{w(S_1)}{value(S_1)} \leq \frac{2k}{2 + kx} = k - \frac{k^2 x}{2} + o(x). \tag{13}$$

By (12) and (13) we get that the performance ratio of the algorithm is at most

$$k - \frac{k^2 x}{2} + o(x) = k - k(k-1) \frac{\ln \ln \Delta}{\ln \Delta} + o\left(\frac{\ln \ln \Delta}{\ln \Delta} \right),$$

and thus Theorem 4.1 is proved.

It is straightforward that our algorithm can be extended to hypergraphs with maximal edge size k. This can be obtained in the following way: Given a hypergraph H with maximal edge size k, take every edge e of size l, where $l < k$, add new $k - l$ vertices to H, and replace e by a new edge containing the new $k - l$ vertices and the vertices of e. It is obvious that the new hypergraph is a k-uniform hypergraph and that Δ remains the same; thus Theorem 4.1 can be used.

It is easy to see that for k which satisfy $k^3 = o(\frac{\ln \ln n}{\ln \ln \ln n})$, for hypergraphs with maximal edge size k, algorithm VC-H achieves a ratio of $k - (1 - o(1))k \frac{\ln \ln n}{\ln n}$, as the maximal degree Δ cannot exceed n^{k-1}.

Algorithm. IS

Input: A weighted graph $G = (V, E)$.
Output: An independent set of G.

1. Solve relaxation (14).
2. Let $S_0 = \{i \in V \mid \frac{v_0 \cdot v_i + 1}{2} < \frac{1}{p}\}$, $S_1 = \{i \in V \mid \frac{1}{p} \leq \frac{v_0 \cdot v_i + 1}{2} < \frac{1}{2}\}$, and $I_2 = S_2 = \{i \in V \mid \frac{1}{2} \leq \frac{v_0 \cdot v_i + 1}{2}\}$.
3. Use the greedy algorithm on the set S_0 to find an independent set $I_0 \subseteq S_0$.
4. Project all vectors v_i corresponding to vertices in S_1 to the space orthogonal to v_0 and normalize them. The projected vectors will be denoted by v_i'.
5. Choose a random n-dimensional vector r, and let $I' = \{i \in S_1 \mid v_i' \cdot r \geq c\}$.
6. Remove all nonisolated vertices in the subgraph induced by I'. The remaining set is I_1.
7. Return the set with the largest total weight between I_0, I_1, I_2.

FIG. 3. *Algorithm* IS *for finding a large independent set on a graph.*

5. Independent set approximation. In this section we consider a simple application of our methods to the *independent set* problem. Recently, the same result was derived independently by Halldórsson [10], using a different approach.

Let $G = (V, E)$ be a weighted graph. Assume $V = \{1, \ldots, n\}$ and that for every $1 \leq i \leq n$ there is a positive weight w_i. Let Δ be the maximal degree in G. Consider the following semidefinite relaxation for the *independent set* problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} w_i \cdot \frac{1 + v_0 \cdot v_i}{2}, \\ & (v_i + v_0) \cdot (v_j + v_0) = 0 \quad , \quad (i,j) \in E, \\ & \|v_i\| = 1 \quad , \quad i \in V. \end{aligned} \tag{14}$$

It is easy to see that this is indeed a relaxation to the *independent set* problem. It can be derived from relaxation (3) by replacing v_0 by $-v_0$.

Let $p \geq 3$ and c be parameters to be chosen later. Consider the algorithm presented in Figure 3. We show that this algorithm achieves the desired ratio. In the next lemma, we show that partitioning the problem into three parts reduces the performance of the algorithm by a factor of three.

LEMMA 5.1. *Assume $Z = Z_0 + Z_1 + Z_2$, where Z_k, for $k = 0, 1, 2$, is the value of the relaxation restricted to the set S_k. For $k = 0, 1, 2$, let $I_k \subseteq S_k$ be an independent set which satisfies $w(I_k) \geq \beta \cdot Z_k$. Let m be such that $w(I_m)$ is maximized. Then $w(I_m) \geq \beta \cdot \alpha(G)/3$, where $\alpha(G)$ is the maximum weight of an independent set in G.*

Proof. It is clear that $Z \geq \alpha(G)$. Therefore, we have the following inequalities:

$$w(I_m) \geq \frac{w(I_0) + w(I_1) + w(I_2)}{3} \geq \frac{\beta(Z_0 + Z_1 + Z_2)}{3} = \frac{\beta}{3} \cdot Z \geq \frac{\beta}{3} \cdot \alpha(G).$$

Therefore, the lemma follows. ∎

We now show that algorithm IS finds an independent set of total weight at least $\Theta(\frac{\log \Delta}{\Delta \cdot \log\log \Delta} \cdot \alpha(G))$. We use Lemma 5.1 with $\beta = \frac{\log \Delta}{\Delta \log\log \Delta}$.

For S_0, we know that $Z_0 \leq \frac{w(S_0)}{p}$ and that the greedy algorithm produces an independent set I_0 of total weight at least $w(S_0)/(\Delta+1)$. Therefore, we get that $w(I_0)$ is of total weight at least $\frac{p}{\Delta+1} \cdot Z_0$.

For S_2, by the constraints of relaxation (14), we have that S_2 is an independent set, and therefore I_2 is an independent set, and thus $w(I_2) = w(S_2) = Z_2$.

We now analyze the case of S_1. For this we need the following lemma.

LEMMA 5.2. *The randomized algorithm* IS *produces with a nonnegligible probability a set I_1 of total weight at least $\Omega(\frac{\Delta^{2/p} \cdot Z_1}{\Delta\sqrt{\ln \Delta}})$.*

Proof. As in section 4.3, for every $i \in S_1$, let $a_i = v_0 \cdot v_i$ and assume $v_i = a_i v_0 + u_i$, where $u_i \cdot v_0 = 0$. For $i \in S_1$, we have that $-1 + \frac{2}{p} \leq a_i < 0$. Now,

$$\|u_i\| = \sqrt{1 - a_i^2},\ u_i \cdot u_j = v_i \cdot v_j - a_i a_j.$$

For $i \in S_1$, let $v_i' = \frac{u_i}{\|u_i\|}$ be the normalized projection of v_i. We thus have the following for $(i,j) \in E$:

$$\begin{aligned}
\frac{u_i \cdot u_j}{\|u_i\|\|u_j\|} &= \frac{v_i \cdot v_j - a_i a_j}{\sqrt{(1-a_i^2)(1-a_j^2)}} \\
&= \frac{-1 - a_i - a_j - a_i a_j}{\sqrt{(1-a_i^2)(1-a_j^2)}} \\
&= -\sqrt{\frac{(1+a_i)(1+a_j)}{(1-a_i)(1-a_j)}} \\
&\leq -\frac{1}{p-1}.
\end{aligned}$$

Let $x = \frac{p-2}{2p-2}$. As $p \geq 3$, $x \geq \frac{1}{4}$. We thus have that for every $(i,j) \in e(S_2)$, $v_i' \cdot v_j' \leq -1 + 2x$. Let

$$X = \sum_{(i,j) \in e(I')} (w_i + w_j).$$

As $x \geq \frac{1}{4}$, we have that $x = \omega(\frac{1}{\ln \Delta})$, and thus, by choosing $c = \sqrt{\frac{2x}{1-x} \ln \Delta}$, the terms of Claim 2.5 are fulfilled, and, for sufficiently large Δ, the expectation of X is at most $(1 + o(1))\sqrt{x}$ fraction of the expected total weight of I'. As $x \leq \frac{1}{2}$, we get that the expectation of X is at most $(1 + o(1))\frac{1}{\sqrt{2}}$ fraction of the expectation of I'. Therefore, the expectation of $w(I') - X$ is $\Omega(w(S_1)N(c))$. By applying the Markov inequality to nonnegative random variable $w(S_1) - w(I') + X$, we get that the probability that $w(I') - X$ is less than half its expectation is lower bounded by $\Omega(N(c)) = \Omega(\frac{\Delta^{2/p}}{\Delta\sqrt{\ln \Delta}})$. Clearly, by repeating this process a polynomial number of times, with high probability, in at least one of the iterations $w(I') - X$ is at least half its expectation. By the fact that $\sum_{d_{I'}(i)>0} w_i \leq X$, by taking off vertices with positive degree from the subgraph induced by I', we are left with an independent set of total weight at least

$$w(I') - X = \Omega(w(S_1) \cdot N(c)) = \Omega\left(\frac{\Delta^{2/p} w(S_1)}{\Delta\sqrt{\ln \Delta}}\right).$$

Now, $Z_1 \leq \frac{w(S_1)}{2}$, and so

$$w(I_1) \geq \Omega\left(\frac{\Delta^{2/p} \cdot Z_1}{\Delta\sqrt{\ln \Delta}}\right). \qquad \square$$

Altogether, we have the following:

$$\begin{aligned} w(I_0) &\geq \tfrac{p}{\Delta+1} Z_0, \\ w(I_1) &\geq \Omega\big(\tfrac{\Delta^{2/p}}{\Delta\sqrt{\ln \Delta}} \cdot Z_1\big), \\ w(I_2) &= Z_2. \end{aligned}$$

By Lemma 5.1, setting $p = \frac{4 \ln \Delta}{3 \ln \ln \Delta}$ gives a ratio of $\Omega(\frac{\ln \Delta}{\Delta \ln \ln \Delta})$, as desired.

6. Concluding remarks. We obtained improved algorithms for finding small vertex covers in bounded degree graphs and hypergraphs. On graphs with maximum degree at most Δ, we presented an algorithm which achieves a performance ratio of $2-(1-o(1))\frac{2 \ln \ln \Delta}{\ln \Delta}$. For general graphs, we get a ratio of $2-(1-o(1))\frac{2 \ln \ln n}{\ln n}$. It will be interesting if an approximation algorithm with a ratio of $2-f(\Delta)$ will be introduced for the *vertex cover* problem in bounded degree graphs, where $f(n) = \omega(\frac{\ln \ln n}{\ln n})$. It is also interesting if, for such a function f, an approximation algorithm with a ratio of $2-f(n)$ exists for general graphs. It will also be interesting if a $(k - f(n))$-approximation algorithm will be found for k-uniform hypergraphs, with such a function f.

Acknowledgment. The author would like to thank Uri Zwick for many helpful discussions and comments.

REFERENCES

[1] N. Alon and N. Kahale, *Approximating the independence number via the θ-function,* Math. Programming, 80 (1998), pp. 253–264.

[2] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem,* Ann. Discrete Math., 25 (1985), pp. 27–45.

[3] P. Berman and T. Fujito, *On the approximation properties of independent set problem in degree* 3 *graphs,* in Algorithms and Data Structures, Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, pp. 449–460.

[4] P. Berman and M. Fürer, *Approximating maximum independent set in bounded degree graphs,* in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, Daniel D. Sleator, ed., ACM, New York, 1994, pp. 365–371.

[5] V. Chvatal, *A greedy heuristic for the set covering problem,* Math. Oper. Res., 4 (1979), pp. 233–235.

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

[7] W. Feller, *An Introduction to Probability Theory and Its Applications,* 3rd ed., John Wiley & Sons, New York, 1968.

[8] M. Grötschel, L. Lovász, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization,* Combinatorica, 1 (1981), pp. 169–197.

[9] M. M. Halldórsson, *Approximations of Independent Sets in Graphs,* a survey paper from APPROX, 1998.

[10] M. M. Halldórsson, *Approximations of weighted independent sets and hereditary subset problems,* J. Graph Algorithms Appl., 4 (2000), pp. 1–16.

[11] M. M. Halldórsson and J. Radhakrishnan, *Improved approximations of independent sets in bounded-degree graphs,* in Algorithm Theory—SWAT '94: 4th Scandinavian Workshop on Algorithm Theory, Aarhus, Denmark, 1994, Lecture Notes in Comput. Sci. 824, Erik M. Schmidt and Sven Skyum, eds., Springer-Verlag, Berlin, pp. 195–206.

[12] M. M. Halldórsson and J. Radhakrishnan, *Improved approximations of independent sets in bounded-degree graphs via subgraph removal,* Nordic J. Comput., 1 (1994), pp. 475–492.

[13] M. M. Halldórsson and J. Radhakrishnan, *Greed is good: Approximating independent sets in sparse and bounded-degree graphs,* Algorithmica, 18 (1997), pp. 145–163.

[14] J. Håstad, *Clique is hard to approximate within $n^{1-\epsilon}$,* Acta Math., 182 (1999), pp. 105–142.
[15] J. Håstad, *Some optimal inapproximability results,* J. ACM, 48 (2001), pp. 798–859.
[16] D. S. Hochbaum, *Approximation algorithms for the set covering and vertex cover problems,* SIAM J. Comput., 11 (1982), pp. 555–556.
[17] D. S. Hochbaum, *Efficient bounds for the stable set, vertex cover and set packing problems,* Discrete Appl. Math., 6 (1983), pp. 243–254.
[18] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming,* J. ACM, 45 (1998), pp. 246–265.
[19] J. Kleinberg and M. X. Goemans, *The Lovász theta function and a semidefinite programming relaxation of vertex cover,* SIAM J. Discrete Math., 11 (1998), pp. 196–204.
[20] M. Krivelevich, *Approximate set covering in uniform hypergraphs,* J. Algorithms, 25 (1997), pp. 118–143.
[21] B. Monien and E. Speckenmeyer, *Ramsey numbers and an approximation algorithm for the vertex cover problem,* Acta Inform., 22 (1985), pp. 115–123.
[22] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes,* J. Comput. System Sci., 43 (1991), pp. 425–440.
[23] F. Rendl, R. J. Vanderbei, and H. Wolkowicz, *Interior–Point Methods for Max–Min Eigenvalue Problems,* Technical Report SOR–93–15, Program in Statistics & Operations Research, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 1993.
[24] A. Rényi, *Probability Theory,* Elsevier/North-Holland, Amsterdam, London, New York, 1970.
[25] S. Vishwanathan, *personal communication,* 1996; cited in [9].

SIAM J. COMPUT.
Vol. 31, No. 5, pp. 1624–1643

DUAL-BOUNDED GENERATING PROBLEMS: ALL MINIMAL INTEGER SOLUTIONS FOR A MONOTONE SYSTEM OF LINEAR INEQUALITIES*

E. BOROS†, K. ELBASSIONI‡, V. GURVICH†, L. KHACHIYAN‡, AND K. MAKINO§

Abstract. We consider the problem of enumerating all minimal integer solutions of a monotone system of linear inequalities. We first show that, for any monotone system of r linear inequalities in n variables, the number of maximal infeasible integer vectors is at most rn times the number of minimal integer solutions to the system. This bound is accurate up to a *polylog*(r) factor and leads to a polynomial-time reduction of the enumeration problem to a natural generalization of the well-known dualization problem for hypergraphs, in which dual pairs of hypergraphs are replaced by dual collections of integer vectors in a box. We provide a quasi-polynomial algorithm for the latter dualization problem. These results imply, in particular, that the problem of incrementally generating all minimal integer solutions to a monotone system of linear inequalities can be done in quasi-polynomial time.

Key words. integer programming, complexity of incremental algorithms, dualization, quasi-polynomial time, monotone discrete binary functions, monotone inequalities, regular discrete functions

AMS subject classifications. 68R05, 68Q25, 68Q32

PII. S0097539701388768

1. Introduction.

Consider a system of r linear inequalities in n integer variables

$$Ax \geq b, \quad x \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}, \tag{1.1}$$

where $A \in \mathbb{R}^{r \times n}$ is a given $r \times n$-matrix, $b \in \mathbb{R}^r$ is an r-vector, $c \in \mathbb{R}^n_+$ is a nonnegative n-vector, some or all of whose components may be infinite, and $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{R}_+$ denote, respectively, the set of integers, the set of reals, and the set of nonnegative reals. A vector $x \in \mathcal{C}$ is called a *feasible solution* of (1.1) if $Ax \geq b$. We shall assume that (1.1) is a *monotone system*; i.e., if $x \in \mathcal{C}$ is feasible, then all vectors $y \in \mathcal{C}$ with $y \geq x$ are also feasible. For instance, (1.1) is monotone if the matrix A is nonnegative, though the nonnegativity of A is not necessary; e.g., if $r = 1$, $n = 2$, and $c = (1, 2)$, then the system $5x_1 - x_2 \geq 3$ is monotone.

Let us denote by $\mathcal{F}_{A,b,c}$ the set of all minimal feasible solutions for the monotone system (1.1):

$$\mathcal{F}_{A,b,c} = \left\{ y \;\middle|\; \begin{array}{c} y \in \mathcal{C}, \quad Ay \geq b, \quad \text{and} \\ \nexists \text{ feasible solution } x \text{ of (1.1)} \\ \text{such that } x \leq y \text{ and } x \neq y \end{array} \right\}.$$

*Received by the editors May 1, 2001; accepted for publication (in revised form) May 14, 2002; published electronically August 15, 2002. This research was supported by the National Science Foundation (grant IIS-0118635) and by the Office of Naval Research (grant N00014-92-J-1375). An extended abstract of this paper containing some of the results appeared in the *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming*, Crete, Greece, 2001.
http://www.siam.org/journals/sicomp/31-5/38876.html

†RUTCOR, Rutgers University, 640 Bartholomew Rd, Piscataway NJ 08854-8003 (boros@rutcor.rutgers.edu, gurvich@rutcor.rutgers.edu).

‡Department of Computer Science, Rutgers University, 110 Frelinghuysen Rd, Piscataway, NJ 08854-8004 (elbassio@paul.rutgers.edu, leonid@cs.rutgers.edu).

§Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan (makino@sys.es.osaka-u.ac.jp).

In this paper, we are concerned with the problem of generating all vectors of $\mathcal{F}_{A,b,c}$ for a monotone system (1.1). Since the number of minimal feasible solutions of a monotone system may be exponential in the size of the input, the efficiency of such generation is measured customarily in terms of the sizes of both the input and the output (see, e.g., [19]). More precisely, we focus on the problem of generating a "next" element of $\mathcal{F}_{A,b,c}$ sequentially.

GEN$(A, b, c, \mathcal{X})$. *Given a monotone system* (1.1) *defined by* (A, b, c) *and a subset* $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$ *of its minimal feasible vectors, either find a new minimal integral vector* $x \in \mathcal{F}_{A,b,c} \setminus \mathcal{X}$ *or show that no such vector exists; i.e.,* $\mathcal{X} = \mathcal{F}_{A,b,c}$.

Clearly, the entire set $\mathcal{F}_{A,b,c}$ can be generated by initializing $\mathcal{X} = \emptyset$ and iteratively solving the above problem $|\mathcal{F}_{A,b,c}| + 1$ times. Accordingly, we say that the family $\mathcal{F}_{A,b,c}$ can be generated in *incremental q-time* if the problem GEN$(A, b, c, \mathcal{X})$ can be solved in $q(n, r, |\mathcal{X}|)$ operations for every subset $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$. In particular, we say that $\mathcal{F}_{A,b,c}$ can be generated in *incremental polynomial* or *quasi-polynomial* time if q is a polynomial or quasi-polynomial expression, respectively. These complexity notions are stated here for the unit-cost model of computation, in which the $q(n, r, |\mathcal{X}|)$ bound applies to the number of comparisons $(\leq, \geq)$ and the arithmetic and rounding operations $(+, -, \times, /, \lfloor\ \rfloor)$ required to solve problem GEN$(A, b, c, \mathcal{X})$ for *real* input (A, b, c). All of our results in this paper will also be valid, with the same bound for the number of operations, for *rational* input (A, b, c) in the bit model of computation, where the number of bits used in the operations depends linearly on the binary size of the input data.

Let us note here that the above definition for the complexity of sequential generation, based on GEN$(A, b, c, \mathcal{X})$, does indeed grasp the essence of sequential generation correctly. Namely, if there is any algorithm $\mathbf{A}$ generating all the elements of $\mathcal{F}_{A,b,c}$ sequentially, say, in the order $\mathcal{F}_{A,b,c} = \{x^1, x^2, \ldots, x^K\}$ and $x^{K+1} =$ "STOP" such that $\{x^1, \ldots, x^k\}$ is generated by $q(n, r, k-1)$ operations for every $k = 1, \ldots, K+1$, then, using the same algorithm $\mathbf{A}$, problem GEN$(A, b, c, \mathcal{X})$ can also be solved in $O(q(n, r, |\mathcal{X}|))$ operations *for every subset* $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$.

The problem of generating the family $\mathcal{F}_{A,b,c}$ of all minimal feasible solutions to a monotone system (1.1) includes, as special cases, several well-known problems from the literature.

First, if A is integral and $c = +\infty$, the generation of $\mathcal{F}_{A,b,c}$ can be regarded as the computation of the Hilbert basis for the polynomial ideal which has $\{x \in \mathbb{Z}^n \mid Ax \geq b, x \geq 0\}$ as its Newton polyhedron.

If A is a binary matrix, $b = \mathbf{e}_r$, and $c = \mathbf{e}_n$, where $\mathbf{e}_d$ denotes the vector of all ones of dimension d, then $\mathcal{F}_{A,b,c}$ is the set of (characteristic vectors of) all minimal transversals to the hypergraph defined by the rows of A. In this case, problem GEN$(A, b, c, \mathcal{X})$ turns into the well-known *hypergraph dualization problem*, that is, the incremental enumeration of all minimal transversals (or, equivalently, all maximal independent sets) of a given hypergraph (see, e.g., [12]). For several special classes of hypergraphs, GEN$(A, \mathbf{e}_r, \mathbf{e}_n, \mathcal{X})$ can be solved efficiently, including 2-monotonic, threshold, matroid, read-bounded, and acyclic hypergraphs (see [4, 8, 9, 11, 13, 16, 21, 23, 24, 26, 27]), and, most notably, when the hyperedges are all of size 2, that is, when the hypergraph is the edge set of a graph (see [19, 20, 29]). The latter result has been extended to an incremental polynomial solution for all hypergraphs with bounded edge sizes (see, e.g., [6, 12, 13]), in which case even an efficient parallel solution exists (see [5]). There is no incremental polynomial-time algorithm known for the dualization of an arbitrary hypergraph. The best known result is an incremental

quasi-polynomial-time algorithm, proposed by Fredman and Khachiyan [14], which solves $\mathrm{GEN}(A, \mathbf{e}_r, \mathbf{e}_n, \mathcal{X})$ in $O(nt) + t^{o(\log t)}$ time, where $t = \max\{r, |\mathcal{X}|\}$. Recent improvements on this procedure can be found in [15, 28].

If A is binary, $c = \mathbf{e}_n$, and $b = Ac - \mathbf{e}_r$, then the vectors in $\mathcal{F}_{A,b,c}$ correspond (in a one-to-one way) to the maximal matchings of the hypergraph, formed by the columns of A. It was shown in [20] that $\mathrm{GEN}(A, b, c, \mathcal{X})$ can be solved in this case in polynomial time, thus providing an efficient generation of all maximal matchings. More generally, if A is binary, $c = \mathbf{e}_n$, and b is arbitrary, then the vectors in $\mathcal{F}_{A,b,c}$ are the characteristic vectors of the so-called *multiple transversals* of the hypergraph formed by the rows of A, and an incremental quasi-polynomial generation is provided in [7] for this case.

If $c = \mathbf{e}_n$ and $r = 1$, then (1.1) is also known as a *binary knapsack problem*, for which $\mathcal{F}_{A,b,c}$ can be generated in incremental polynomial time with an amortized complexity of $O(n^2)$ time per element as shown by Lawler, Lenstra, and Rinnooy Kan [20]. Improving on this result, Uno [30] showed recently that, in this special case, $\mathcal{F}_{A,b,c}$ can be generated in $O(n)$ time per element in the worst case.

More generally, Lawler, Lenstra, and Rinnooy Kan considered the case of a nonnegative matrix A with b arbitrary and $c = \mathbf{e}_n$ and conjectured that no efficient generation of $\mathcal{F}_{A,b,\mathbf{e}_n}$ is possible in this case unless P = NP.

As our main result, we shall show that problem $\mathrm{GEN}(A, b, c, \mathcal{X})$ can in fact be solved in quasi-polynomial time for any monotone system (1.1); that is, all minimal integer solutions of (1.1) can be generated in incremental quasi-polynomial time, both in the unit-cost and the bit models of computation.

THEOREM 1.1. *Problem GEN$(A, b, c, \mathcal{X})$ can be solved in $t^{o(\log t)}$ time, where* $t = \max\{n, r, |\mathcal{X}|\}$.

This result implies that $\mathrm{GEN}(A, b, c, \mathcal{X})$ cannot be NP-hard unless all NP-complete problems can be solved in quasi-polynomial time. We mention in passing that, if c is bounded and the number of nonzero coefficients per inequality in (1.1) is fixed, the results of [5] also imply that problem $\mathrm{GEN}(A, b, c, \mathcal{X})$ can be efficiently solved in parallel.

2. Uniformly dual-bounded monotone systems. Let us note first that, without any loss of generality, the finiteness of $\mathcal{C}$ can always be assumed. Namely, defining $J^* = \{j \mid c_j = \infty\}$, $J_* = \{1, \dots, n\} \setminus J^*$, and

$$\bar{c}_j = \begin{cases} \max_{i:a_{ij}>0} \left\lceil \frac{b_i - \sum_{k\in J_*} \min\{0, a_{ik}\} c_k}{a_{ij}} \right\rceil & \text{for } j \in J^* \text{ and} \\ c_j & \text{for } j \in J_*, \end{cases} \tag{2.1}$$

we have $\mathcal{F}_{A,b,c} = \mathcal{F}_{A,b,\bar{c}}$. To see this equality, let us consider an arbitrary vector $x = (x_1, \dots, x_n) \in \mathcal{F}_{A,b,c}$ such that $x_j > 0$ for some $j \in J^*$. We claim that $x_j \le \bar{c}_j$. For this, let us observe first that the restriction of A on J^* must be nonnegative: $a_{ij} \ge 0$ for all $i \in \{1, \dots, r\}$ and $j \in J^*$; otherwise, the monotonicity of (1.1) could be violated by sufficiently increasing x_j, the jth component of x. Thus we have

$$\begin{aligned} a_{ij}x_j + \sum_{k\in J_*} \min\{0, a_{ik}\} c_k &\le a_{ij}x_j + \sum_{k\in J_*} \min\{0, a_{ik}\} x_k \\ &\le a_{ij}x_j + \sum_{k\in J_*} a_{ik}x_k \\ &\le \sum_{k\in J^*} a_{ik}x_k + \sum_{k\in J_*} a_{ik}x_k = a^i x \end{aligned}$$

since the restriction of A on J^* is nonnegative, where a^i denotes the ith row of A. Let us consider now the vector x' obtained from x by decreasing its jth component by 1. Then $x' \in \mathcal{C}$, and, by the minimality of x, x' is infeasible for (1.1). Hence $b_i - a_{ij} \leq a^i x - a_{ij} = a^i x' < b_i$ for some $i \in \{1, \ldots, r\}$, implying $a^i x < b_i + a_{ij}$. Thus we can conclude that

$$a_{ij} x_j + \sum_{k \in J_*} \min\{0, a_{ik}\} c_k < b_i + a_{ij},$$

from which our claim follows.

Since the bounds in (2.1) are easy to compute and since appending these bounds to (1.1) does not change the set $\mathcal{F}_{A,b,c}$ by the above observation, we shall assume in what follows that all components of the nonnegative vector c are finite, even though this may not be the case for the original system. This assumption does not entail any loss of generality and allows us to consider $\mathcal{F}_{A,b,c}$ as a system of integral vectors in a finite box.

Let us also note that the input monotone system (1.1) can be assumed to be feasible and nontrivial; that is, $\mathcal{F}_{A,b,c} \neq \emptyset$ and $\mathcal{F}_{A,b,c} \neq \{0\}$. For a finite and nonnegative c, this is equivalent to $Ac \geq b$ and $0 \not\geq b$.

Generalizing systems of monotone inequalities, we shall consider arbitrary monotone subsets of the finite integral box $\mathcal{C}$. For a collection of integral vectors $\mathcal{A} \subseteq \mathcal{C}$, let us denote, respectively, by $\mathcal{A}^+ = \{x \in \mathcal{C} \mid x \geq a \text{ for some } a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{x \in \mathcal{C} \mid x \leq a \text{ for some } a \in \mathcal{A}\}$ the *ideal* and *filter* generated by $\mathcal{A}$. Generalizing standard terminology of the theory of hypergraphs, elements in $\mathcal{C} \setminus \mathcal{A}^+$ will be called *independent* of $\mathcal{A}$, and let us denote by $\mathcal{I}(\mathcal{A})$ the set of all *maximal independent* elements of $\mathcal{A}$. Then, for a finite box $\mathcal{C}$, we have

$$\mathcal{A}^+ \cap \mathcal{I}(\mathcal{A})^- = \emptyset \quad \text{and} \quad \mathcal{A}^+ \cup \mathcal{I}(\mathcal{A})^- = \mathcal{C}. \tag{2.2}$$

In particular, if $\mathcal{A} = \mathcal{F}_{A,b,c}$ is the set of all minimal feasible integral vectors for (1.1) and $\mathcal{C}$ is finite, then the ideal $\mathcal{F}^+_{A,b,c}$ is the set of all feasible solutions of (1.1), while the filter $\mathcal{C} \setminus \mathcal{F}^+_{A,b,c}$ is the collection of all *infeasible* vectors. This filter is generated by the set $\mathcal{I}(\mathcal{F}_{A,b,c})$ consisting of all *maximal infeasible* integral vectors for (1.1):

$$\mathcal{C} \setminus \mathcal{F}^+_{A,b,c} \;=\; \mathcal{I}(\mathcal{F}_{A,b,c})^- \;=\; \{x \in \mathcal{C} \mid Ax \not\geq b\} \;=\; \bigcup_{y \in \mathcal{I}(\mathcal{F}_{A,b,c})} \{y\}^-.$$

Let us return finally to the generation of minimal feasible solutions of a monotone system (1.1). Our approach for the solution of this problem will be based on the *joint generation* of the union $\mathcal{F}_{A,b,c} \cup \mathcal{I}(\mathcal{F}_{A,b,c})$. The following statement, ensuring the efficiency of such an approach, is instrumental in our proofs and may also be of independent interest.

THEOREM 2.1. *If the monotone system* (1.1) *is feasible, then, for any nonempty subset* $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$, *we have*

$$|\mathcal{I}(\mathcal{X}) \cap \mathcal{I}(\mathcal{F}_{A,b,c})| \leq rn|\mathcal{X}|. \tag{2.3}$$

In particular, for $\mathcal{X} = \mathcal{F}_{A,b,c}$, *this implies the inequality*

$$|\mathcal{I}(\mathcal{F}_{A,b,c})| \leq rn|\mathcal{F}_{A,b,c}|.$$

Using the terminology introduced in [7], the above statement claims that $\mathcal{F}_{A,b,c}$ for a monotone system (1.1) is *uniformly dual-bounded*. This property, more precisely inequality (2.3), guarantees that, if we manage to generate the elements of the union $\mathcal{F}_{A,b,c} \cup \mathcal{I}(\mathcal{F}_{A,b,c})$ one-by-one in some efficient way, then, at any given moment during this generation, the number of produced maximal infeasible vectors is polynomially limited by the number of obtained minimal feasible vectors. Thus, by simply disregarding the maximal infeasible vectors in this process, we get an efficient algorithm for the generation of minimal feasible vectors.

Let us remark here that such an approach would certainly be too wasteful for the generation of maximal infeasible solutions, as the following example shows. Let us consider the monotone system (A, b, c) consisting of the r inequalities $x_1 + x_2 \geq 1$, $x_3 + x_4 \geq 1, \ldots, x_{2r-1} + x_{2r} \geq 1$ in $n = 2r$ variables, with $c = \mathbf{e}_{2r}$. This system has 2^r minimal feasible binary vectors and only r maximal infeasible ones, i.e.,

$$|\mathcal{F}_{A,b,c}| \geq 2^{|\mathcal{I}(\mathcal{F}_{A,b,c})|}.$$

Thus $|\mathcal{F}_{A,b,c}|$ cannot be bounded by a polynomial in r, n, and $|\mathcal{I}(\mathcal{F}_{A,b,c})|$ in general. Therefore, in the joint generation process, we may get unlucky in the worst case and get first exponentially many minimal feasible vectors before obtaining the first maximal infeasible one.

In fact, not only is such a joint generation approach inefficient for the generation of maximal infeasible vectors, but, more generally, a result analogous to that of Theorem 1.1 is highly unlikely to hold.

PROPOSITION 2.2. *Let us consider a monotone system* (1.1)*, where A is an $r \times n$ binary matrix, $c = \mathbf{e}_n$, and all of the r components of b but one are equal to* 1*. Further, let $\mathcal{X} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$. Then it is NP-complete to decide if $\mathcal{I}(\mathcal{F}_{A,b,c}) \setminus \mathcal{X} \neq \emptyset$.*

Proof. We can use arguments analogous to those of [22]. Consider the well-known NP-hard problem of determining whether a given graph $G = (V, E)$ contains an independent set of size t, where $t \geq 2$ is a given threshold. Let us introduce $|V|$ binary variables $x_v, v \in V$, and write $|E|$ inequalities $x_v + x_{v'} \geq 1$, one for every edge $e = (v, v') \in E$, followed by the inequality $\sum_{v \in V} x_v \geq |V| - t$. It is easily seen that, if x is the characteristic vector of an edge $e \in E$, then $\mathbf{e} - x$ is a maximal infeasible binary vector for the above system of inequalities. Deciding whether there are other such vectors is equivalent to determining whether the given graph G has an independent set of size at least t. □

According to our earlier remark, Proposition 2.2 implies that no incrementally efficient generation of maximal infeasible vectors is possible, regardless of the order of their generation, unless all NP-hard problems can be solved with the same efficiency. On the other hand, it is easy to see that, if the right-hand side vector b is bounded by a constant and the matrix A is integral and nonnegative, then listing all maximal infeasible binary vectors of (1.1) can be done in polynomial time.

Note also that the enumeration of all maximal infeasible binary vectors to a system of nonnegative linear inequalities $a^1 x \geq b_1, \ldots, a^r x \geq b_r$, with integer coefficients, is equivalent to the enumeration of all maximal solutions satisfying at least one of r knapsack inequalities $a^1 x \leq b_1 - 1, \ldots, a^r x \leq b_r - 1$. An exponential algorithm for the latter problem and its relation to the facet enumeration for some $0, 1$-polytopes [1] are discussed in [20].

Let us remark finally that the bounds in Theorem 2.1 are sharp for $r = 1$: for instance, the single inequality $x_1 + \cdots + x_n \geq n$ in binary variables has only one maximal infeasible vector and exactly n minimal feasible ones. For larger values of

r, these bounds are accurate up to a factor polylogarithmic in r. To see this, let us consider the input (A, b, c) consisting of $r = 2^k$ inequalities of the form

$$x_{i_1} + x_{i_2} + \cdots + x_{i_k} \geq 1, \quad i_1 \in \{1, 2\},\ i_2 \in \{3, 4\}, \ldots, i_k \in \{2k-1, 2k\},$$

in $n = 2k$ variables, where $x = (x_1, \ldots, x_n) \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$. For any positive integer vector c, we have 2^k maximal infeasible integer vectors and only k minimal feasible integer vectors in this system; that is,

$$|\mathcal{I}(\mathcal{F}_{A,b,c})| = \frac{rn}{2(\log r)^2} |\mathcal{F}_{A,b,c}|.$$

3. Joint generation and dualization. As we have indicated, maximal infeasible vectors play a crucial role in our analysis. More precisely, for $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$, we shall use the equivalence

$$\mathcal{X} = \mathcal{F}_{A,b,c} \Longleftrightarrow \mathcal{X}^+ \cup \mathcal{I}(\mathcal{F}_{A,b,c})^- = \mathcal{C}$$

to verify the "completeness" of $\mathcal{X}$. In other words, our generation of minimal feasible solutions will in fact involve all maximal infeasible solutions as well. For this reason, we shall consider the *joint generation* of minimal feasible and maximal infeasible vectors by repeatedly solving the following problem.

GEN$(A, b, c, \mathcal{X}, \mathcal{Y})$. *Given a monotone system* (1.1) *defined by* (A, b, c) *and subsets* $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$ *and* $\mathcal{Y} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$*, either find a new integral vector* $x \in (\mathcal{F}_{A,b,c} \cup \mathcal{I}(\mathcal{F}_{A,b,c})) \setminus (\mathcal{X} \cup \mathcal{Y})$ *or show that no such vector exists; i.e.,* $\mathcal{X} = \mathcal{F}_{A,b,c}$ *and* $\mathcal{Y} = \mathcal{I}(\mathcal{F}_{A,b,c})$.

Extending the results of [3] and [17] from the Boolean case $\mathcal{C} = \{0, 1\}^n$ to arbitrary integer boxes, we show that, for a monotone system, GEN$(A, b, c, \mathcal{X}, \mathcal{Y})$ is polynomially equivalent to the following *dualization* problem.

DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$. *Given a family of vectors* $\mathcal{A} \subseteq \mathcal{C}$ *and a subset* $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ *of its maximal independent vectors, either find a new maximal independent vector* $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$ *or prove that no such vector exists; i.e.,* $\mathcal{B} = \mathcal{I}(\mathcal{A})$.

More specifically, we present a simple algorithm which, given a proper subset $\mathcal{Z}$ of the (disjoint) union $\mathcal{F}_{A,b,c} \cup \mathcal{I}(\mathcal{F}_{A,b,c})$, finds a new vector in the union by performing $poly(n, r)$ comparisons and operations $+, -, \times, /, \lfloor\ \rfloor$ and by solving, if necessary, problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ for $\mathcal{A} = \mathcal{Z} \cap \mathcal{F}_{A,b,c}$ and $\mathcal{B} = \mathcal{Z} \cap \mathcal{I}(\mathcal{F}_{A,b,c})$.

Next, by invoking Theorem 2.1, which we shall prove separately in section 5, we can argue that, since the number of maximal infeasible integer vectors for (1.1) is relatively small, we can efficiently generate all elements of $\mathcal{F}_{A,b,c}$ by generating all vectors in the union $\mathcal{F}_{A,b,c} \cup \mathcal{I}(\mathcal{F}_{A,b,c})$ and discarding the elements belonging to $\mathcal{I}(\mathcal{F}_{A,b,c})$. This leads to the following result.

THEOREM 3.1. *Problem GEN$(A, b, c, \mathcal{X})$ can be reduced in strongly polynomial time to DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$.*

To complete the proof of Theorem 1.1, we need to show that DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$, the *dualization problem over integer boxes*, can indeed be solved efficiently. As mentioned earlier, for $\mathcal{C} = \{0, 1\}^n$, problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ turns into the well-known hypergraph dualization problem. In section 7, we extend the hypergraph dualization algorithms of [14] to problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ and show that the latter problem can be solved in quasi-polynomial time.

THEOREM 3.2. *Given two sets* $\mathcal{A}$ *and* $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ *in an* n*-dimensional box* $\mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$*, problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ can be solved in* $poly(n) + m^{o(\log m)}$ *time, where* $m = |\mathcal{A}| + |\mathcal{B}|$.

As before, Theorem 3.2 is stated in the unit-cost model of computation: the bound of the theorem applies to the number of comparisons between components of $\mathcal{A}$ and $\mathcal{B}$ required to solve the dualization problem. Other applications of the dualization problem on boxes can be found in [2, 10, 25].

Clearly, Theorem 1.1 follows from Theorems 3.1 and 3.2. The special cases of Theorems 2.1 and 3.1 for Boolean systems can be found in [7].

4. Bounded number of inequalities. Even though generating all maximal infeasible vectors for (1.1) is NP-hard by Proposition 2.2, this problem can be solved efficiently if the number of inequalities in (1.1) is a fixed constant. Specifically, for $r = \text{const}$, $|\mathcal{F}_{A,b,c}|$ can be bounded by a polynomial in n and $|\mathcal{I}(\mathcal{F}_{A,b,c})|$, and, consequently, all elements of $\mathcal{I}(\mathcal{F}_{A,b,c})$ can be generated in quasi-polynomial time. In fact, for $r = \text{const}$, the problem of generating $\mathcal{I}(\mathcal{F}_{A,b,c})$ as well as the problem of generating $\mathcal{F}_{A,b,c}$ can be solved separately in incremental polynomial time.

For a subset $\mathcal{Y} \subseteq \mathcal{C}$, let $\mathcal{I}^{-1}(\mathcal{Y})$ denote the set of all minimal integral vectors of the ideal $\mathcal{C} \setminus \mathcal{Y}^-$.

THEOREM 4.1. *Suppose that the monotone system* (1.1) *is nontrivial; i.e.,* $0 \notin \mathcal{F}_{A,b,c}$. *Then, for any nonempty subset* $\mathcal{Y} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$, *we have*

$$\left|\mathcal{I}^{-1}(\mathcal{Y}) \cap \mathcal{F}_{A,b,c}\right| \leq \left(n|\mathcal{Y}|\right)^r. \tag{4.1}$$

In particular, for $\mathcal{Y} = \mathcal{I}(\mathcal{F}_{A,b,c})$, *we get*

$$|\mathcal{F}_{A,b,c}| \leq \left(n|\mathcal{I}(\mathcal{F}_{A,b,c})|\right)^r.$$

Generalizing the results of [20, 30] for knapsack problems, we can show that both minimal feasible and maximal infeasible solutions can be generated efficiently if the number of inequalities in the input monotone system is fixed.

THEOREM 4.2. *If the number of inequalities in* (1.1) *is bounded, then both* $\mathcal{F}_{A,b,c}$ *and* $\mathcal{I}(\mathcal{F}_{A,b,c})$ *can be generated in incremental polynomial time.*

The remainder of the paper consists of the proofs of Theorems 2.1, 3.1, 3.2, 4.1, and 4.2 in sections 5, 6, 7, 8, and 9, respectively.

5. Proof of Theorem 2.1. We first need some notation and definitions.

Let $\mathcal{C} = \{x \in \mathbb{Z}^n \mid 0 \leq x \leq c\}$ be a box, and let $f : \mathcal{C} \to \{0,1\}$ be a discrete binary function. The function f is called *monotone* if $f(x) \geq f(y)$ whenever $x \geq y$ and $x, y \in \mathcal{C}$. We denote by $T(f)$ and $F(f)$ the sets of all true and all false vectors of f; i.e.,

$$T(f) = \{x \in \mathcal{C} | f(x) = 1\} = (\text{MIN}[f])^+, \qquad F(f) = \{x \in \mathcal{C} | f(x) = 0\} = (\text{MAX}[f])^-,$$

where MIN$[f]$ and MAX$[f]$ are the sets of all minimal true and all maximal false vectors of f, respectively.

Let $\sigma \in \mathbb{S}_n$ be a permutation of the coordinates, and let x, y be two n-vectors. We say that y is a *left-shift* of x and write $y \succeq_\sigma x$ if the inequalities

$$\sum_{j=1}^{k} y_{\sigma_j} \geq \sum_{j=1}^{k} x_{\sigma_j}$$

hold for all $k = 1, \ldots, n$. A discrete binary function $f : \mathcal{C} \to \{0,1\}$ is called 2-*monotonic with respect to* σ if $f(y) \geq f(x)$ whenever $y \succeq_\sigma x$ and $x, y \in \mathcal{C}$. Clearly, $y \geq x$ implies $y \succeq_\sigma x$ for any $\sigma \in \mathbb{S}_n$ so that any 2-monotonic function is monotone.

The function f will be called *regular* if it is 2-monotonic with respect to the identity permutation $\sigma = (1, 2, \dots, n)$. Any 2-monotonic function can be transformed into a regular one by appropriately reindexing its variables. To simplify notation, we shall state Lemma 5.1 below for regular functions; i.e., we fix $\sigma = (1, 2, \dots, n)$ in this lemma.

For a given subset $\mathcal{A} \subseteq \mathcal{C}$, let us denote by $\mathcal{A}^*$ all of the vectors which are left-shifts of some vectors of $\mathcal{A}$; i.e., $\mathcal{A}^* = \{y \in \mathcal{C} \mid y \succeq x \text{ for some } x \in \mathcal{A}\}$. Clearly, $T(f) = (\mathrm{MIN}[f])^*$ for a regular function f. (In fact, the subfamily of *right-most* vectors of $\mathrm{MIN}[f]$ would be enough to use here.)

Given monotone discrete functions f and g, we call g a *regular majorant* of f if $g(x) \geq f(x)$ for all $x \in \mathcal{C}$ and g is regular. Clearly, $T(g) \supseteq (\mathrm{MIN}[f])^*$ must hold in this case, and the discrete function h defined by $T(h) = (\mathrm{MIN}[f])^*$ is the unique minimal regular majorant of f.

For a vector $x \in \mathcal{C}$ and for an index $1 \leq k \leq n$, let the vectors $x(k]$ and $x[k)$ be defined by

$$x_j(k] = \begin{cases} x_j & \text{for } j \leq k, \\ 0 & \text{otherwise} \end{cases}$$

and

$$x_j[k) = \begin{cases} x_j & \text{for } j \geq k, \\ 0 & \text{otherwise.} \end{cases}$$

Let us denote by $\mathbf{e}^j$ the jth unit vector in $\mathbb{R}^n$ for $j = 1, \dots, n$, and let $p(x)$ denote the number of positive components of the vector $x \in \mathcal{C}$.

LEMMA 5.1. *Given a monotone discrete binary function $f : \mathcal{C} \to \{0, 1\}$ such that $f \not\equiv 0$ and a regular majorant $g \geq f$, we have the inequality*

$$|F(g) \cap \mathrm{MAX}[f]| \leq \sum_{x \in \mathrm{MIN}[f]} p(x). \tag{5.1}$$

Proof. Let us denote by h the unique minimal regular majorant of f. Then we have $F(g) \cap \mathrm{MAX}[f] \subseteq F(h) \cap \mathrm{MAX}[f]$, and hence it is enough to show the statement for $g = h$, i.e., when $T(g) = (\mathrm{MIN}[f])^*$.

For a vector $y \in \mathcal{C} \setminus \{c\}$, let us denote by $l = l_y$ the index of the last component which is less than c_l; i.e., $l = \max\{j \mid y_j < c_j\} \in \{1, \dots, n\}$. We claim that, for every $y \in F(h) \cap \mathrm{MAX}[f]$, there exists an $x \in \mathrm{MIN}[f]$ such that

$$y = x(l-1] + (x_l - 1)\mathbf{e}^l + c[l+1), \tag{5.2}$$

where $l = l_y$. To see this claim, first observe that $y \neq c$ because $y \in F(f)$ and $f \not\equiv 0$. Second, for any j with $y_j < c_j$, we have $y + \mathbf{e}^j \in T(f)$ by the definition of a maximal false point. Hence there exists a minimal true vector $x \in \mathrm{MIN}[f]$ such that $x \leq y + \mathbf{e}^l$ for $l = l_y$. We must have $x(l-1] = y(l-1]$ since, if $x_i < y_i$ for some $i < l$, then $y \geq x + \mathbf{e}^i - \mathbf{e}^l \succeq x$ would hold, i.e., $y \succeq x$ would follow, implying $y \in (\mathrm{MIN}[f])^*$ and yielding a contradiction with $y \in F(h) = \mathcal{C} \setminus (\mathrm{MIN}[f])^*$. Finally, the definition of $l = l_y$ implies that $y[l+1) = c[l+1)$. Hence our claim and the equality (5.2) follow.

The above claim implies that

$$F(h) \cap \mathrm{MAX}[f] \subseteq \{x(l-1] + (x_l - 1)\mathbf{e}^l + c[l+1) \mid x \in \mathrm{MIN}[f],\ x_l > 0\},$$

and hence (5.1) and thus the lemma follow. □

LEMMA 5.2. *Let $f : \mathcal{C} \to \{0,1\}$ be a monotone discrete binary function such that $f \not\equiv 0$ and*

$$x \in T(f) \quad \Rightarrow \quad \alpha x \stackrel{\text{def}}{=} \alpha_1 x_1 + \cdots \alpha_n x_n \geq \beta, \tag{5.3}$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a given real vector and β is a real threshold. Then

$$|\{x \in \mathcal{C} \mid \alpha x < \beta\} \cap \mathrm{MAX}[f]| \leq \sum_{x \in \mathrm{MIN}[f]} p(x).$$

Proof. Suppose that some of the weights $\alpha_1, \ldots, \alpha_n$ are negative, say, $\alpha_1 < 0, \ldots, \alpha_k < 0$, and $\alpha[k+1) \geq 0$. Since $\alpha x \geq \beta$ for any $x \in T(f)$ and since f is monotone, we have $x \in T(f) \ \Rightarrow \alpha[k+1)x \geq \beta - \alpha(k]c(k]$. By the negativity of the weights $\alpha_1, \ldots, \alpha_k$, we also have $\{x \in \mathcal{C} \mid \alpha x < \beta\} \subseteq \{x \in \mathcal{C} \mid \alpha[k+1)x < \beta - \alpha(k]c(k]\}$. Hence it suffices to prove the lemma for the nonnegative weight vector $\alpha[k+1)$ and the threshold $\beta - \alpha(k]c(k]$. In other words, we can assume without loss of generality that the original weight vector α is nonnegative.

Let $\sigma \in \mathbb{S}_n$ be a permutation such that $\alpha_{\sigma_1} \geq \alpha_{\sigma_2} \geq \cdots \geq \alpha_{\sigma_n} \geq 0$. Then the threshold function

$$g(x) = \begin{cases} 1 & \text{if } \alpha x \geq \beta, \\ 0 & \text{otherwise} \end{cases}$$

is 2-monotonic with respect to σ. By (5.3), we have $g \geq f$ for all $x \in \mathcal{C}$; i.e., g is a majorant of f. In addition, $F(g) = \{x \in \mathcal{C} \mid \alpha x < \beta\}$, and hence Lemma 5.2 follows from Lemma 5.1. □

Proof of Theorem 2.1. We are now ready to show inequality (2.3) and finish the proof of Theorem 2.1. For this, we shall prove, in fact, the following stronger inequality:

$$|\mathcal{I}(\mathcal{X}) \cap \mathcal{I}(\mathcal{F}_{A,b,c})| \leq r \sum_{x \in \mathcal{X}} p(x). \tag{5.4}$$

Given a nonempty set $\mathcal{X} \subseteq \mathcal{F}_{A,b,c}$, consider the monotone discrete function $f : \mathcal{C} \to \{0,1\}$ defined by the condition $\mathrm{MIN}[f] = \mathcal{X}$. Since (1.1) is monotone, any true vector of f also satisfies (1.1):

$$x \in T(f) \quad \Rightarrow a_{k1}x_1 + \cdots + a_{kn}x_n \geq b_k$$

for all $k = 1, \ldots, r$. In addition, $f \not\equiv 0$ because $\mathcal{X} \neq \emptyset$. Thus, by Lemma 5.2, we have the inequalities

$$|\{x \mid a_{k1}x_1 + \cdots + a_{kn}x_n < b_k\} \cap \mathrm{MAX}[f]| \leq \sum_{x \in \mathcal{X}} p(x) \tag{5.5}$$

for each $k = 1, \ldots, r$. Now, from $\mathrm{MAX}[f] = \mathcal{I}(\mathcal{X})$, we deduce that

$$\mathcal{I}(\mathcal{F}_{A,b,c}) \cap \mathcal{I}(\mathcal{X}) \subseteq \bigcup_{k=1}^{r} \{x \mid a_{k1}x_1 + \cdots + a_{kn}x_n < b_k\} \cap \mathrm{MAX}[f],$$

and hence (5.4) follows by (5.5), thus implying (2.3) and completing the proof of the theorem. □

6. Generating minimal feasible solutions. As mentioned in the introduction, the proof of Theorem 3.1 has two ingredients. First, we show that, given a monotone system (1.1), the sets $\mathcal{I}(\mathcal{F}_{A,b,c})$ and $\mathcal{F}_{A,b,c}$ can be jointly enumerated by iteratively solving the dualization problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$. Second, we argue that, due to Theorem 2.1, the number of maximal infeasible vectors for (1.1) is relatively small, and hence the generation of $\mathcal{F}_{A,b,c}$ polynomially reduces to the joint generation of $\mathcal{I}(\mathcal{F}_{A,b,c})$ and $\mathcal{F}_{A,b,c}$.

6.1. Joint generation of dual subsets in an integral box. As was observed in [3, 17], for $c = \mathbf{e}_n$, problem GEN$(A, b, c, \mathcal{A}, \mathcal{B})$ can be reduced in polynomial time to the dualization problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$. Extending this observation, we prove it here for any integral vector c.

PROPOSITION 6.1. *Problem GEN$(A, b, c, \mathcal{A}, \mathcal{B})$ can be solved in poly$(n, |\mathcal{A}|, |\mathcal{B}|) + T^*$ time, where T^* denotes the time required to solve problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$.*

Proof. In fact, we can prove a more general statement. Let us consider an arbitrary *antichain* $\mathcal{F} \subseteq \mathcal{C}$ (i.e., a family $\mathcal{F}$ of vectors such that $x \not\leq y$ for any two distinct elements $x, y \in \mathcal{F}$), assume that there is a polynomial-time membership oracle $\mathfrak{O}(\mathcal{F}^+)$ for the monotone set $\mathcal{F}^+$, and consider the following problem.

GEN$(\mathfrak{O}(\mathcal{F}^+), \mathcal{A}, \mathcal{B})$. *Given subsets $\mathcal{A} \subseteq \mathcal{F}$ and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$, either find a new element $x \in (\mathcal{F} \cup \mathcal{I}(\mathcal{F})) \setminus (\mathcal{A} \cup \mathcal{B})$ or show that no such vector exists, i.e., $\mathcal{A} = \mathcal{F}$ and $\mathcal{B} = \mathcal{I}(\mathcal{F})$.*

We can show that this more general problem also reduces in polynomial time to DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$.

Our proof uses two subroutines, the first of which takes as input a vector $x \in \mathcal{F}^+$ and returns a minimal vector x^* in $\mathcal{F}^+ \cap \{x\}^-$. Such a vector $x^* = \min_{\mathcal{F}}(x)$ can, for instance, be computed by coordinate descent:

$$\begin{array}{rcl} x_1^* & \leftarrow & \min\{y_1 \mid (y_1, y_2, \dots, y_{n-1}, y_n) \in \mathcal{F}^+ \cap \{x\}^-\}, \\ x_2^* & \leftarrow & \min\{y_2 \mid (x_1^*, y_2, \dots, y_{n-1}, y_n) \in \mathcal{F}^+ \cap \{x\}^-\}, \\ & \dots & \\ x_n^* & \leftarrow & \min\{y_n \mid (x_1^*, x_2^*, \dots, x_{n-1}^*, y_n) \in \mathcal{F}^+ \cap \ \{x\}^-\}. \end{array}$$

The second subroutine is to compute, for a given vector $x \in \mathcal{I}(\mathcal{F})^-$, a maximal vector $x^* \in \mathcal{I}(\mathcal{F})^- \cap x^+$. Similarly, this problem can be done by coordinate descent. Note that each of the n coordinate steps in the above procedures can be reduced via binary search to at most $\log(\|c\|_\infty + 1)$ membership queries for the monotone family $\mathcal{F}^+$. Though this bound depends on the size of the box $\mathcal{C}$, in general, in our case, when $\mathcal{F} = \mathcal{F}_{A,b,c}$ is the set of minimal integer solutions for an explicitly given monotone system (1.1), both of the above coordinate descents can be clearly performed in $O(nr)$ comparisons $(\leq, \geq)$ and arithmetic operations $(+, -, \times, /, \lfloor\ \rfloor)$, regardless of the box size.

Using these subroutines, our proof of Proposition 6.1 can now be completed by the following algorithm.

Algorithm $\mathcal{J}$.

Input: An oracle $\mathfrak{O}(\mathcal{F}^+)$ and subsets $\mathcal{A} \subseteq \mathcal{F}$ and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$.

Step 1. Check whether $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$. Since $\mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$ and $\mathcal{A} \subseteq \mathcal{F}$, each vector $x \in \mathcal{B}$ is independent of $\mathcal{A}$, and we have only to check the maximality of x for $\mathcal{I}(\mathcal{A})$. In other words, we have to check whether or not $x + \mathbf{e}^j \geq y$ for some unit vector $\mathbf{e}^j$, $j \in \{1, \dots, n\}$, and some vector $y \in \mathcal{A}$. Since both $\mathcal{A}$ and $\mathcal{B}$ are explicitly given, this can be done in *poly*$(n, |\mathcal{A}|, |\mathcal{B}|)$ comparisons. If there is an $x \in \mathcal{B} \setminus \mathcal{I}(\mathcal{A})$, then $x \notin \mathcal{F}^+$ because $x \in \mathcal{B} \subseteq \mathcal{I}(\mathcal{F})$. This and the inclusion $\mathcal{A} \subseteq \mathcal{F}$ imply that $x \notin \mathcal{A}^+$. Since

$x \notin \mathcal{I}(\mathcal{A})$, we can find a coordinate $j \in \{1, \ldots, n\}$ for which $y = x + \mathbf{e}^j \notin \mathcal{A}^+$. By the maximality of x in $\mathcal{C} \setminus \mathcal{F}^+$, y belongs to $\mathcal{F}^+$. Now, using the first subroutine and letting $y^* = \min_{\mathcal{F}}(y)$, we can conclude that $y^* \in \mathcal{F} \setminus \mathcal{A}$; i.e., y^* is a new minimal integral vector in $\mathcal{F}$. Otherwise, if $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, we continue with the next step.

Step 2. Similar to the previous step, we check whether $\mathcal{A} \subseteq \mathcal{I}^{-1}(\mathcal{B})$, where $\mathcal{I}^{-1}(\mathcal{B})$ denotes the set of integral vectors which are minimal in $\mathcal{C} \setminus \mathcal{B}^-$. If $\mathcal{A}$ contains an element that is not minimal in $\mathcal{C} \setminus \mathcal{B}^-$, we can find a new vector in $\mathcal{I}(\mathcal{F}) \setminus \mathcal{B}$ and stop. Otherwise, we continue with the next step.

Step 3. In this case, we have $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ and $\mathcal{A} \subseteq \mathcal{I}^{-1}(\mathcal{B})$, and thus the following equivalence holds:

$$(\mathcal{A}, \mathcal{B}) = (\mathcal{F}, \mathcal{I}(\mathcal{F})) \quad \Leftrightarrow \quad \mathcal{B} = \mathcal{I}(\mathcal{A}).$$

To see this, assume that $\mathcal{B} = \mathcal{I}(\mathcal{A})$, and suppose, on the contrary, that there is an $x \in \mathcal{F} \setminus \mathcal{A}$. Since $x \notin \mathcal{A} = \mathcal{I}^{-1}(\mathcal{B})$ and $x \notin \mathcal{B}^- \subseteq \mathcal{I}(\mathcal{F})^-$, there must exist a $y \in \mathcal{I}^{-1}(\mathcal{B}) = \mathcal{A} \subseteq \mathcal{F}$ such that $y \leq x$. Hence we get two distinct elements $x, y \in \mathcal{F}$ such that $y \leq x$, which contradicts the definition of $\mathcal{F}$. The existence of an $x \in \mathcal{I}(\mathcal{F}) \setminus \mathcal{B}$ leads to a similar contradiction.

To check the stopping criterion $\mathcal{B} = \mathcal{I}(\mathcal{A})$, we solve problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$. If $\mathcal{B} \neq \mathcal{I}(\mathcal{A})$, we obtain a new point $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$. By (2.2), either $x \in \mathcal{F}^+$ or $x \in \mathcal{I}(\mathcal{F})^-$, and we can decide which of these two cases holds by asking the oracle $\mathfrak{O}(\mathcal{F}^+)$ or, in our special case, by checking the feasibility of x for (1.1). In the first case, we conclude that $x^* = \min_{\mathcal{F}}(x)$ is a new vector in $\mathcal{F} \setminus \mathcal{A}$. In the second case, we can extend $\mathcal{I}(\mathcal{F}) \setminus \mathcal{B}$ by using the second subroutine to compute a maximal vector in $x^+ \cap \mathcal{I}(\mathcal{F})^-$. □

As we noted, the above procedure can be used for any antichain $\mathcal{F} \subseteq \mathcal{C}$ defined by a polynomial-time membership oracle for $\mathcal{F}^+$ since the coordinate descend subroutines used by the algorithm can always be implemented in $n \log(\|c\|_\infty + 1)$ membership tests. Accordingly, Proposition 6.1 also holds for an arbitrary antichain defined by a polynomial-time membership oracle, provided that the polynomial term in the proposition includes a multiplicative factor of $\log(\|c\|_\infty + 1)$.

6.2. Uniformly dual-bounded antichains. Let $\mathcal{F} \subseteq \mathcal{C}$ be an antichain defined by a polynomial-time membership oracle $\mathfrak{O}(\mathcal{F}^+)$ for $\mathcal{F}^+$. Given an input description $\mathcal{D}$ of $\mathcal{F}$ and a vector $x \in \mathcal{C}$, such an oracle checks the membership of x in $\mathcal{F}^+$ in time bounded by a polynomial in the size of x and the length $|\mathcal{D}|$ of the input description of $\mathcal{F}$. For instance, the antichain $\mathcal{F}_{A,b,c}$ of minimal feasible integer vectors for (1.1) is defined by the triple $\mathcal{D} = (A, b, c)$, and the membership test for a given $x \in \mathcal{C}$ simply checks if $Ax \geq b$. The generation problem can thus be considered more generally for arbitrary antichains.

GEN$(\mathfrak{O}(\mathcal{F}^+), \mathcal{X})$. *Given an antichain $\mathcal{F} \subseteq \mathcal{C}$ defined by a polynomial-time membership oracle $\mathfrak{O}(\mathcal{F}^+)$ and a subset $\mathcal{X} \subseteq \mathcal{F}$, either find a new vector $x \in \mathcal{F} \setminus \mathcal{X}$ or show that no such vector exists; i.e., $\mathcal{X} = \mathcal{F}$.*

Extending the notion of dual-bounded hypergraphs defined in [7], we say that an antichain $\mathcal{F} \subseteq \mathcal{C}$ with an input description $\mathcal{D}$ is *uniformly dual-bounded* if there exists a polynomial $q(x, y)$ such that

$$|\mathcal{I}(\mathcal{F}) \cap \mathcal{I}(\mathcal{X})| \leq q(|\mathcal{D}|, |\mathcal{X}|)$$

for any nonempty subset $\mathcal{X} \subseteq \mathcal{F}$ (see [7] for further details and examples). As we show below, the uniform dual-boundedness of an antichain is a sufficient condition to be able to reduce polynomially generation to dualization.

PROPOSITION 6.2. *Suppose $\mathcal{F}$ is uniformly dual-bounded and defined by a polynomial-time membership oracle $\mathfrak{O}(\mathcal{F}^+)$ for $\mathcal{F}^+$. Then problem GEN$(\mathfrak{O}(\mathcal{F}^+), \mathcal{X})$ is polynomial-time reducible to at most $q(|\mathcal{D}|, |\mathcal{X}|) + 1$ instances of problem DUAL $(\mathcal{C}, \mathcal{A}, \mathcal{B})$.*

Proof. Given a set $\mathcal{X} \subseteq \mathcal{F}$, we repeatedly run Algorithm $\mathcal{J}$, starting with $\mathcal{A} = \mathcal{X}$ and $\mathcal{B} = \emptyset$, until it either produces a new element in $\mathcal{F} \setminus \mathcal{X}$ or proves that $\mathcal{X} = \mathcal{F}$ by generating the entire family $\mathcal{I}(\mathcal{F})$. By Step 1, either $\mathcal{B} \subseteq \mathcal{I}(\mathcal{X})$ is maintained during the execution of the algorithm or a new element $x \in \mathcal{F}$ can be found. Thus, as long as Algorithm $\mathcal{J}$ outputs elements of $\mathcal{I}(\mathcal{F})$, these elements also belong to $\mathcal{I}(\mathcal{X})$, and hence the total number of such elements does not exceed $q(|\mathcal{D}|, |\mathcal{X}|)$. □

Proof of Theorem 3.1. By Theorem 2.1, the antichain $\mathcal{F} = \mathcal{F}_{A,b,c}$ of minimal integer solutions to the monotone system (1.1), for which we have $\mathcal{D} = (A, b, c)$, is uniformly dual-bounded with $q(|\mathcal{D}|, |\mathcal{X}|) = rn|\mathcal{X}|$. Furthermore, the reduction of Proposition 6.2 is strongly polynomial. For this reason, Theorem 3.1 follows from Theorem 2.1 and Proposition 6.2. □

7. Dualization in products of chains. In this section, we prove Theorem 3.2. Let $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ be an integer box defined by the product of n chains $\mathcal{C}_i = [l_i : u_i]$, where $l_i, u_i \in \mathbb{Z}$ are, respectively, the lower and upper bounds of chain $\mathcal{C}_i$. For given antichains $\mathcal{A} \subseteq \mathcal{C}$ and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, we say that $\mathcal{B}$ is *dual to* $\mathcal{A}$ if $\mathcal{B} = \mathcal{I}(\mathcal{A})$; i.e., $\mathcal{B}$ contains all the maximal elements of $\mathcal{C} \setminus \mathcal{A}^+$. If $\mathcal{C}$ is the unit cube, we obtain the familiar notion of dual hypergraphs, where $\mathcal{I}(\mathcal{A})$ corresponds to the family of complementary sets of the minimal transversals of $\mathcal{A}$. In the following two subsections, we will show how to extend the hypergraph dualization algorithms of [14] to arbitrary antichains $\mathcal{A}$ of integral vectors in a box $\mathcal{C}$. Note that, by (2.2), our problem can be stated as of checking whether $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$, and, if not, find an element $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$.

As in [14], we shall analyze the running time of the algorithms in terms of the *volume* $v = v(\mathcal{A}, \mathcal{B}) \stackrel{\text{def}}{=} |\mathcal{A}||\mathcal{B}|$ of the input problem. In general, a given problem will be decomposed into a number of subproblems of smaller volume, which we will solve recursively. Since we have assumed that $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$, (2.2) implies that the following condition holds for the original problem and all subsequent subproblems:

$$a \not\leq b \quad \text{for all} \quad a \in \mathcal{A}, b \in \mathcal{B}. \tag{7.1}$$

Let $C(v) = C(v(\mathcal{A}, \mathcal{B}))$ denote the number of subproblems that have to be solved in order to solve the original problem (the maximum number of recursive calls on a problem of volume $\leq v$), and let $m \stackrel{\text{def}}{=} |\mathcal{A}| + |\mathcal{B}|$ and $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. We start with the following proposition that provides the base case for recursion.

PROPOSITION 7.1. *Suppose* $\min\{|\mathcal{A}|, |\mathcal{B}|\} \leq \text{const}$*; then problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ is solvable in time poly(n, m).*

Proof. Let us assume without loss of generality that $\mathcal{B} = \{b^1, \ldots, b^k\}$ for some constant k. Then $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$ *if and only if*, for every $t = (t_1, \ldots, t_k) \in [n]^k$ for which

$$b^j_{t_j} \neq u_{t_j} \quad \text{for all} \quad j \in [k], \tag{7.2}$$

there exists an $a \in \mathcal{A}$ such that

$$a_i \leq \max(\{b^j_i + 1 \mid j \in [k], t_j = i\} \cup \{l_i\}) \quad \text{for all } i \in [n]. \tag{7.3}$$

To see this, assume first that $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$, and consider any $t \in [n]^k$ such that (7.2) holds. Let $x \in \mathcal{C}$ be defined by taking $x_i = \max(\{b_i^j + 1 \mid j \in [k],\ t_j = i\} \cup \{l_i\})$ for $i = 1, \ldots, n$. Then $x \in \mathcal{C} \setminus \mathcal{B}^-$, and hence $x \in \mathcal{A}^+$, implying that there is an $a \in \mathcal{A}$ satisfying (7.3). On the other hand, let us assume that, for every $t \in [n]^k$ satisfying (7.2), there is an $a \in \mathcal{A}$ for which (7.3) holds. Consider an arbitrary $x \in \mathcal{C} \setminus \mathcal{B}^-$. Then there must exist an index $t_j \in [n]$ for every $j \in [k]$ such that $x_{t_j} \geq b_{t_j}^j + 1$. The vector $t = (t_1, \ldots, t_k) \in [n]^k$ constructed in this way then satisfies (7.2), and, therefore, there is an $a \in \mathcal{A}$ such that $a_i \leq \max(\{b_i^j + 1 \mid j \in [k],\ t_j = i\} \cup \{l_i\}) \leq x_i$ for all $i = 1, \ldots, n$, thus implying $x \in \mathcal{A}^+$.

The condition given in (7.2) and (7.3) can clearly be checked in $poly(n, m)$ time for any constant k. If the condition does not hold for some $t \in [n]^k$, then the element x, defined by $x_i = \max(\{b_i^j + 1 \mid j \in [k],\ t_j = i\} \cup \{l_i\})$ for $i = 1, \ldots, n$, belongs to $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$. □

Note that, to complete the solution of the dualization problem, we need to find an element *maximal* in $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$. This can easily be done in polynomial time and even independently of the chain sizes, as shown in the next proposition. Therefore, in the following subsections, we shall only show how to obtain an element $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$, when such an element exists.

PROPOSITION 7.2. *Given an $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$, it can be extended to a maximal element with the same property in $O(nm)$ time.*

Proof. Note that, for $i = 1, \ldots, n$, the ith component of any maximal element in $\mathcal{C} \setminus \mathcal{A}^+$ must belong to the set $\{a_i - 1 \mid a \in \mathcal{A}\} \cup \{u_i\}$. Thus a new element $x' \geq x$ maximal in $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$ can be found as follows. For $i = 1, \ldots, n$, we find iteratively the set $\mathcal{A}_i \stackrel{\text{def}}{=} \{a \in \mathcal{A} \mid a_1 \leq x'_1, \ldots, a_{i-1} \leq x'_{i-1}, a_i > x_i, a_{i+1} \leq x_{i+1}, \ldots, a_n \leq x_n\}$ and then set $x'_i \leftarrow \min(\{a_i - 1 \mid a \in \mathcal{A}_i\} \cup \{u_i\})$. □

7.1. Algorithm A. Assume that $\mathcal{A}, \mathcal{B}$ satisfy (7.1), and let $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$, where $\mathcal{C}_i = [l_i : u_i]$. We say that a coordinate $i \in [n]$ is *essential* for a point $a \in \mathcal{A}$ ($b \in \mathcal{B}$) if $a_i > l_i$ (respectively, $b_i < u_i$). Let us denote by $E(x)$ the set of essential coordinates of a point $x \in \mathcal{A} \cup \mathcal{B}$. The following lemma generalizes a well-known fact for dual Boolean functions (cf. [14]).

LEMMA 7.3. *Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$ be given subsets. Then either* (i) *there exists an element $y \in \mathcal{A} \cup \mathcal{B}$ with few essential coordinates $|E(y)| \leq \log m$, where $m = |\mathcal{A}| + |\mathcal{B}|$, or* (ii) *if no such element y exists, then $\mathcal{B} \neq \mathcal{I}(\mathcal{A})$, and an element $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$ can be found in $poly(n, m)$ time.*

Proof. Let $z \in \mathcal{C}$ be the vector obtained by picking each coordinate z_i randomly from $\{l_i, u_i\}$, $i = 1, \ldots, n$, and consider the random variable $N(z) \stackrel{\text{def}}{=} |\{a \in \mathcal{A} \mid z \geq a\}| + |\{b \in \mathcal{B} \mid z \leq b\}|$. Then the expected value of $N(z)$ is given by

$$\begin{aligned} \mathbb{E}[N(z)] &= \sum_{a \in \mathcal{A}} \Pr\{z \geq a\} + \sum_{b \in \mathcal{B}} \Pr\{z \leq b\} \\ &= \sum_{a \in \mathcal{A}} \prod_{i \in E(a)} \Pr\{z_i = u_i\} + \sum_{b \in \mathcal{B}} \prod_{i \in E(b)} \Pr\{z_i = l_i\} \\ &= \sum_{a \in \mathcal{A}} \left(\frac{1}{2}\right)^{|E(a)|} + \sum_{b \in \mathcal{B}} \left(\frac{1}{2}\right)^{|E(b)|}. \end{aligned} \tag{7.4}$$

If we have $\mathbb{E}[N(z)] \geq 1$, then, by letting $r = \min\{|E(z)| \ :\ z \in \mathcal{A} \cup \mathcal{B}\}$, we get by (7.4) that

$$1 \leq \mathbb{E}[N(z)] \leq (|\mathcal{A}| + |\mathcal{B}|) \left(\frac{1}{2}\right)^r = m \left(\frac{1}{2}\right)^r,$$

from which part (i) of the lemma follows.

On the other hand, if $\mathbb{E}[N(z)] < 1$, then we can find an element $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$ (in fact a corner of the finite box $\mathcal{C}$) in polynomial time, proving part (ii) of the lemma. To see this, let us consider, for each $i = 1, \ldots, n$, the variables $z_j \in \{l_j, u_j\}$ random for $j > i$, as above, compute the expectations of $N(x_1, \ldots, x_{i-1}, l_i, z_{i+1}, \ldots, z_n)$ and $N(x_1, \ldots, x_{i-1}, u_i, z_{i+1}, \ldots, z_n)$ analogously to (7.4), and select the value for $x_i \in \{l_i, u_i\}$ so as to minimize the corresponding expectation. Clearly, $N(x) < 1$ will hold in this case; thus $N(x) = 0$, implying that $x \notin \mathcal{A}^+ \cup \mathcal{B}^-$. □

Next we show that, for any dual pair $(\mathcal{A}, \mathcal{B})$, an essential coordinate with high frequency exists for either $\mathcal{A}$ or $\mathcal{B}$.

LEMMA 7.4. *Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$ be a pair of dual subsets, $\mathcal{B} = \mathcal{I}(\mathcal{A})$, for which $|\mathcal{A}||\mathcal{B}| \geq 1$. Then there exists a coordinate $i \in [n]$, which is essential in $\mathcal{A}$ or in $\mathcal{B}$ with frequency at least $1/\log m$, where $m = |\mathcal{A}| + |\mathcal{B}|$.*

Proof. By Lemma 7.3, the set $\mathcal{A} \cup \mathcal{B}$ must contain an element y with a logarithmically small number of essential coordinates. Suppose, without loss of generality, that $y \in \mathcal{A}$. From our assumptions and condition (7.1), we know that, for every $b \in \mathcal{B}$, there is an $i \in E(b) \cap E(y)$ such that $b_i < y_i$. Letting $\mathcal{B}_i^y \stackrel{\text{def}}{=} \{b \in \mathcal{B} \mid b_i < y_i\}$ for $i \in E(y)$, we conclude that

$$|\mathcal{B}| = \left| \bigcup_{i \in E(y)} \mathcal{B}_i^y \right| \leq \sum_{i \in E(y)} |\mathcal{B}_i^y|,$$

and therefore there is an $i \in [n]$ which is essential for at least $|\mathcal{B}|/|E(y)| \geq |\mathcal{B}|/\log m$ many elements of $\mathcal{B}$. □

We are now ready to state the first dualization algorithm. Given an integral box $\mathcal{C}$ and subsets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$ that satisfy (7.1), in time $poly(n, m) + m^{O(\log^2 m)}$ the algorithm will either prove that $\mathcal{C} = \mathcal{A}^+ \cup \mathcal{B}^-$ or find an $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$.

Step 1. If $|\mathcal{A}||\mathcal{B}| \leq 1$, then the dualization problem can be solved in $O(n)$ time.

Step 2. For each $k \in [n]$, if $a_k < l_k$ for some $a \in \mathcal{A}$ ($b_k > u_k$ for some $b \in \mathcal{B}$), set $a_k \leftarrow l_k$ (respectively, set $b_k \leftarrow u_k$). (Note that $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$ holds initially but might not hold after decomposing $\mathcal{C}$; see Step 4 below.) Note that condition (7.1) continues to hold after such replacements. Thus, in $O(nm)$ time, we can ensure that $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$ hold.

Step 3. Check if there is a $y \in \mathcal{A} \cup \mathcal{B}$ with at most $\log m$ essential coordinates. If no such y can be found, a new point in $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$ can be obtained as described in the proof of Lemma 7.3 (ii). Otherwise, assume, without loss of generality, that $y = a^o \in \mathcal{A}$, and find an $i \in E(a^o)$ for which $|\{b \in \mathcal{B} \mid b_i < a_i^o\}| \geq |\mathcal{B}|/\log m$. Let us suppose, again without any loss of generality, that $i = 1$ and set $\mathcal{C}_1' \leftarrow [a_1^o : u_1]$, $\mathcal{C}_1'' \leftarrow [l_1 : a_1^o - 1]$. Let, further,

$$\mathcal{A}'' = \{a \in \mathcal{A} \mid a_1 < a_1^o\}, \qquad \mathcal{B}' = \{b \in \mathcal{B} \mid b_1 \geq a_1^o\}.$$

Observe that $|\mathcal{A}''| \leq |\mathcal{A}| - 1$ since $a^o \notin \mathcal{A}''$ and that $|\mathcal{B}'| \leq (1 - 1/\log m)|\mathcal{B}|$ by our choice of a^o and i.

Step 4. Denoting by $\mathcal{C}' = \mathcal{C}_1' \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_n$ and $\mathcal{C}'' = \mathcal{C}_1'' \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_n$ the two subboxes of $\mathcal{C}$ induced by the above partitioning, it is then easy to see that $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$ *if and only if*

$$\mathcal{A}, \mathcal{B}' \text{ are dual in } \mathcal{C}' \tag{7.5}$$

and

$$\mathcal{A}'', \mathcal{B} \text{ are dual in } \mathcal{C}''. \tag{7.6}$$

Thus, by applying the algorithm recursively to these two subproblems, we reduce the computation on a problem of size $v = |\mathcal{A}||\mathcal{B}|$ to computing the solution for two subproblems (7.5)–(7.6) of volumes

$$\begin{aligned} v(\mathcal{A}, \mathcal{B}') &= |\mathcal{A}||\mathcal{B}'| \leq |\mathcal{A}|(1-\epsilon)|\mathcal{B}| = (1-\epsilon)v \quad \text{and} \\ v(\mathcal{A}'', \mathcal{B}) &= |\mathcal{A}''||\mathcal{B}| = (|\mathcal{A}|-1)|\mathcal{B}| \leq v-1, \end{aligned}$$

where $\epsilon = 1/\log m$. This leads to the recurrence

$$C(v) \leq 1 + C((1-\epsilon)v) + C(v-1),$$

which was shown in [14] to evaluate to $C(v) \leq (3+2v\epsilon)^{\log v/\epsilon}$, implying that the running time of the algorithm is $poly(n) + m^{O(\log^2 m)}$.

In the next subsection, we shall give an algorithm that solves the problem in $poly(n,m) + m^{o(\log m)}$ time.

7.2. Algorithm B. Algorithm A of the previous subsection decomposes problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ into two subproblems, (7.5) and (7.6). As we shall see below, subproblems (7.5) and (7.6) are not independent, and we can utilize their dependence to get a more efficient dualization algorithm. Given an integral box $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$, where $\mathcal{C}_i = [l_i : u_i]$, and subsets of integral vectors $\mathcal{A}, \mathcal{B}$ that satisfy the necessary condition (7.1), we proceed as follows.

Step 1. If $\min\{|\mathcal{A}|, |\mathcal{B}|\} \leq 2$, the duality of $\mathcal{A}$ and $\mathcal{B}$ can be tested in $O(n^3 m)$ using Proposition 7.1.

Step 2. For each $k \in [n]$,

2.1. if $a_k > u_k$ for some $a \in \mathcal{A}$ ($b_k < l_k$ for some $b \in \mathcal{B}$), then a (respectively, b) can clearly be discarded from further consideration;

2.2. if $a_k < l_k$ for some $a \in \mathcal{A}$ ($b_k > u_k$ for some $b \in \mathcal{B}$), we set $a_k \leftarrow l_k$ (respectively, $b_k \leftarrow u_k$).

Thus we may assume for the next steps that $\mathcal{A}, \mathcal{B} \subseteq \mathcal{C}$.

Step 3. Let $a^o \in \mathcal{A}$, $b^o \in \mathcal{B}$. By (7.1), there exists an $i \in [n]$ such that $a^o_i > b^o_i$. Let us assume, without any loss of generality, that $i = 1$ and set $\mathcal{C}'_1 \leftarrow [a^o_1 : u_1]$, $\mathcal{C}''_1 \leftarrow [l_1 : a^o_1 - 1]$. (Alternatively, we may set $\mathcal{C}''_1 \leftarrow [l_1 : b^o_1]$ and $\mathcal{C}'_1 \leftarrow [b^o_1 + 1 : u_1]$.) Let $\mathcal{C}'$ and $\mathcal{C}''$ be the two resulting subboxes as defined in Step 4 of the previous subsection. Define, further,

$$\begin{aligned} \mathcal{A}'' &= \{a \in \mathcal{A} \mid a_1 < a^o_1\}, \qquad \mathcal{A}' = \mathcal{A} \setminus \mathcal{A}'', \qquad \epsilon_1^{\mathcal{A}} = \frac{|\mathcal{A}'|}{|\mathcal{A}|}, \\ \mathcal{B}' &= \{b \in \mathcal{B} \mid b_1 \geq a^o_1\}, \qquad \mathcal{B}'' = \mathcal{B} \setminus \mathcal{B}', \qquad \epsilon_1^{\mathcal{B}} = \frac{|\mathcal{B}''|}{|\mathcal{B}|}. \end{aligned}$$

Observe that $\epsilon_1^{\mathcal{A}} > 0$ and $\epsilon_1^{\mathcal{B}} > 0$ since $a^o \in \mathcal{A}'$ and $b^o \in \mathcal{B}''$.

Step 4. Define

$$\epsilon(v) = 1/\chi(v), \qquad \text{where } \chi(v)^{\chi(v)} = v = v(\mathcal{A}, \mathcal{B}).$$

If $\min\{\epsilon_1^{\mathcal{A}}, \epsilon_1^{\mathcal{B}}\} > \epsilon(v)$, we use the decomposition rule given above, which amounts to solving recursively two subproblems, (7.5), (7.6), of respective volumes:

$$\begin{aligned} v(\mathcal{A}, \mathcal{B}') &= |\mathcal{A}||\mathcal{B}'| = |\mathcal{A}|(1-\epsilon_1^{\mathcal{B}})|\mathcal{B}| = (1-\epsilon_1^{\mathcal{B}})v(\mathcal{A}, \mathcal{B}), \\ v(\mathcal{A}'', \mathcal{B}) &= |\mathcal{A}''||\mathcal{B}| = (1-\epsilon_1^{\mathcal{A}})|\mathcal{A}||\mathcal{B}| = (1-\epsilon_1^{\mathcal{A}})v(\mathcal{A}, \mathcal{B}). \end{aligned}$$

This gives rise to the recurrence

$$C(v) \le 1 + C((1-\epsilon_1^{\mathcal{B}})v) + C((1-\epsilon_1^{\mathcal{A}})v) \le 1 + 2C((1-\epsilon(v))v). \tag{7.7}$$

Step 5. Let us now suppose that $\epsilon_1^{\mathcal{B}} \le \epsilon(v)$. In this case, we begin by solving subproblem (7.5). If $\mathcal{A}, \mathcal{B}'$ are not dual in $\mathcal{C}'$, we get a point x maximal in $\mathcal{C}' \setminus [\mathcal{A}^+ \cup (\mathcal{B}')^-]$, and we are done. Otherwise, we claim that

$$\mathcal{A}'', \mathcal{B} \text{ are dual in } \mathcal{C}'' \iff \text{for all } a \in \widetilde{\mathcal{A}} : \mathcal{A}'', \mathcal{B}'' \text{ are dual in } \mathcal{C}''(a), \tag{7.8}$$

where $\widetilde{\mathcal{A}} = \{a \in \mathcal{A} \mid a_1 \le a_1^o\}$ and $\mathcal{C}''(a) = \mathcal{C}_1'' \times [a_2 : u_2] \times \cdots \times [a_n : u_n]$.

Proof of (7.8). The forward direction does not use (7.5). Suppose that there is an $a \in \widetilde{\mathcal{A}}$ such that $\mathcal{A}''$ and $\mathcal{B}''$ are not dual in $\mathcal{C}''(a)$; i.e., there is an $x \in \mathcal{C}''(a) \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}'')^-]$. Then $x_i \ge a_i$ for $i = 2, \ldots, n$. If $x \in (\mathcal{B}')^-$, i.e., $x \le b$, for some $b \in \mathcal{B}'$, then, by the definition of $\mathcal{B}'$, $b_1 \ge a_1^o$. On the other hand, $a \in \widetilde{\mathcal{A}}$ implies that $a_1 \le a_1^o$. However, then,

$$(a_1, a_2, \ldots, a_n) \le (a_1^o, x_2, \ldots, x_n) \le (b_1, b_2, \ldots, b_n),$$

which contradicts the assumed condition (7.1). This shows that $x \in \mathcal{C}'' \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}' \cup \mathcal{B}'')^-]$, and hence $\mathcal{A}''$ and $\mathcal{B}$ are not dual in $\mathcal{C}$.

For the other direction, let $x \in \mathcal{C}'' \setminus [(\mathcal{A}'')^+ \cup \mathcal{B}^-]$. Since $x \notin (\mathcal{B}')^-$ and $x = (x_1, x_2, \ldots, x_n) < y \stackrel{\text{def}}{=} (a_1^o, x_2, \ldots, x_n)$, the vector y also satisfies $y \in \mathcal{C}' \setminus (\mathcal{B}')^-$. We conclude, therefore, assuming (7.5), that $y \in \mathcal{A}^+$; i.e., there is an $a \in \mathcal{A}$ such that $a \le y$. However, this implies that $a \in \widetilde{\mathcal{A}}$ and hence that $x \in \mathcal{C}''(a) \setminus [(\mathcal{A}'')^+ \cup (\mathcal{B}'')^-]$ for some $a \in \widetilde{\mathcal{A}}$.

It follows by (7.8) that, once we discover that (7.5) holds, we can reduce the solution of subproblem (7.6) to solving $|\widetilde{\mathcal{A}}|$ subproblems, each of which has a volume of $v(|\mathcal{A}''|, |\mathcal{B}''|) \le \epsilon_1^{\mathcal{B}} v(\mathcal{A}, \mathcal{B})$. Thus we obtain the recurrence

$$C(v) \le 1 + C((1-\epsilon_1^{\mathcal{B}})v) + |\mathcal{A}|C(\epsilon_1^{\mathcal{B}} v). \tag{7.9}$$

Step 6. Finally, if $\epsilon_1^{\mathcal{A}} \le \epsilon(v) < \epsilon_1^{\mathcal{B}}$, we solve subproblem (7.6), and, if we discover that $\mathcal{A}'', \mathcal{B}$ are dual in $\mathcal{C}''$, we obtain the following decomposition rule, symmetric to (7.8):

$$\mathcal{A}, \mathcal{B}' \text{ are dual in } \mathcal{C}' \iff \text{for all } b \in \widetilde{\mathcal{B}} : \mathcal{A}', \mathcal{B}' \text{ are dual in } \mathcal{C}'(b),$$

where $\widetilde{\mathcal{B}} = \{b \in \mathcal{B} \mid b_1 \ge a_1^o - 1\}$ and $\mathcal{C}'(b) = \mathcal{C}_1' \times [l_2 : b_2] \times \cdots \times [l_n : b_n]$. This reduces our original problem into one subproblem of volume $\le (1-\epsilon_1^{\mathcal{A}})v$, plus $|\widetilde{\mathcal{B}}|$ subproblems, each of volume at most $\epsilon_1^{\mathcal{A}} v$, thus giving the recurrence

$$C(v) \le 1 + C((1-\epsilon_1^{\mathcal{A}})v) + |\mathcal{B}|C(\epsilon_1^{\mathcal{A}} v), \tag{7.10}$$

which is the symmetric version of (7.9).

Using induction on $v \geq 9$, it can be shown that recurrences (7.7), (7.9), and (7.10) imply $C(v) \leq v^{\chi(v)}$ (see [14]). As $\chi(m^2) < 2\chi(m)$ and $v(\mathcal{A}, \mathcal{B}) < m^2$, we get $\chi(v) < \chi(m^2) < 2\chi(m) \sim 2 \log m / \log \log m$. Let us also note that every step above can be implemented in $O(n^3 m)$ time, independently of the sizes of the chains $|\mathcal{C}_i|$. This establishes the bound stated in Theorem 3.2.

8. Bounding the size of $\mathcal{F}_{A,b,c}$. To prove inequality (4.1) of Theorem 4.1, let us consider an arbitrary nonempty antichain $\mathcal{Y} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$. For any $y \in \mathcal{I}(\mathcal{F}_{A,b,c})$, we can find an index $i = \rho(y) \in [r] \stackrel{\text{def}}{=} \{1, \ldots, r\}$ such that y violates the ith inequality of the system; i.e., $a^i y < b_i$, where a^i and b_i denote the ith row and component of A and b, respectively.

Consider a vector $x \in \mathcal{I}^{-1}(\mathcal{Y}) \cap \mathcal{F}_{A,b,c}$, and let x_l be a positive component of x. Then there exists a vector $y^l \in \mathcal{Y}$ such that $y^l \geq x - \mathbf{e}^l$. Let $i = \rho(y^l)$, and assume, without loss of generality, that

$$a^i_1 \geq a^i_2 \geq \cdots \geq a^i_n. \tag{8.1}$$

We claim that $x(l] = z^l(l]$, where

$$z^l = y^l(l] + \mathbf{e}^l. \tag{8.2}$$

It follows from $y^l \geq x - \mathbf{e}^l$ that $z^l(l] \geq x(l]$. If $z^l_l > x_l$, then $y^l_l \geq x_l$, which implies $y^l \geq x$, which is a contradiction. Thus $z^l_l = x_l$ holds. Moreover, if $z^l_j > x_j$ for some $j < l$, then we have $y^l_j \geq x_j + 1$. By (8.1), $a^i(y^l - \mathbf{e}^j + \mathbf{e}^l) < b_i$; i.e., $y^l - \mathbf{e}^j + \mathbf{e}^l$ is infeasible for (1.1). However, $y^l - \mathbf{e}^j + \mathbf{e}^l \geq x$ by $y^l \geq x - \mathbf{e}^l$, and hence $y^l - \mathbf{e}^j + \mathbf{e}^l$ must be feasible. This shows that $x(l] = z^l(l]$ and consequently leads to the representation

$$x = \bigvee_{l \in [n]:\, x_l > 0} z^l, \tag{8.3}$$

where, for vectors $v, u \in \mathcal{C}$, we let $v \vee u$ denote the componentwise maximum of v and u.

Not all of the vectors z^l are necessary for this representation. Suppose that $\rho(y^l) = \rho(y^{l'}) = i$ for some positive components x_l and $x_{l'}$ of x and for $l' < l$. Then (8.3) remains valid if we drop $z^{l'}$, the vector with the smaller index l'. In other words, by sorting the ith row of A and then selecting from among the vectors $y^l \in \rho^{-1}(i)$ the one with the highest $l = l_i$, we obtain at most r vectors z^{l_i} such that

$$x = \bigvee_{i \in [r]} z^{l_i}. \tag{8.4}$$

The latter representation readily implies (4.1).

9. Polynomial generation of $\mathcal{F}_{A,b,c}$ and $\mathcal{I}(\mathcal{F}_{A,b,c})$. Theorem 4.1 implies that, for $r \leq$ const, the antichain $\mathcal{I}(\mathcal{F}_{A,b,c})$ is uniformly dual-bounded, and, consequently, $\mathcal{I}(\mathcal{F}_{A,b,c})$ can be generated in incremental quasi-polynomial time via Algorithm $\mathcal{J}$ presented in section 6. In this section, we show that, for bounded r, the antichains $\mathcal{I}(\mathcal{F}_{A,b,c})$ and $\mathcal{F}_{A,b,c}$ can, in fact, be generated in incremental polynomial time. Since the sizes $|\mathcal{F}_{A,b,c}|$ and $|\mathcal{I}(\mathcal{F}_{A,b,c})|$ are (uniformly) polynomially related by Theorems 2.1 and 4.1, the required result will follow from Proposition 6.1, provided that Step 3 of Algorithm $\mathcal{J}$, the dualization step, can be done in polynomial time. Thus it is enough to show that problem DUAL$(\mathcal{C}, \mathcal{A}, \mathcal{B})$ can be solved in polynomial

time if $\mathcal{A} \subseteq \mathcal{F}_{A,b,c}$ is a subset of the minimal solutions of a monotone system (1.1) with bounded r.

For $i \in [r] = \{1, \dots, r\}$, let $\sigma_{(i)} = (\sigma_{(i)1}, \dots, \sigma_{(i)n}) \in \mathbb{S}_n$ be a permutation of the coordinates such that

$$a^i_{\sigma_{(i)1}} \geq a^i_{\sigma_{(i)2}} \geq \cdots \geq a^i_{\sigma_{(i)n}}. \tag{9.1}$$

Given $\mathcal{A} \subseteq \mathcal{F}_{A,b,c}$ and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{F}_{A,b,c})$, we may assume that $0 \notin \mathcal{A}$ (otherwise, $\mathcal{F}_{A,b,c} = \{0\}$ and $\mathcal{I}(\mathcal{F}_{A,b,c}) = \emptyset$) and $\mathcal{B} \neq \emptyset$ (otherwise, Proposition 7.1 can be used to generate a point $x \in \mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$). Now we proceed in two basic steps.

Step 1. For every $y \in \mathcal{B}$ and for each pair of indices $\sigma_{(i)j}$ and $\sigma_{(i)l}$ with $y_{\sigma_{(i)j}} > 0$, $y_{\sigma_{(i)l}} < c_{\sigma_{(i)l}}$, and $j < l$, check if there exists a $y' \in \mathcal{B}$ such that

$$y' \geq y - \mathbf{e}^{\sigma_{(i)j}} + \mathbf{e}^{\sigma_{(i)l}}, \tag{9.2}$$

where $i = \rho(y)$ is the index of an infeasible inequality for y, as defined in the previous section. Note that $y - \mathbf{e}^{\sigma_{(i)j}} + \mathbf{e}^{\sigma_{(i)l}}$ is infeasible and hence is an independent element of $\mathcal{A}$. If no such y' can be found, we generate a new maximal independent vector $y' \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, satisfying (9.2), and halt.

Step 2. For every collection $(y^i \in \rho^{-1}_{\mathcal{B}}(i) \mid i \in [r])$, where $\rho^{-1}_{\mathcal{B}}(i) \stackrel{\text{def}}{=} \{y \in \mathcal{B} \mid \rho(y) = i\}$, and for every $(l_i \mid i \in [r], y^i_{l_i} < c_{l_i}) \in [n]^r$, construct the vector $x = \bigvee_{i \in [r]} z^{l_i}$, where z^{l_i} is given by (8.2) (according to the permutation $\sigma_{(i)}$ and using $y^l = y^{l_i}$). If $x \notin \mathcal{A}^+ \cup \mathcal{B}^-$, then a new maximal independent vector can be generated.

Clearly, the above two steps run in $poly(n, m) + O((n|\mathcal{B}|)^r)$ time, which is polynomially bounded for constant r. It is also clear that, if the algorithm outputs a point $x \in \mathcal{C}$, then $x \notin \mathcal{A}^+ \cup \mathcal{B}^-$, so it remains to verify that the algorithm indeed outputs such a point if $\mathcal{A}^+ \cup \mathcal{B}^- \neq \mathcal{C}$. To see this, let x be a minimal vector in $\mathcal{C} \setminus (\mathcal{A}^+ \cup \mathcal{B}^-)$. From our assumptions, it follows that $x \neq 0$, and thus there exists an index l with $x_l > 0$. By the minimality of x, there exists a vector $y^l \in \mathcal{B}$ such that $y^l \geq x - \mathbf{e}^l$. Let $i = \rho(y^l)$, assume without any loss of generality that (8.1) holds, and consider an index $j < l$. If $y^l_j > x_j$, we get $y^l - \mathbf{e}^j + \mathbf{e}^l \geq x$, and therefore a new maximal independent point $x' \geq x$ must have been output in Step 1 of the algorithm (cf. (9.2)). On the other hand, if, for every $l \in [n]$ such that $x_l > 0$ and for every $y^l \in \mathcal{B}$ such that $y^l \geq x - \mathbf{e}^l$, we have $x(l] = z^l(l]$ (in the ordering implied by $\sigma(i)$, where $i = \rho(y^l)$), then we can conclude that x satisfies (8.4), and, consequently, it must have been created in Step 2.

10. Concluding remarks. We mention in closing that, in section 5, we actually proved Theorem 2.1 for arbitrary systems of 2-monotonic inequalities in integer variables. Consequently, the set of minimal feasible integer vectors for any system of 2-monotonic inequalities is uniformly dual-bounded. By Proposition 6.2, this implies that all minimal integer solutions to a system of 2-monotonic inequalities can be generated in incremental quasi-polynomial time, provided that the system has a polynomial-time feasibility oracle. Theorems 4.1 and 4.2 also hold for arbitrary systems of 2-monotonic inequalities. These generalize results known for a single Boolean 2-monotonic inequality discussed in [4, 8, 9, 18, 24, 26, 27].

Acknowledgments. K. Elbassioni and V. Gurvich are grateful for partial support from DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science. K. Makino is grateful for partial support from the Scientific Grant in Aid of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

REFERENCES

[1] E. Balas and E. Zemel, *All the Facets of Zero-One Programming Polytopes with Positive Coefficients*, Management Sciences Research Report 374, Carnegie Mellon University, Pittsburgh, PA, 1975.

[2] J. C. Bioch, *Dualization, decision lists and identification of monotone discrete functions*, Ann. Math. Artificial Intelligence, 24 (1998), pp. 69–91.

[3] J. C. Bioch and T. Ibaraki, *Complexity of identification and dualization of positive Boolean functions*, Inform. and Comput., 123 (1995), pp. 50–63.

[4] P. Bertolazzi and A. Sassano, *An $O(nm)$ algorithm for regular set-covering problems*, Theoret. Comput. Sci., 54 (1987), pp. 237–247.

[5] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan, *An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension*, Parallel Process. Lett., 10 (2000), pp. 253–266.

[6] E. Boros, V. Gurvich, and P. L. Hammer, *Dual subimplicants of positive Boolean functions*, Optim. Methods Softw., 10 (1998), pp. 147–156.

[7] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, *Dual-bounded generating problems: Partial and multiple transversals of a hypergraph*, SIAM J. Comput., 30 (2001), pp. 2036–2050.

[8] E. Boros, P. L. Hammer, T. Ibaraki, and K. Kawakami, *Polynomial-time recognition of 2-monotonic positive Boolean functions given by an oracle*, SIAM J. Comput., 26 (1997), pp. 93–109.

[9] Y. Crama, *Dualization of regular Boolean functions*, Discrete Appl. Math., 16 (1987), pp. 79–85.

[10] Y. Crama, P. L. Hammer, and T. Ibaraki, *Cause-effect relationships and partially defined Boolean functions*, Ann. Oper. Res., 16 (1988), pp. 299–326.

[11] C. Domingo, N. Mishra, and L. Pitt, *Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries*, Machine Learning, 37 (1999), pp. 89–110.

[12] T. Eiter and G. Gottlob, *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput., 24 (1995), pp. 1278–1304.

[13] T. Eiter, G. Gottlob, and K. Makino, *New results on monotone dualization and generating hypergraph transversals*, in Proceedings of the 34th ACM Symposium on Theory of Computing, Montreal, QC, Canada, 2002, pp. 14–22.

[14] M. L. Fredman and L. Khachiyan, *On the complexity of dualization of monotone disjunctive normal forms*, J. Algorithms, 21 (1996), pp. 618–628.

[15] D. R. Gaur, *Satisfiability and Self-Duality of Monotone Boolean Functions*, Ph.D. thesis, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, 1999.

[16] D. R. Gaur and R. Krishnamurti, *Self-duality of bounded monotone Boolean functions and related problems*, in Proceedings of the 11th International Conference on Algorithmic Learning Theory, Lecture Notes in Comput. Sci. 1968, Springer-Verlag, Berlin, 2000, pp. 209–223.

[17] V. Gurvich and L. Khachiyan, *On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions*, Discrete Appl. Math., 96–97 (1999), pp. 363–373.

[18] P. L. Hammer, U. N. Peled, and M. A. Pollatschek, *An algorithm to dualize a regular switching function*, IEEE Trans. Comput., 28 (1979), pp. 238–243.

[19] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.

[20] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, SIAM J. Comput., 9 (1980), pp. 558–565.

[21] K. Makino, *Efficient Dualization of $O(\log n)$-Term Monotone Disjunctive Normal Forms*, Tech. report 00-07, Discrete Mathematics and Systems Science, Osaka University, Osaka, Japan, 2000; Discrete Appl. Math., to appear.

[22] K. Makino and T. Ibaraki, *Interior and exterior functions of Boolean functions*, Discrete Appl. Math., 69 (1996), pp. 209–231.

[23] K. Makino and T. Ibaraki, *The maximum latency and identification of positive Boolean functions*, SIAM J. Comput., 26 (1997), pp. 1363–1383.

[24] K. Makino and T. Ibaraki, *A fast and simple algorithm for identifying 2-monotonic positive Boolean functions*, J. Algorithms, 26 (1998), pp. 291–305.

[25] O. L. Mangasarian, *Mathematical programming in machine learning*, in Nonlinear Optimization and Applications, G. Di. Pillo and F. Giannessi, eds., Plenum, New York, 1996, pp. 283–295.

[26] U. N. Peled and B. Simeone, *Polynomial-time algorithm for regular set-covering and threshold synthesis*, Discrete Appl. Math., 12 (1985), pp. 57–69.
[27] U. N. Peled and B. Simeone, *An $O(nm)$-time algorithm for computing the dual of a regular Boolean function*, Discrete Appl. Math., 49 (1994), pp. 309–323.
[28] H. Tamaki, *Space-efficient enumeration of minimal transversals of a hypergraph*, IPSJ J., 75 (2000), pp. 29–36.
[29] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, *A new algorithm for generating all the maximal independent sets*, SIAM J. Comput., 6 (1977), pp. 505–517.
[30] T. Uno, *Private communication*, Department of Applied Mathematics and Physics, Kyoto University, Kyoto, Japan, 2002.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1645–1662

LOWER BOUNDS FOR LEADER ELECTION AND COLLECTIVE COIN-FLIPPING IN THE PERFECT INFORMATION MODEL*

ALEXANDER RUSSELL†, MICHAEL SAKS‡, AND DAVID ZUCKERMAN§

Abstract. Collective coin-flipping is the problem of producing common random bits in a distributed computing environment with adversarial faults. We consider the *perfect information* model: all communication is by broadcast and corrupt players are computationally unbounded. Protocols in this model may involve many asynchronous rounds. We assume that honest players communicate only uniformly random bits. We demonstrate that any n-player coin-flipping protocol that is resilient against corrupt coalitions of linear size must use either at least $[1/2 - o(1)] \log^* n$ communication rounds or at least $[\log^{(2k-1)} n]^{1-o(1)}$ communication bits in the kth round, where $\log^{(j)}$ denotes the logarithm iterated j times. In particular, protocols using one bit per round require $[1/2 - o(1)] \log^* n$ rounds. These bounds also apply to the leader election problem. The primary component of this result is a new bound on the influence of random sets of variables on Boolean functions. Finally, in the one-round case, using other methods we prove a new bound on the influence of sets of variables of size βn for $\beta > 1/3$.

Key words. perfect information model, collective coin-flipping, leader election

AMS subject classifications. 68Q17, 91A15, 05D40

PII. S0097539700376007

1. Introduction.

Collective coin-flipping is the problem of producing a common random bit in a distributed computing environment with adversarial faults. We consider the *perfect information* model introduced by Ben-Or and Linial [5], which can be informally described as follows. A protocol in this model consists of a sequence of rounds. In each round, each player privately generates a uniformly random string of bits of some specified length (possibly 0) and broadcasts the string. Each broadcast is received by all players and the identity of the sender is known with certainty. The round ends after all broadcasts are received. After the completion of all rounds, the outcome of the protocol is computed separately by each player as a prespecified function of all the values broadcast during the protocol; for the coin-flipping problem the outcome is a single bit. A protocol Π is said to be an (n, r, ℓ)-protocol if n is the number of players, r is the number of rounds, and each player broadcasts at most ℓ bits in each round.

Faults are modeled by the presence of an unknown set of b corrupt players who collude in order to bias the outcome. Players are assumed to be computationally unbounded. In addition, the system is not able to enforce perfect synchrony within a

*Received by the editors July 28, 2000; accepted for publication (in revised form) April 18, 2002; published electronically September 5, 2002. A preliminary version of this paper appeared in *Proceedings of the* 31*st ACM Symposium on Theory of Computing*, 1999, pp. 339–347.

http://www.siam.org/journals/sicomp/31-6/37600.html

†Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269 (acr@cse.uconn.edu). This research was done while the author was a postdoctoral fellow at the University of Texas at Austin. The research of the first author was supported by NSF NYI grant CCR-9457799 and a David and Lucile Packard Fellowship for Science and Engineering.

‡Department of Mathematics, Rutgers University, Piscataway, NJ 08854 (saks@math.rutgers.edu). The research of the second author was supported by NSF grant CCR-9700239. This work was done in part while on sabbatical at University of Washington.

§Department of Computer Science, University of Texas, Austin, TX 78712 (diz@cs.utexas.edu). The research of the third author was supported in part by NSF NYI grant CCR-9457799, a David and Lucile Packard Fellowship for Science and Engineering, and an Alfred P. Sloan Research Fellowship.

round; thus in each round, the corrupt players may wait to see the broadcasts of the other players before selecting their strings.

While not necessary for previous upper bounds, we strengthen our lower bounds by assuming that corrupt players may cheat only in ways that are undetectable to the other players. This means that when the protocol specifies that such a player broadcast a bit string of a given length, he must do so; however, he may cheat by broadcasting a string that he chooses rather than a random string.

The simplest protocol is one that designates a single player to flip a coin, the value of which is the outcome of the protocol; of course, this is unsatisfactory if that player happens to be faulty. More generally, an $(n, 1, 1)$-protocol is defined by a Boolean function $f : \{0,1\}^n \to \{0,1\}$; each player i broadcasts a bit r_i and the outcome is $f(r_1, \ldots, r_n)$. Throughout the paper we use the terms Boolean function and $(n, 1, 1)$-protocol interchangeably.

The primary goal in designing a protocol is to ensure that it can tolerate as many cheaters as possible.

DEFINITION 1.

1 *Let* Π *be a coin-flipping protocol for* n *players, and let* $\gamma \in (0, 1/2]$.

(a) *For* $B \subseteq [n]$, Π *is* (B, γ)-resilient *if for any strategy of the players in* B,

$$\gamma \leq \Pr[\Pi \text{ has outcome } 1] \leq 1 - \gamma,$$

where the probability is taken with respect to the random bits generated by the players outside of B.

(b) Π *is* (b, γ)*-resilient for an integer* $b \leq n$ *if it is* (B, γ)*-resilient for all* B *with* $|B| \leq b$.

2. *Let* $\Pi = (\Pi_n : n \geq 1)$ *be a sequence where* Π_n *is an* n*-player protocol, and let* $b(n)$ *be a function mapping* n *to an integer* $b(n) \leq n$. *We say that* Π *is* $b(n)$-resilient *if there exists* $\gamma > 0$ *(independent of* n*) such that for all* n, Π_n *is* $(b(n), \gamma)$*-resilient.*

For example, in the case of $(n, 1, 1)$-protocols, the PARITY function, $\sum_{i=1}^{n} r_i \bmod 2$, is not even 1-resilient, while the MAJORITY function is $c\sqrt{n}$-resilient for any positive c. Ajtai and Linial [1] constructed a Boolean function that is $\Omega(n/\log^2 n)$-resilient. Kahn, Kalai, and Linial, in a 1988 tour de force, proved an upper bound on the resilience of Boolean functions.

THEOREM 2 (see [13]). *If* $b(n) = \omega(\frac{n}{\log n})$, *then no sequence* $(f_n : n \geq 1)$ *is* $b(n)$*-resilient.*

We emphasize that this bound applies only to $(n, 1, 1)$-protocols. Indeed, Alon and Naor [2] showed that there are protocols using n rounds that are $\Omega(n)$-resilient, and this was followed by a sequence of papers giving more efficient protocols with linear resilience. In what follows $\log^{(k)}(n)$ denotes the maximum of 1 and the kth iterated base 2 logarithm, and $\log^* n$ is the least integer k such that $\log^{(k)}(n) = 1$. The most efficient known protocol is that of [16], requiring $\log^* n + O(1)$ rounds; players send messages of length $O(\log^{(k)} n)$ during the kth round. The protocol achieves βn-resilience for any $\beta < 1/2$. (As noted in [17] no protocol can be $n/2$-resilient.) This protocol can be modified to yield a one bit per round protocol with $\big[1 + o(1)\big] \log n$ rounds. Subsequently, Feige [9] gave a simpler protocol with similar properties.

Despite rapid progress in our understanding of *protocols* for the problem, very little beyond Theorem 2 was known on the negative side. The major contribution of this article is an extension of Theorem 2 to protocols with many rounds. We will prove the following theorem.

THEOREM 3. *Let $\Pi = (\Pi_n : n \geq 1)$ be a sequence of protocols, where Π_n is an $(n, r(n), 1)$-protocol for $r(n) \leq \frac{1}{2}\log^* n - \log^* \log^* n$. Then*

1. *Π_n is not $\Omega(n)$-resilient and*
2. *if*

$$b(n) = \omega\left(\frac{(r(n))^2}{\log^{(2r(n)-1)} n} \cdot n\right),$$

then Π is not $b(n)$-resilient.

For instance, when $r(n) = 1$ this reduces to Theorem 2, and when $r(n) = 2$ it implies that no $(n, 2, 1)$-protocol can be $\omega(n/\log\log\log n)$-resilient.

We extend the notation above to describe protocols with variable communication complexity: a protocol Π is said to be an $(n, r, \vec{\ell})$-protocol if n is the number of players, r is the number of rounds, and no more than ℓ_k bits are broadcast by any player in the kth round, where $\vec{\ell} = (\ell_1, \ldots, \ell_r)$. We will prove that the conclusion of Theorem 3 holds even if we relax the requirement that each player sends only one bit per round.

THEOREM 4. *There is a function $\eta : \mathbb{N} \to [0, 1]$ with $\eta(n) = o(1)$ so that for any sequence $\Pi = (\Pi_n : n \geq 1)$ of protocols, where Π_n is an $(n, r(n), \vec{\ell})$-protocol with*

$$r(n) \leq \frac{1}{2}\log^* n - \log^* \log^* n \qquad \text{and} \qquad \ell_k(n) \leq (\log^{(2k-1)} n)^{1-\eta(n)},$$

Π_n is not $\Omega(n)$-resilient.

Recall that current upper bounds provide $(n, \log^* n + O(1), \vec{\ell})$-protocols which are linearly resilient, where $\ell_k = O(\log^{(k)} n)$.

The *leader election* problem is that of selecting a "leader" among n players so that the probability that any coalition (of appropriate size) can elect one of its own members is at most $1 - \epsilon$ for a constant $\epsilon > 0$ independent of n. Adopting the above model, the notion of resilience may be extended to this scenario. Collective coin-flipping may be reduced to leader election at the cost of an extra round: the leader may flip a fair coin. Our bounds shall then naturally apply to this problem as well. For a more detailed discussion of coin-flipping, leader election, and the perfect information model, see [7, 14].

Section 2 gives definitions, notation, and preliminary facts. The two main theorems are proved in section 3 and section 4. In section 5 an observation is made about the behavior of large linear sized coalitions. We conclude with some open questions.

2. Preliminaries.

2.1. General notation. Throughout, $\ln x$ denotes the natural logarithm and $\log x$ the logarithm base 2. To avoid logarithms of negative numbers, iterated logarithms are defined inductively as follows: for $x \geq 1$, $\log^{(0)}(x) = x$, and for $k \geq 1$,

$$\log^{(k)} x = \begin{cases} 1 & \text{if } \log^{(k-1)} x < 2, \\ \log\left(\log^{(k-1)} x\right) & \text{otherwise.} \end{cases}$$

For $x \geq 1$, define $\log^*(x)$ to be the smallest natural number k for which $\log^{(k)} x = 1$.

For a positive real number y and integer k the *tower* function $\mathrm{T}(k; y)$ is defined by

$$\mathrm{T}(0; y) = y, \text{ and}$$
$$\mathrm{T}(k; y) = 2^{\mathrm{T}(k-1;y)} \text{ for } k > 0.$$

Observe that for any $y \geq 1$, $k \leq \ell$, $\log^{(k)}(\mathrm{T}(\ell; y)) = \mathrm{T}(\ell - k; y)$.

For an integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. For $J \subseteq [n]$, a finite set X, and $\alpha \in X^J$, $C(\alpha)$ denotes the set of all points $x \in X^n$ such that $x_j = \alpha_j$ for all $j \in J$. If $\alpha \in X^J$ and $\beta \in X^{[n]\setminus J}$, then $[\alpha : \beta]$ denotes the unique point of $\{0,1\}^n$ belonging to $C(\alpha) \cap C(\beta)$.

If S is a set, the notation $x \in_U S$ indicates that x is selected uniformly at random from S.

2.2. Coin-flipping protocols and influence. We want to formalize the definition of protocol given in the introduction. We below define $(n, r, \vec{\ell})$-protocols and a number of related notions; (n, r, ℓ)-protocols, where communication is constant across rounds, are covered as a special case. For an $(n, r, 1)$-protocol Π we suppress the third index and simply say that Π is an (n, r)-protocol.

Formally, an $(n, r, \vec{\ell})$-*protocol* is a function

$$\Pi : (\{0,1\}^{\ell_1})^n \times \cdots \times (\{0,1\}^{\ell_r})^n \to \{0,1\}.$$

Such a protocol is executed in r rounds. In the presence of a set $B \subset [n]$ of bad players, the protocol operates as follows. In round i, the players in $[n] \setminus B$ select $\alpha^i \in (\{0,1\}^{\ell_i})^{[n]\setminus B}$ uniformly at random. Then, depending on $\alpha^1, \ldots, \alpha^i$, the players in B choose their values. Formally, an $(n, r, \vec{\ell})$-*strategy* for B is a sequence $S = (S_1, S_2, \ldots, S_r)$ of functions where

$$S_i : \left(\{0,1\}^{\ell_1}\right)^{[n]\setminus B} \times \cdots \times \left(\{0,1\}^{\ell_i}\right)^{[n]\setminus B} \to \left(\{0,1\}^{\ell_i}\right)^{B}.$$

The function S_i specifies the choices of the bad players in round i as a function of the choices of the good players in the first i rounds. The outcome of protocol Π, with bad player set B playing strategy S, is a function of the sequence

$$\vec{\alpha} = (\alpha^1, \ldots, \alpha^r) \in \left(\{0,1\}^{\ell_1}\right)^{[n]\setminus B} \times \cdots \times \left(\{0,1\}^{\ell_r}\right)^{[n]\setminus B}$$

of the random coins of the good players, which is denoted $\Pi(\vec{\alpha}; S)$ and is defined to be

$$\Pi\big([\alpha^1 : S_1(\alpha^1)], \ldots, [\alpha^r : S_r(\alpha^1, \ldots, \alpha^r)]\big).$$

Definition 5. *For a protocol Π, $B \subseteq [n]$, and strategy S,*

- *$p^1_\Pi(B; S)$ denotes the probability that $\Pi(\vec{\alpha}; S) = 1$ if*

$$\vec{\alpha} \in_U \left(\{0,1\}^{\ell_1}\right)^{[n]\setminus B} \times \cdots \times \left(\{0,1\}^{\ell_r}\right)^{[n]\setminus B};$$

- *$p^1_\Pi(B)$ is the maximum of $p^1_\Pi(B; S)$ over all strategies S;*
- *$p^1_\Pi = p^1_\Pi(\emptyset)$, the* natural probability of Π*, is the probability that the outcome is 1 if there are no bad players;*
- *$I^1_\Pi(B)$, the* influence of B towards 1*, is defined to be $p^1_\Pi(B) - p^1_\Pi$;*
- *$p^0_\Pi(B; S)$, $p^0_\Pi(B)$, p^0_Π, and $I^0_\Pi(B)$ are defined analogously;*
- *$I_\Pi(B)$, the* influence of B on Π*, is defined to be*

$$I^1_\Pi(B) + I^0_\Pi(B).$$

An $(n,1)$-protocol corresponds to a Boolean function $f : \{0,1\}^n \to \{0,1\}$, and we typically use the letter f (instead of Π) for such a protocol. It is not hard to see that $p_f^1(B)$ is the probability, with respect to $\alpha \in_U \{0,1\}^{[n]\setminus B}$, that $1 \in f(C(\alpha))$ and that $I_f(B)$ is the probability that f is not constant on $C(\alpha)$. Furthermore, if $|B| = 1$, then $I_f^1(B) = I_f^0(B)$.

The following result, observed in [6] (cf. Proposition 2.2 of [13]), implies that the most resilient one-round protocols are given by Boolean functions that are monotone.

PROPOSITION 6. *For any Boolean function f, there exists a monotone Boolean function g on the same set of variables for which*

1. $p_f^1 = p_g^1$ *and*
2. *for all $B \subset [n]$, $I_f^1(B) \geq I_g^1(B)$ and $I_f^0(B) \geq I_g^0(B)$.*

Finally, we need a variant of a fact first noted in [10] and based on a result in [13], which asserts that if no variable in a Boolean function has large influence, then the average influence of a variable cannot be too small. For completeness, we include a proof.

LEMMA 7. *Let $\gamma \in (0, \frac{1}{2})$ and $\theta \in (0, \frac{1}{8})$. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function with $p_f^1 \in (\gamma, 1-\gamma)$. If $I_f(\{i\}) \leq \theta$ for each $i \in [n]$, then*

$$\sum_{i\in[n]} I_f(\{i\}) \geq \frac{\gamma \log(\frac{1}{\theta})}{20}.$$

Proof. Let $\bar{v} \in \mathbb{R}^n$ denote the vector with $v_i = I_f(\{i\})$. For $p > 0$, the l_p norm of $\bar{v}$, denoted $\|\bar{v}\|_p$, is defined to be $(S_p)^{1/p}$, where $S_p = \sum |v_i|^p$.

By complementing if necessary, we may assume that $p_f^1 \leq 1/2$. Since the function f is Boolean and $p_f^1 \geq \gamma$, [13, eq. (3.4.1)] asserts that for any $\delta \in (0,1)$ and $t \geq 1$,

$$\delta^{-t} S_{\frac{2}{1+\delta}} + t^{-1} S_1 \geq \frac{\gamma}{2}. \tag{2.1}$$

Inequality (2.10.1) of Hardy, Littlewood, and Pólya [11] asserts that

$$S_r \leq (S_q)^{\frac{s-r}{s-q}} (S_s)^{\frac{r-q}{s-q}}$$

for $0 < q < r < s$, which is equivalent to

$$(S_r)^{\frac{s-q}{s}} \leq (S_q)^{\frac{s-r}{s}} (\|v\|_s)^{r-q}.$$

Setting $q = 1$ and $r = \frac{2}{1+\delta}$ and letting s tend to ∞, we obtain

$$S_{\frac{2}{1+\delta}} \leq S_1 \theta^{\frac{1-\delta}{1+\delta}}.$$

Substituting this into the inequality (2.1) and setting $\delta = 1/2$ we get

$$\left(2^t \theta^{\frac{1}{3}} + \frac{1}{t}\right) S_1 \geq \frac{\gamma}{2}.$$

Choose t such that $\theta = 2^{-3t}/t^3$ (noting that $t > 1$ since $\theta < 1/8$). Then the previous inequality implies $S_1 \geq t\gamma/4$. Since $2^{-3t}/t^3 \geq 2^{-5t}$ for $t \geq 1$, we have

$$t \geq \frac{1}{5} \log\left(\frac{1}{\theta}\right),$$

and therefore

$$S_1 \geq \frac{\gamma \log \frac{1}{\theta}}{20}. \qquad \square$$

2.3. A tail bound for submartingales. Our main theorems are proved by considering a certain stochastic process which, for a Boolean function, selects a set of variables likely to have large influence. Our analysis of this stochastic process involves a tail bound for submartingales, which we record below.

DEFINITION 8. *A* submartingale *is a sequence of real valued random variables* $Z_0, Z_1, \ldots$ *for which* $\mathbb{E}[Z_i \mid Z_{i-1}] \geq Z_{i-1}$.

We were unable to find the exact form of the following tail bound in the literature, so we have included a proof. The basic method is developed in [8, 4, 12]. Our treatment follows [3, 15].

LEMMA 9. *Let* $(Z_i : i \in \{0, \ldots, n\})$ *form a submartingale with* $Z_0 = 0$. *Define* $X_i = Z_i - Z_{i-1}$ *for* $i \in \{1, \ldots, n\}$ *and assume that* $X_i \in [0,1]$ *and* $\mathbb{E}[X_i \mid Z_{i-1}] \geq \mu_i$. *Setting* $\mu = \sum_i \mu_i$ *and* $Z = Z_n$,

$$\Pr[Z < (1-\delta)\mu] < e^{-\frac{\delta^2 \mu}{2}}$$

for all $\delta > 0$.

Proof. Observe that for any $\alpha > 0$,

$$\Pr[Z < (1-\delta)\mu] = \Pr\left[e^{-\alpha Z} > e^{-\alpha(1-\delta)\mu}\right] < \frac{\mathbb{E}\left[e^{-\alpha Z}\right]}{e^{-\alpha(1-\delta)\mu}}.$$

Letting $\ell(x) = 1 + x(e^{-\alpha} - 1)$, we have $e^{-\alpha x} \leq \ell(x)$ for all $x \in [0,1]$ because the exponential function is convex. For any $[0,1]$ valued random variable Y,

$$\mathbb{E}\left[e^{-\alpha Y}\right] \leq \mathbb{E}\left[\ell(Y)\right] = 1 + \mathbb{E}[Y](e^{-\alpha} - 1).$$

By induction, we compute

$$\begin{aligned}
\mathbb{E}\left[e^{-\alpha Z_k}\right] &= \mathbb{E}\left[e^{-\alpha Z_{k-1}} \cdot e^{-\alpha X_k}\right] \\
&= \mathbb{E}\left[\left(e^{-\alpha Z_{k-1}}\right) \mathbb{E}\left[e^{-\alpha X_k} \,\middle|\, Z_{k-1}\right]\right] \\
&\leq \mathbb{E}\left[\left(e^{-\alpha Z_{k-1}}\right) \left(1 + \mathbb{E}\left[X_k \,\middle|\, Z_{k-1}\right](e^{-\alpha} - 1)\right)\right] \\
&\leq \prod_i (1 + \mu_i(e^{-\alpha} - 1)) < e^{\sum_i^k \mu_i (e^{-\alpha} - 1)}.
\end{aligned}$$

Hence

$$\Pr\left[Z < (1-\delta)\mu\right] < \frac{e^{\mu(e^{-\alpha}-1)}}{e^{-\alpha(1-\delta)\mu}}.$$

Setting $\alpha = \ln(\frac{1}{1-\delta})$, we have

$$\Pr\left[Z < (1-\delta)\mu\right] < \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu} \leq e^{-\frac{\delta^2 \mu}{2}},$$

since $(1-\delta)^{(1-\delta)} > e^{-\delta + \frac{\delta^2}{2}}$. ∎

3. Proof of Theorem 3. We begin by considering (n, r)-protocols; each round consists of a single bit broadcast by each player. Fix the integer r. We say that a protocol Π is α-*nontrivial* if the natural probability of Π is at least α, i.e., $p^1_\Pi \geq \alpha$, terminology that we apply also in the multibit case. By complementing the output

if necessary, we may assume that the protocol is 1/2-nontrivial. We want to show that if Π is an (n,r)-protocol, then for n sufficiently large there is a set B of $b \ll n$ players so that B can almost always force the outcome to 1. For $r = 1$ this follows from Theorem 2.

We illustrate the ideas for $r > 1$ by looking at the two round case. By separating the inputs associated with each round, a two round protocol may be viewed as a function to functions:

$$\Pi : \{0,1\}^n \to \{g : \{0,1\}^n \to \{0,1\}\}.$$

As Π is 1/2-nontrivial, many g's will be 1/4-nontrivial (any constant less than 1/2 would do) and by Theorem 2, for each such g there is a sublinear set of players B_2 that can force this g to be 1 with high probability. Also by Theorem 2, there is a sublinear set of players B_1 that can likewise force the output of Π to be one of these g's. A natural strategy is to choose $B = B_1 \cup B_2$.

The problem with this plan is that B_2 depends on g; we really need one B_2 that works for many g's. We show this by proving that a random B_2 will work with significant probability for any 1/4-nontrivial g. It follows that a random B_2 will work for many g's. For general r, we will proceed by induction, with our inductive assumption being that a random sublinear set of players can control the protocol with significant probability.

To make these ideas rigorous, we begin with some definitions. For $\beta \in [0,1]$, we say that a subset B is *β-powerful in* Π if $p^1_\Pi(B) \geq 1 - \beta$.

DEFINITION 10. *Let $C_n(r;\alpha,\beta)$ (written $C_n(r;\gamma)$ when $\alpha = \beta = \gamma$) denote the collection of pairs $\langle \delta, b\rangle$ so that for any (n,r)-protocol Π that is α-nontrivial, at least a δ fraction of sets $B \subset [n]$ of size b are β-powerful in Π.*

In this notation, we are aiming to show that for some $\delta > 0$ and $b \ll n$, $(\delta, b) \in C_n(r; 1/2, o(1))$ for sufficiently large n. We prove the somewhat stronger statement that $(\delta, b) \in C_n(r; o(1))$.

The basis case of the induction on r is provided by the following result for one-round protocols.

LEMMA 11. *Let $n \in \mathbb{N}$ and $\gamma \in (0, \frac{1}{2})$ and $b \leq n$, and assume $\gamma b \geq 400n/\log n$. Then $\langle \delta(n,b,\gamma), b\rangle \in C_n(1;\gamma)$, where*

$$\delta(n,b,\gamma) = \frac{1}{2}\left(\frac{b}{4n}\right)^{2^{\frac{80n}{b\gamma}}}.$$

The induction step is provided by the following lemma.

LEMMA 12. *Fix n. If $\langle \delta_1, b_1\rangle \in C_n(r_1;\gamma_1)$ and $\langle \delta_2, b_2\rangle \in C_n(r_2;\gamma_2)$, then*

$$\left\langle \frac{\delta_1\delta_2}{2}, b_1 + b_2 \right\rangle \in C_n\left(r_1 + r_2; \frac{2\gamma_1}{\delta_2} + \gamma_2\right).$$

These two lemmas are combined to prove the following lemma.

LEMMA 13. *Let $b \leq n \in \mathbb{N}$ and $\gamma \in (0, \frac{1}{2})$. Define $\lambda_0 = \frac{1}{2}$ and for $r \geq 1$ define*

$$\lambda_r = 4\lambda_{r-1}\left(\frac{4n}{b}\right)^{2^{\frac{160n}{b\gamma}\lambda_{r-1}}}.$$

Then for all $r \geq 1$ such that $\gamma b \geq 800\lambda_{r-1}n/\log n$,

$$\left\langle \frac{1}{\lambda_r}, rb \right\rangle \in C_n(r; r\gamma).$$

An immediate consequence of this lemma is the following corollary.

COROLLARY 14. *Let n, b, γ, and λ_i be as in Lemma 13 and suppose that r is an integer such that*

$$\frac{\gamma b \log n}{800n} \geq \lambda_{r-1}.$$

Then if Π is an (n,r)-protocol that is $r\gamma$-nontrivial, there exists at least one subset B of size rb that is $r\gamma$-powerful.

We first deduce the main theorem from this corollary.

Proof of Theorem 3. We prove the second part of the theorem first.

Let $r(n)$ be an integer valued function with $r(n) \leq (1/2 - \epsilon)\log^* n$ and let $\Pi = (\Pi_n : n \geq 1)$ be a sequence where Π_n is an $(n, r(n))$-protocol. Let

$$b(n) = \frac{(r(n))^2 n}{\log^{(2r(n)-1)} n} a(n),$$

where $a(n)$ is any function tending to infinity. Let n be sufficiently large, and suppose for contradiction that for some $\gamma > 0$, Π is $(b(n), \gamma)$-resilient. Without loss of generality we may assume that $p^1_{\Pi_n} \geq 1/2$. Let $b'(n) = b(n)/r(n)$ and

$$\gamma' = \gamma'(n) = \frac{\gamma}{2r(n)}.$$

By the previous corollary applied to b' and γ', if

$$\frac{\gamma a(n) \log n}{1600 \log^{(2r-1)} n} \geq \lambda_{r-1}, \tag{3.1}$$

then there is at least one subset of size $r(n)b'(n) = b(n)$ that is $(r(n)\gamma'(n) = \gamma/2)$-powerful, which would contradict our assumption. So it suffices to show that inequality (3.1) hold When $r = 1$ inequality (3.1) holds, for large enough n, by inspection. Otherwise, taking $\log^{(2r-2)}$ of both sides, the left-hand side is at least $\frac{1}{2}\log^{(2r-1)} n$ for large enough n and so it suffices to show that this is an upper bound on $\log^{(2r-2)} \lambda_{r-1}$. In the following proposition, T denotes the tower function, as defined in section 2.1.

PROPOSITION 15. *Let $b \leq n$ and $\gamma \in (0,1)$. For all integers $r \geq 0$, $\lambda_r \leq \kappa_r$ where*

$$\kappa_r = \frac{b\gamma}{320n} \mathrm{T}\left(2r; \frac{640n}{b\gamma}\right).$$

Proof. κ_r satisfies the recurrence

$$\kappa_0 = 2,$$
$$\kappa_r = \left(\frac{b\gamma}{320n}\right) 2^{2^{\frac{320n}{b\gamma}\kappa_{r-1}}},$$

so it suffices to show that

$$\lambda_r \leq \left(\frac{b\gamma}{320n}\right) 2^{2^{\frac{320n}{b\gamma}\lambda_{r-1}}},$$

which follows from

$$\begin{aligned}
\lambda_r &= 4\lambda_{r-1}\left(\frac{4n}{b}\right)^{2^{\frac{160n}{b\gamma}\lambda_{r-1}}} \\
&= \left(\frac{b\gamma}{320n}\right)\left(\frac{1280n\lambda_{r-1}}{b\gamma}\right)\left(\frac{4n}{b}\right)^{2^{\frac{160n}{b\gamma}\lambda_{r-1}}} \\
&\le \left(\frac{b\gamma}{320n}\right)\left(\frac{5120n^2\lambda_{r-1}}{b^2\gamma}\right)^{2^{\frac{160n}{b\gamma}\lambda_{r-1}}} \\
&= \left(\frac{b\gamma}{320n}\right)2^{\log\left(\frac{5120n^2\lambda_{r-1}}{b^2\gamma}\right)2^{\frac{160n}{b\gamma}\lambda_{r-1}}} \\
&\le \left(\frac{b\gamma}{320n}\right)2^{2^{\frac{320n}{b\gamma}\lambda_{r-1}}}. \qquad \square
\end{aligned}$$

Using the proposition, and the assumption about b, for n sufficiently large we have

$$\log^{(2r-2)}\lambda_{r-1} \le \frac{640n}{b\gamma} \le \frac{640}{a(n)\gamma}\log^{(2r(n)-1)} n < \frac{1}{2}\log^{(2r(n)-1)} n,$$

as required to complete the proof of the second part of the theorem.

For the first part of the theorem, it suffices to note that if $r(n) \le \frac{1}{2}\log^* n - \Delta$ for $\Delta = \log^*\log^* n$, then $r(n)^2 = o(\log^{(2r(n)-1)} n)$. This follows by taking $\log^{(\Delta)}$ of both sides: $\log^{(\Delta)}(r(n)^2) \le 2$ while $\log^{(\Delta)}(\log^{(2r(n)-1)} n) \ge \mathrm{T}(\Delta; 2)$. Hence we can choose $b(n) = o(n)$ so that it satisfies the hypothesis and, hence, the conclusion of the second part of the theorem. $\square$

So it remains to prove Lemmas 11, 12, and 13.

3.1. Proof of Lemma 11. Let f be a γ-nontrivial function on n variables. We want to show that for b in the given range, a "large" fraction of the sets of size b are γ-powerful. In light of Proposition 6, we may assume that f is monotone.

Fix $\gamma \in (0, 1/2)$. We first describe a stochastic process for selecting a sequence of variables $v_1, v_2, \ldots, v_d$ for an integer d to be specified, and show that with probability at least $1/2$, the process produces a set of variables that is γ-powerful. The process depends on a parameter s, which we will also specify later. Having selected the first k of these variables $v_1, \ldots, v_k$, let f_k denote the (monotone) Boolean function on $n-k$ variables obtained by setting each v_i to 1. We then select v_{k+1} as follows:

1. If there is a variable v whose influence in f_k is at least 2^{-s}, let v_{k+1} be such a variable of lowest index.
2. Otherwise, choose v_{k+1} uniformly at random from among the remaining $n-k$ variables.

We will establish the following claim.

CLAIM A. *Let n be sufficiently large and let $d \in [n]$ and $\gamma \in (0, 1/2)$, and suppose that $\gamma d \ge \frac{160n}{\log n}$. Let s, the parameter of the process, be $\frac{80n}{\gamma d}$. Then*

$$\Pr\Big[\{v_1, \ldots, v_d\} \textit{ is } \gamma\textit{-powerful in } f\Big] \ge 1/2.$$

Define random variables X_k and Z_k, for $i = 0, \ldots, d$, by

$$X_k = \begin{cases} 1 & \text{if } p^1_{f_{k-1}} \geq 1 - \gamma, \\ I^1_{f_{k-1}}(v_k) & \text{otherwise,} \end{cases}$$

$$Z_k = \sum_{i=1}^{k} X_i.$$

Claim A is easily deduced from the following two claims. In both claims, n, d, γ, and s are as in Claim A.

CLAIM B. *If $Z_d \geq 1 - 2\gamma$, then $\{v_1, \ldots, v_d\}$ is γ-powerful in f.*

CLAIM C. *Suppose $3 \leq s \leq \log(20n) - \log\log(20n)$. For each $k = 1, \ldots, d$,*

$$\mathbb{E}\Big[X_k \,\Big|\, X_0, \ldots, X_{k-1}\Big] \geq \frac{s\gamma}{20n}.$$

Assume Claims B and C. Let $s = 80n/\gamma d$. Since $\gamma d \geq 160n/\log n$, and $d \leq n$, s satisfies the hypothesis of Claim C and therefore

$$\mathbb{E}[Z_d] \geq \frac{ds\gamma}{20n} \geq 4.$$

Applying Lemma 9 with $\mu = 4$ and $\delta = 3/4$ gives

$$\Pr[Z_d < 1] \leq e^{-9/8};$$

now applying Claim B yields the conclusion of Claim A.

To prove Claim B, assume that $Z_d \geq 1 - 2\gamma$. It suffices to show that $p^1_{f_d} \geq 1 - \gamma$, since this is equivalent to $\{v_1, \ldots, v_d\}$ being γ-powerful. Since $p^1_{f_k}$ is nondecreasing in k, we may assume that $p^1_{f_k} < 1 - \gamma$ for $k < d$. Then, recalling the definition of Z_d,

$$Z_d = \sum_{k=1}^{d} I_{f_{k-1}(v_k)}.$$

Now for each $k \geq 1$, $p^1_{f_k} = p^1_{f_{k-1}} + I^1_{f_{k-1}}(v_k)$, and hence $p^1_{f_d} = p^1_f + Z_d$. Since $p^1_f \geq \gamma$ by hypothesis, $p^1_{f_d} \geq 1 - \gamma$, as required for Claim B.

Since v_1, v_2, …, v_{k-1} determine X_0, …, X_{k-1}, Claim C follows if we show

$$\mathbb{E}[X_k | v_1, \ldots, v_{k-1}] \geq \frac{s\gamma}{20n}.$$

If $p_{f_{k-1}} \geq 1-\gamma$, then X_k is identically 1. Otherwise, $X_k = I_{f_{k-1}}(v_k)$. If v_k was selected by rule 1, then $X_k \geq 2^{-s}$, which is at least $s\gamma/(20n)$ for $s \leq \log(20n) - \log\log(20n)$. If rule 2 was used to select v_k, Lemma 7 gives the desired conclusion. This establishes Claim C and thus Claim A.

We now complete the proof of Lemma 11. The idea is that there are few variables chosen by rule 1, so with nonnegligible probability a random set of variables will contain them all.

More specifically, the hypothesis of the lemma implies that if we set $d = \lfloor b/2 \rfloor$, then d and γ satisfy the hypothesis of Claim A. We will use Claim A to show that a random subset of size b is γ-powerful with the required probability. Choose $s = 80n/(\gamma d)$ in accordance with Claim A, and observe that $s \leq \frac{1}{2}\log n$.

First, we reformulate the selection process for $v_1, v_2, \ldots, v_d$ in such a way that all random selections are made at the beginning of the process. We select a pair (S, σ), where S is a set of d variables chosen uniformly at random and σ is a bijection from $[d]$ to S chosen uniformly at random from the $d!$ such maps. Then $\vec{v}(S, \sigma) = (v_1, \ldots, v_d)$ is selected as above, except that rule 2 is replaced by "let i be the least integer such that $\sigma(i)$ is not a member of $\{v_1, \ldots, v_{k-1}\}$ and set $v_k = \sigma(i)$." It is easy to see that this process generates the same distribution over sequences $v_1, \ldots, v_d$ as the original process. Let $R_1(S, \sigma)$ be the set of v_i's chosen according to rule 1, and let $R_2(S, \sigma)$ be the set of those v_i's chosen according to rule 2. Obviously, $R_2(S, \sigma) \subseteq S$. Also, $|R_1(S, \sigma)| \leq 2^s$, since $I_{f_{k-1}}(v_k) \geq 2^{-s}$ if $v_k \in R_1(S, \sigma)$ and $\sum_{k=1}^{d} I_{f_{k-1}}(v_k) \leq p^1_{f_d} \leq 1$.

Now, consider a randomly chosen set B of size b. We view the probability space for B as consisting of triples (S, σ, T), where S and σ are as above and T is a random subset of size $b - d$ of $[n] \setminus S$. The set B is $S \cup T$. For B to be γ-powerful, it suffices that (i) $\vec{v}(S, \sigma)$ is γ-powerful and (ii) $R_1(S, \sigma) - S \subseteq T$, since then B contains the γ-powerful set $R_1(S, \sigma) \cup R_2(S, \sigma)$. By Claim A, event (i) occurs with probability at least $1/2$. Now, for any $S_0 \subset [n]$ of size b, we may estimate the probability of event (ii) conditioned on $S = S_0$: since $R_1(S, \sigma) - S$ has size at most $2^s = 2^{80n/\gamma d} \leq b/4$, and T is a random subset of $[n] \setminus S$ of size $b - d \geq b/2$, the probability that T contains $R_1(S, \sigma) - S$ is at least

$$\left(\frac{b-d}{n-d}\right)\left(\frac{b-d-1}{n-d-1}\right)\cdots\left(\frac{b-d-\lfloor 2^s\rfloor+1}{n-d-\lfloor 2^s\rfloor+1}\right) \geq \left(\frac{b}{4n}\right)^{2^s} \geq \left(\frac{b}{4n}\right)^{2^{80n/\gamma d}}.$$

Then $\Pr[\text{event (ii)} \mid \text{event (i)}] \geq \left(\frac{b}{4n}\right)^{2^{80n/\gamma d}}$, from which follows the statement of the lemma.

This completes the proof of Lemma 11.

3.2. Proof of Lemma 12. We first give a modified (and slightly more general) formulation of the lemma which will make the exposition a bit clearer.

LEMMA 16. *Fix n. If $\langle \delta_1, b_1 \rangle \in C_n(r_1; \frac{\alpha_1 \delta_2}{2}, \beta_1)$ and $\langle \delta_2, b_2 \rangle \in C_n(r_2; \alpha_2, \beta_2)$, then*

$$\left\langle \frac{\delta_1 \delta_2}{2}, b_1 + b_2 \right\rangle \in C_n(r_1 + r_2; \alpha_1 + \alpha_2, \beta_1 + \beta_2).$$

To deduce Lemma 12 from this, suppose that δ_1, b_1, r_1, γ_1, δ_2, b_2, r_2, and γ_2 are given satisfying the hypotheses of Lemma 12. Apply the above lemma with the same δ_i, b_i, and r_i, and with $\alpha_1 = 2\gamma_1/\delta_2$, $\beta_1 = \gamma_1$, and $\alpha_2 = \beta_2 = \gamma_2$.

So we prove Lemma 16.

Proof. Let $\Pi : (\{0,1\}^n)^{r_1 + r_2} \to \{0,1\}$ be an $(n, r_1 + r_2)$-protocol with $p^1_\Pi \geq \alpha_1 + \alpha_2$. We want to lower bound the probability that a (uniformly) random subset B of $[n]$ of size $b_1 + b_2$ is $\beta_1 + \beta_2$-powerful in Π.

A random subset B of size $b_1 + b_2$ can be selected by selecting subsets B_1, B_2, C, where B_1 is a uniformly random subset of size b_1, B_2 is a uniformly random subset of size b_2, and C is a uniformly random subset of $n - (B_1 \cup B_2)$ of size $b_1 + b_2 - |B_1 \cup B_2|$. Clearly, the probability that $B = B_1 \cup B_2 \cup C$ is $\beta_1 + \beta_2$-powerful is at least the probability that $B_1 \cup B_2$ is $\beta_1 + \beta_2$-powerful, so we lower bound this latter probability.

To do this, we define an event V that implies that $B_1 \cup B_2$ is $\beta_1 + \beta_2$-powerful and such that $\Pr_{B_1, B_2}[V]$ can be analyzed.

The input to Π is a vector in $(\{0,1\}^n)^{r_1 + r_2}$. Fixing the outcome of the first r_1 rounds to $\vec{\sigma} \in (\{0,1\}^n)^{r_1}$ gives rise to an (n, r_2)-protocol $\Pi[\vec{\sigma}] : (\{0,1\}^n)^{r_2} \to \{0,1\}$

by assigning

$$\Pi[\vec{\sigma}](\vec{\tau}) = \Pi(\sigma^1, \ldots, \sigma^{r_1}, \tau^1, \ldots, \tau^{r_2}).$$

Then $p^1_{\Pi[\vec{\sigma}]}$ can be viewed as a function of $\vec{\sigma}$. Let $\mathcal{E}$ be the set of those $\vec{\sigma}$ for which $p^1_{\Pi[\vec{\sigma}]} \geq \alpha_2$.

For $B_2 \subset [n]$, let $\mathcal{E}_{B_2}$ be the set of all $\vec{\sigma} \in \mathcal{E}$ such that B_2 is β_2-powerful with respect to the protocol $\Pi[\vec{\sigma}]$, i.e.,

$$\mathcal{E}_{B_2} = \left\{ \vec{\sigma} \in \mathcal{E} \ : \ p^1_{\Pi[\vec{\sigma}]}(B_2) \geq 1 - \beta_2 \right\}.$$

For each $B_2 \subseteq [n]$ of size b_2, let $\hat{\Pi}_{B_2}$ be the (n, r_1)-protocol $\hat{\Pi} = \hat{\Pi}_{B_2}$ given by

$$\hat{\Pi}_{B_2}(\vec{\sigma}) = \begin{cases} 1 & \text{if } \vec{\sigma} \in \mathcal{E}_{B_2}, \\ 0 & \text{otherwise.} \end{cases}$$

We now define V to be the event (depending on B_1 and B_2) that B_1 is β_1-powerful in $\hat{\Pi}_{B_2}$.

First we show that V implies that $B_1 \cup B_2$ is $\beta_1 + \beta_2$-powerful in Π. Consider the following two-step strategy for $B_1 \cup B_2$: (i) For the first r_1 rounds, B_1 plays so as to maximize the probability that $\vec{\sigma} \in \mathcal{E}_{B_2}$. Assuming this is successful then (ii) during the next r_2 rounds, B_2 tries to force the outcome of Π to be 1. The probability that this strategy fails is at most the sum of the probability that (i) fails and that (ii) fails given that (i) succeeds. The probability that (i) fails is at most β_1 by the definition of V. Assuming that (i) succeeds, the probability that (ii) fails is at most β_2 by the definition of the relation $\mathcal{E}_{B_2}$. Thus, given V, $B_1 \cup B_2$ is $\beta_1 + \beta_2$-powerful.

It remains to show that $\Pr[V] \geq \delta_1\delta_2/2$. To do this we consider, for $\eta > 0$, the event U_η (depending on B_2 alone) that $\Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}_{B_2}] \geq \eta$. We will show that when $\eta = \frac{\alpha_1\delta_2}{2}$, $\Pr[U_\eta] \geq \delta_2/2$ and $\Pr[V|U_\eta] \geq \delta_1$, which immediately gives the desired lower bound on $\Pr[V]$.

First we lower bound $\Pr[U_\eta]$. For fixed B_2 we have

$$\Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}_{B_2}] = \Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}] \times \frac{|\mathcal{E}_{B_2}|}{|\mathcal{E}|}. \tag{3.2}$$

By the definition of $\mathcal{E}$,

$$\mathop{\mathbb{E}}_{\vec{\sigma}}[p^1_{\Pi[\vec{\sigma}]}] \leq \Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}] + (1 - \Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}])\alpha_2 \leq \Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}] + \alpha_2.$$

We also have $\mathbb{E}_{\vec{\sigma}}[p^1_{\Pi[\vec{\sigma}]}] = p^1_\Pi \geq \alpha_1 + \alpha_2$, and thus

$$\Pr_{\vec{\sigma}}[\vec{\sigma} \in \mathcal{E}] \geq \alpha_1. \tag{3.3}$$

Letting $W = W(B_2)$ denote the random variable $|\mathcal{E}_{B_2}|/|\mathcal{E}|$ and combining (3.3) and (3.2), we have

$$\Pr_{B_2}[U_\eta] \geq \Pr_{B_2}[W \geq \eta/\alpha_1].$$

So we lower bound this latter probability. For $\sigma \in \mathcal{E}$, the protocol $\Pi[\vec{\sigma}]$ is an (n, r_2) protocol that is α_2-nontrivial. Thus, by the hypothesis of the lemma, for any $\vec{\sigma} \in \mathcal{E}$,

$$\Pr_{\substack{B_2 \subset [n] \\ |B_2| = b_2}} \left[\vec{\sigma} \in \mathcal{E}_{B_2} \right] \geq \delta_2.$$

Summing over $\sigma \in \mathcal{E}$ and dividing by $|\mathcal{E}|$ we obtain $\mathbb{E}_{B_2}[W] \geq \delta_2$. Since $W \in [0,1]$, we also have $\mathbb{E}_{B_2}[W] \leq \Pr_{B_2}[W \geq \eta/\alpha_1] + \eta/\alpha_1$, which implies $\Pr_{B_2}[W \geq \eta/\alpha_1] \geq \delta_2 - \eta/\alpha_1$. Setting $\eta = \alpha_1\delta_2/2$ we have $\Pr_{B_2}[U_{\alpha_1\delta_2/2}] \geq \Pr_{B_2}[W \geq \delta_2/2] \geq \delta_2/2$ as required.

Finally, we lower bound $\Pr[V|U_\eta]$. V is the event that B_1 is β_1-powerful in $\hat{\Pi}_{B_2}$. The event U_η implies that the protocol $\hat{\Pi}_{B_2}$ is η-nontrivial, and for $\eta = \alpha_1\delta_2/2$, the hypothesis of the lemma implies that the probability that V occurs is at least δ_1. ∎

3.3. Proof of Lemma 13. Fix b, n, and γ as hypothesized. Let $H(r)$ denote the hypothesis $\gamma b \geq 800 n \lambda_{r-1}/\log n$, and let $C(r)$ denote the conclusion

$$\left\langle \frac{1}{\lambda_r}, rb \right\rangle \in C_n(r; r\gamma).$$

We want to show that $H(r)$ implies $C(r)$ for all $r \geq 1$. We proceed by induction on r.

The basis case is immediate from Lemma 11. For the induction step, let $r \geq 1$, and suppose that $H(r)$ implies $C(r)$. Assume $H(r+1)$ is true; we want to show $C(r+1)$ holds. Now $H(r+1)$ implies $H(r)$ since $\lambda_r \geq \lambda_{r-1}$, and hence $C(r)$ holds. If $\gamma' \in (0, 1/2)$ is such that $\gamma' b \geq 400 n/\log n$, then Lemma 11 implies

$$\langle \delta(n, b, \gamma'), b \rangle \in C_n(1; \gamma').$$

Combining this and $C(r)$ using Lemma 12, and setting $\gamma' = \frac{\gamma}{2\lambda_r}$, gives

$$\left\langle \frac{\delta(n, b, \gamma')}{2\lambda_r}, (r+1)b \right\rangle \in C_n(r+1, (r+1)\gamma),$$

which is equivalent to $C(r+1)$.

This completes the proof of Lemma 13 and the proof of the main theorem.

4. Extensions to protocols with longer messages. We now indicate how to generalize the bounds proven above to protocols which permit players to send longer messages. Recall that for $n, r \in \mathbb{N}$ and $\vec{\ell} = (\ell_1, \ldots, \ell_r) \in \mathbb{N}^r$, we say that Π is a $(n, r, \vec{\ell})$-protocol if n is the number of players, r is the number of rounds, and no more than ℓ_k bits are broadcast by each player in the kth round.

We extend Definition 10 to account for variable message lengths.

DEFINITION 17. *Let $C_n^{\vec{\ell}}(r; \alpha, \beta)$ (written $C_n^{\vec{\ell}}(r; \gamma)$ when $\alpha = \beta = \gamma$) denote the collection of pairs $\langle \delta, b \rangle$ so that for any $(n, r, \vec{\ell})$-protocol Π that is α-nontrivial, at least a δ fraction of sets $B \subset [n]$ of size b are β-powerful in Π.*

We begin by considering a single-round protocol $f : (\{0,1\}^\ell)^n \to \{0,1\}$ in which each player broadcasts ℓ bits. Simply treating f as a function on $n\ell$ Boolean variables and examining the stochastic process of Section 3.1 yields the following version of Claim A.

CLAIM D (cf. Claim A). *Let $n\ell$ be sufficiently large and let $d \in [n]$ and $\gamma \in (0, 1/2)$, and suppose that $\gamma d \geq \frac{160n\ell}{\log(n\ell)}$. Let s, the parameter of the process, be $\frac{80n\ell}{\gamma d}$. Then*

$$\Pr\Big[\{v_1, \ldots, v_d\} \text{ is } \gamma\text{-powerful in } f\Big] \geq 1/2.$$

If the Boolean variables $\{v_1, \ldots, v_d\}$ are γ-powerful in $f : \{0,1\}^{n\ell} \to \{0,1\}$, then the $\{0,1\}^\ell$-valued variables $\{x \mid \exists i, v_i \text{ is a component of } x\}$ are γ-powerful in f, again viewed as a function on $(\{0,1\}^\ell)^n$. Observe that applying Claim A in this way does not exploit the fact that each player controls many bits of the function f. The proof of Lemma 11 now yields the following lemma.

LEMMA 18 (cf. Lemma 11). *Let $n, \ell \in \mathbb{N}$ and $\gamma \in (0, \frac{1}{2})$ and $b \leq n$, and assume $\gamma b \geq 400n\ell/\log(n\ell)$. Then $\langle \delta, b\rangle \in C_n^{(\ell)}(1;\gamma)$, where*

$$\delta = \delta(n, b, \ell, \gamma) = \frac{1}{2}\left(\frac{b}{4n}\right)^{2^{\frac{80n\ell}{\gamma b}}}.$$

The number of bits broadcast per round is immaterial to the proof of Lemma 12; restating that lemma for multibit protocols yields the following lemma.

LEMMA 19 (cf. Lemma 12). *Fix n. If $\langle \delta_1, b_1\rangle \in C_n^{\vec{\ell}}(r_1;\gamma_1)$ and $\langle \delta_2, b_2\rangle \in C_n^{\vec{m}}(r_2;\gamma_2)$, then*

$$\left\langle \frac{\delta_1\delta_2}{2}, b_1 + b_2 \right\rangle \in C_n^{(\vec{\ell},\vec{m})}\left(r_1 + r_2; \frac{2\gamma_1}{\delta_2} + \gamma_2\right),$$

where $(\vec{\ell}, \vec{m})$ denotes the vector $(\ell_1, \ldots, \ell_{r_1}, m_1, \ldots, m_{r_2})$.

We combine these to prove the following lemma.

LEMMA 20 (cf. Lemma 13). *Let $b \leq n$, $\gamma \in (0, 1/2)$, and $l_i \in \{1, 2, \ldots\}$ for each $i \geq 0$. Define $\lambda_0 = \frac{1}{2}$, and for $r \geq 1$ define*

$$\lambda_r = 4\lambda_{r-1}\left(\frac{4n}{b}\right)^{2^{\frac{160nl_{r-1}}{\gamma b}\lambda_{r-1}}}.$$

Assume that for each $r \geq 1$, $\lambda_r l_r \geq \lambda_{r-1} l_{r-1}$. Then, if $\gamma b \geq 800nl_{r-1}\lambda_{r-1}/\log n$,

$$\left\langle \frac{1}{\lambda_r}, rb \right\rangle \in C_n^{\vec{\ell}}(r; r\gamma),$$

where $\ell_i = l_{r-i}$, so $\vec{\ell} = (\ell_1, \ldots, \ell_r) = (l_{r-1}, \ldots, l_0)$.

Proof. Fix b, n, γ, and l_i as hypothesized. Let $H(r)$ denote the hypothesis $\gamma b \geq 800nl_{r-1}\lambda_{r-1}/\log n$, and let $C(r)$ denote the conclusion

$$\left\langle \frac{1}{\lambda_r}, rb \right\rangle \in C_n^{\vec{\ell}}(r; r\gamma),$$

where $\vec{\ell} = (l_{r-1}, \ldots, l_0)$. We want to show that $H(r)$ implies $C(r)$ for all $r \geq 1$. We proceed by induction on r.

The basis case is immediate from Lemma 18. For the induction step, let $r \geq 1$, and suppose that $H(r)$ implies $C(r)$. Assume $H(r+1)$ is true; we want to show that

$C(r+1)$ holds. Now $H(r+1)$ implies $H(r)$ since, by assumption, $\lambda_r l_r \geq \lambda_{r-1} l_{r-1}$, and hence $C(r)$ holds. If $\gamma' \in (0, 1/2)$ is such that $\gamma' b \geq 400 n l_r / \log n$, then Lemma 18 implies that

$$\langle \delta(n, b, l_r, \gamma')), b \rangle \in C_n^{(l_r)}(1; \gamma').$$

Combining this and $C(r)$ using Lemma 19, and setting $\gamma' = \frac{\gamma}{2\lambda_r}$, gives

$$\left\langle \frac{\delta(n, b, l_r, \gamma')}{2\lambda_r}, (r+1)b \right\rangle \in C_n^{\vec{\ell}}(r+1, (r+1)\gamma),$$

where $\vec{\ell} = (l_r, \ldots, l_0)$, which is equivalent to $C(r+1)$. □

This may be applied to prove Theorem 4.

Proof of Theorem 4. Fix n. Set $\alpha = \frac{1}{\log^* n}$ and define $\gamma = \alpha^2$, $b = \lceil \alpha^2 n \rceil$, and, for $i \in \{0, \ldots, r-1\}$,

$$l_i = \max\left(1, \left\lfloor \frac{\alpha\gamma b (\log^{(2(r-i)-1)} n)^{1-\alpha}}{800n} \right\rfloor\right). \tag{4.1}$$

Note that

$$\begin{aligned} l_o &\geq \frac{\alpha^5 (\log^{(2r-1)} n)^{1-\alpha}}{800} = \frac{(\log^{(\log^* n - 2\log^* \log^* n - 1)} n)^{1-\alpha}}{800(\log^* n)^5} \\ &= \frac{(T(2\log^* \log^* n - 1; 1))^{1-o(1)}}{800(\log^* n)^5} = (\log^* n)^{\omega(1)} \end{aligned}$$

so that, when n is sufficiently large, $\gamma < \frac{1}{2}$ and $l_{r-1} \geq \cdots \geq l_0 > 1$. In this case, with λ_i defined as in Lemma 20,

$$\begin{aligned} \lambda_i = 4\lambda_{i-1}\left(\frac{4n}{b}\right)^{2^{\frac{160 n l_{i-1}\lambda_{i-1}}{b\gamma}}} &= 2^{\log 4 + \log \lambda_{i-1} + 2^{\left(\frac{160 n l_{i-1}\lambda_{i-1}}{b\gamma} + \log\log \frac{4n}{b}\right)}} \\ &\leq 2^{2^{\frac{160 n l_{i-1}\lambda_{i-1}}{b\gamma} + \log\log 4 + \log\log \lambda_{i-1} + \log\log \frac{4n}{b}}} \end{aligned}$$

and, as $\max(\log\log 4, \log\log \lambda_{i-1}, \log\log(4n/b)) < 160 n l_{i-1}\lambda_{i-1}/\gamma b$,

$$\lambda_i \leq 2^{2^{\frac{640 n l_{i-1}\lambda_{i-1}}{b\gamma}}} \leq 2^{2^{\alpha(\log^{(2(r-i)+1)} n)^{1-\alpha}\lambda_{i-1}}}.$$

As noted above, these l_i are monotonically increasing (in i) and therefore satisfy the hypothesis of Lemma 20. We show that for sufficiently large n, $\lambda_i \leq (\log^{(2(r-i)-1)} n)^\alpha$. Since this is clearly true for λ_0, by induction

$$\lambda_i \leq 2^{2^{\alpha(\log^{(2(r-i)+1)} n)^{1-\alpha}\lambda_{i-1}}} \leq 2^{2^{\alpha(\log^{(2(r-i)+1)} n)}} \leq (\log^{(2(r-i)-1)} n)^\alpha, \tag{4.2}$$

where we have applied the inequality $x^\epsilon \leq \epsilon x$, valid when, for example, $x \geq 4$ and $\epsilon \in [1/\sqrt{x}, 1]$. (We apply the inequality with $\epsilon = \alpha$ and $x = \log^{(2(r-i))} n$; both these requirements are met for sufficiently large n.)

Finally, from (4.1) and (4.2) above,

$$\frac{800 n l_{r-1}\lambda_{r-1}}{\log n} \leq \alpha\gamma b \leq \gamma b,$$

so that Lemma 20 applies. This asserts the existence of an $r\gamma = o(1)$-powerful set of $rb = o(n)$ players for any protocol Π under the following assumptions:

- Π is $r\gamma = o(1)$-nontrivial,
- Π lasts for r rounds, with $r \leq \frac{1}{2}\log^* n - \log^* \log^* n$, and
- Π calls for no more than

$$l_{r-k} = \Omega\left(\frac{(\log^{(2k-1)} n)^{(1-\alpha)}}{\text{poly}(\log^* n)}\right) = (\log^{(2k-1)} n)^{(1-O(\alpha))}$$

communication bits in the kth round. □

5. The influence of large coalitions. Applying the results of [13], one can show that, for a Boolean function f with $p_f^1 = 1/2$ and $b(n) = \Theta(n)$, there is always a coalition L of size $b(n)$ for which $p_f^0(L) \geq 1 - 1/n^c$ for some appropriate constant c (depending on b). When $b(n) \geq n/2$, however, the following observation from [17] may be applied.

PROPOSITION 21. *Let X be a finite probability space and $f : X^n \to \{0,1\}$. Let $A_1, A_2 \subset [n]$ be a partition of the variables on which f is defined (so that $A_1 \cup A_2 = [n]$ and $A_1 \cap A_2 = \emptyset$). Then for at least one of these two sets, A_i,*

$$p_f^1(A_i) = 1 \qquad \text{or} \qquad p_f^0(A_i) = 0.$$

Below we observe that near this $\frac{n}{2}$ threshold (specifically, for $b(n) > (1/3 + \epsilon)n$), the above bound of [13] may be improved to $1 - 1/\exp(\Omega(n))$.

In preparation for the lemma, we record a Chernoff bound (see, e.g., [3]).

LEMMA 22. *Let $X_i, i = 1, \ldots, n$, be independent random variables, each uniformly distributed in $\{0,1\}$. Then*

$$\Pr\left[\sum_i X_i - \frac{n}{2} > a\right] < \exp\left(-\frac{a^2}{2n}\right).$$

THEOREM 23. *Let $\gamma > \frac{1}{3}$. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and let $\mathfrak{B} = \{B \subset [n] : |B| = \lceil \gamma n \rceil\}$. If $p_f^1(B) < 1$ for all $B \in \mathfrak{B}$, then for all $B \in \mathfrak{B}$, $p_f^0(B) \geq 1 - \epsilon$, where*

$$\epsilon = \exp\left(-\frac{(1-3\gamma)^2}{8(1-\gamma)}n\right).$$

Proof. Assume that f is monotone. Recall that a min-term of a monotone function f is a minimal subset of variables which, if set to 1, ensures that $f = 1$. If f has a min-term of cardinality at most γn, then clearly there is $B \in \mathfrak{B}$ for which $p_f^1(B) = 1$. Otherwise all min-terms have cardinality larger than γn. Fix $B \in \mathfrak{B}$ and consider an input $\vec{x} = x_1 \ldots x_n$, where each x_i, for $i \notin B$, is chosen independently at random in $\{0,1\}$, and $x_i = 0$ for $i \in B$. Then $\mathbb{E}[\sum_i x_i] \leq \frac{1-\gamma}{2}n$, so that by applying the above Chernoff bound,

$$\Pr\left[\sum_i x_i > \gamma n\right] < \exp\left(-\frac{(3\gamma-1)^2}{8(1-\gamma)}n\right).$$

Then $p_f^0(B) > 1 - \exp\left(-\frac{(3\gamma-1)^2}{8(1-\gamma)}n\right)$, as desired. □

6. Open problems. We summarize the known results concerning protocols that are resilient against a linear number of corrupt players:

1. By [16] there is an $(n, \big[1+o(1)\big]\log n, 1)$-protocol which is $\Omega(n)$-resilient. By Theorem 3, there is no $(n, (1/2-\epsilon)\log^* n, 1)$-protocol that is $\Theta(n)$-resilient.
2. By [16], there is an $(n, \log^* n + O(1), \vec{\ell})$-protocol, where $\ell_k = O(\log^{(k)} n)$, that is $\Omega(n)$-resilient. By Theorem 4, there is no $(n, (1/2-o(1))\log^* n, \vec{\ell})$-protocol that is $\Theta(n)$-resilient for some $\ell_k = (\log^{(2k-1)} n)^{1-o(1)}$.
3. It is not difficult to show that Theorem 2 actually implies that there can be no $(n, 1, o(\log n))$-protocol that is $\Theta(n)$-resilient.

These suggest several avenues of investigation:

1. In the case where each player sends a single bit per round (item 1 above), $\big[1+o(1)\big]\log n$ rounds are sufficient to guarantee $\Omega(n)$-resilience, $\big[1/2 - o(1)\big]\log^* n$ rounds are necessary—what is the right answer?
2. In the general case (item 2 above), can Theorem 4 be strengthened to show any $\Omega(n)$-resilient protocol has some round k during which $\Omega(\log^{(k)} n)$ communication occurs?
3. From (3) above, no one-round protocol using $o(\log n)$ bits per player can be $\Omega(n)$-resilient. Even abandoning all constraints on the number of bits sent per round, is there a one (or even constant) round $\Omega(n)$-resilient protocol?
4. We have focused on protocols where honest players flip a fair coin; what can be said when the honest players' coin flips are biased?

Acknowledgments. We thank Nati Linial for pointing out the failure of iterative methods in the multibit case and several illuminating discussions. We also thank Uri Feige for suggesting our last open question and for useful discussions.

REFERENCES

[1] M. Ajtai and N. Linial, *The influence of large coalitions*, Combinatorica, 13 (1993), pp. 129–145.
[2] N. Alon and M. Naor, *Coin-flipping games immune against linear-sized coalitions*, SIAM J. Comput., 22 (1993), pp. 403–417.
[3] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.
[4] K. Azuma, *Weighted sums of certain dependent random variables*, Tôhoku Math. J. (2), 19 (1967), pp. 357–367.
[5] M. Ben-Or and N. Linial, *Collective coin flipping, robust voting schemes and minima of Banzhaf values*, in 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, 1985, IEEE, pp. 408–416.
[6] M. Ben-Or and N. Linial, *Collective coin flipping*, in Randomness and Computation, S. Micali, ed., Academic Press, New York, 1990, pp. 91–115.
[7] M. Ben-Or, N. Linial, and M. Saks, *Collective coin flipping and other models of imperfect randomness*, in Proceedings of the Seventh Hungarian Colloquium on Combinatorics, Colloq. Math. Soc. János Bolyai 52, A. Hajnal, L. Lovász, and V. Sós, eds., North-Holland, Amsterdam, 1988, pp. 75–112.
[8] H. Chernoff, *A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Statistics, 23 (1952), pp. 493–507.
[9] U. Feige, *Noncryptographic selection protocols*, in 40th Annual Symposium on Foundations of Computer Science, 1999, IEEE, pp. 142–152.
[10] E. Friedgut and G. Kalai, *Every monotone graph property has a sharp threshold*, Proc. Amer. Math. Soc., 124 (1996), pp. 2993–3002.
[11] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, 2nd ed., Cambridge University Press, Cambridge, UK, 1952.
[12] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[13] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions (extended abstract)*, in 29th Annual Symposium on Foundations of Computer Science, White Plains, NY, 1988, IEEE, pp. 68–80.

[14] N. Linial, *Game-theoretic aspects of computing*, in Handbook of Game Theory with Economic Applications, Vol. II, R. J. Aumann and S. Hart, eds., North-Holland, Amsterdam, 1994, pp. 1339–1395.

[15] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[16] A. Russell and D. Zuckerman, *Perfect information leader election in* $\log^* n + O(1)$ *rounds*, J. Comput. System Sci., 63 (2001), pp. 612–626.

[17] M. Saks, *A robust noncryptographic protocol for collective coin flipping*, SIAM J. Discrete Math., 2 (1989), pp. 240–244.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1663–1686

HARDNESS OF APPROXIMATE HYPERGRAPH COLORING*

VENKATESAN GURUSWAMI†, JOHAN HÅSTAD‡, AND MADHU SUDAN§

Abstract. We introduce the notion of covering complexity of a verifier for probabilistically checkable proofs (PCPs). Such a verifier is given an input, a claimed theorem, and an oracle, representing a purported proof of the theorem. The verifier is also given a random string and decides whether to accept the proof or not, based on the given random string. We define the covering complexity of such a verifier, on a given input, to be the minimum number of proofs needed to "satisfy" the verifier on every random string; i.e., on every random string, at least one of the given proofs must be accepted by the verifier. The covering complexity of PCP verifiers offers a promising route to getting stronger inapproximability results for some minimization problems and, in particular, (hyper)graph coloring problems. We present a PCP verifier for NP statements that queries only four bits and yet has a covering complexity of one for true statements and a superconstant covering complexity for statements not in the language. Moreover, the acceptance predicate of this verifier is a simple not-all-equal check on the four bits it reads. This enables us to prove that, for *any* constant c, it is NP-hard to color a 2-colorable 4-uniform hypergraph using just c colors and also yields a superconstant inapproximability result under a stronger hardness assumption.

Key words. graph coloring, hypergraph coloring, hardness of approximations, PCP, covering PCP, set splitting

AMS subject classifications. 68Q17, 68Q15

PII. S0097539700377165

1. Introduction.

The study of probabilistically checkable proof (PCP) systems has led to major breakthroughs in theoretical computer science in the past decade. In particular, this study has led to a surprisingly clear understanding of the complexity of finding approximate solutions to optimization problems. A recurring theme in this study is the association of new complexity measures to verifiers of PCP systems and construction of efficient verifiers under the new measure. The new measures are then related to some special subclass of optimization problems to gain new insight about the approximability of problems in this subclass of optimization problems. This paper presents yet another such complexity measure, the *covering complexity* of a verifier, and relates it to a subclass of optimization problems, namely, *hypergraph coloring* problems. Below we elaborate on some of the notions above such as PCPs, approximability, and hypergraph coloring, and we introduce our new complexity measure.

PCPs. The centerpiece of a PCP system is the probabilistic verifier. This verifier is a randomized polynomial time algorithm whose input is a "theorem" and who is

*Received by the editors August 25, 2000; accepted for publication (in revised form) March 5, 2002; published electronically September 5, 2002. A preliminary version of this paper [15] appears in the *Proceedings of the* 41*st Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 149–158.

http://www.siam.org/journals/sicomp/31-6/37716.html

†Department of Computer Science, University of Washington, Seattle, WA 98195 (venkat@lcs.mit.edu). This work was done while this author was at MIT and was supported in part by NSF grants CCR-9875511 and CCR-9912342 and in part by an IBM Graduate Fellowship.

‡Department of Numerical Analysis and Computer Science, Royal Institute of Technology, SE-100 44 Stockholm, Sweden (johanh@nada.kth.se). This author was supported in part by the Göran Gustafsson Foundation and in part by NSF grant CCR-9987077.

§Laboratory for Computer Science, MIT, 200 Technology Square, Cambridge, MA 02139 (madhu@mit.edu). This author was supported in part by an MIT-NEC Research Initiation Award, a Sloan Foundation Fellowship, NSF Career grant CCR-9875511, NSF grant CCR-9912342, and MIT-NTT grant MIT 2001-04.

also given oracle access to a "proof." Using the traditional equivalence associated with randomized algorithms, it is convenient to think of the verifier as having two inputs: the "theorem" and a "random string." Based on these two inputs, the verifier settles on a strategy to verify the proof; namely, it decides on a sequence of queries to ask the oracle and prepares a predicate P. It then queries the oracle, and, if it receives as response bits $a_1, \ldots, a_q$, it applies the predicate $P(a_1, \ldots, a_q)$ and accepts iff the predicate is satisfied.[1] The quality of the PCP system is roughly related to its ability to distinguish valid proofs (true "theorems" with correct "proofs") from invalid theorems (incorrect "theorems" from any purported "proof"). Hopefully the verifier accepts valid proofs with much higher probability than it does invalid theorems.

To study the power of PCP systems in a complexity-theoretic setting, we quantify some of the significant resources of the verifier above and then study the resources needed to verify proofs of membership for some hard language. Fix such a language L, and consider a verifier V whose goal is to verify proofs of membership in L. The above paragraph already hints at four measures we may associate with such a verifier, and we define them in two steps. For functions $r, q : \mathbb{Z}^+ \to \mathbb{Z}^+$, we say that a V is (r, q)-restricted if, on input x (implying the theorem $x \in L$) of length n, V requires a random string of length $r(n)$ and makes $q(n)$ queries to the proof oracle. We say that V verifies L with completeness c and soundness s if (1) for every $x \in L$, there exists an oracle Π such that V, on input x and oracle access to Π, outputs accept with probability at least c, and (2) for every $x \notin L$ and every oracle Π, V outputs accept with probability at most s. The class of all languages L that have an (r, q)-restricted verifier verifying it with completeness c and soundness s is denoted $\mathrm{PCP}_{c,s}[r, q]$.

Covering complexity. In the variant of PCPs that we consider here, we stick with (r, q)-restricted verifiers but alter the notion of completeness and soundness. Instead of focusing on the one proof that maximizes the probability with which the verifier accepts a given input, here we allow multiple proofs to be provided to the verifier. We say that a set of proofs $\{\Pi_1, \ldots, \Pi_k\}$ *covers* a verifier V on input x if, for every random string, there exists one proof Π_i such that V accepts Π_i on this random string. We are interested in the smallest set of proofs that satisfy this property, and the cardinality of this set is said to be the *covering complexity* of the verifier on this input. Analogous to the class PCP, we may define the class $\mathrm{cPCP}_{c,s}[r, q]$ (for covering PCP) to be the class of all languages for which there exist (r, q)-restricted verifiers that satisfy the following conditions: (Completeness) If $x \in L$, the covering complexity of V on x is at most $1/c$. (Soundness) If $x \notin L$, then the covering complexity of V on x is at least $1/s$.

Notions somewhat related to covering complexity have been considered in the literature implicitly and explicitly in the past. Typically these notions have been motivated by the approximability of minimization problems, such as graph coloring, set cover, and the closest vector problem. Our specific notion is motivated by graph and hypergraph coloring problems. We describe our motivation next. We defer the comparison with related notions to later in this section.

Hypergraph coloring, approximability, and inapproximability. An l-uniform hypergraph H is given by a set of vertices V and a set of edges E, where an edge $e \in E$ is itself a subset of V of cardinality l. A k-coloring of H is a map

[1] This description of a verifier is somewhat restrictive. More general definitions allow the verifier to be adaptive, deciding on later queries based on responses to previous ones. For this paper, the restricted version suffices.

from V to the set $\{1, \ldots, k\}$ such that no edge is *monochromatic*. The hypergraph coloring problem is that of finding, given H, the smallest k for which a k-coloring of H exists. When $l = 2$, then the hypergraph is just a graph, and the hypergraph coloring problem is the usual graph coloring problem.

Graph and hypergraph coloring problems have been studied extensively in the literature from both the combinatorial and algorithmic perspectives. The task of determining if an l-uniform graph is k-colorable is trivial if $l = 1$ or $k = 1$, and almost so if $l = k = 2$. Every other case turns out to be NP-hard. The case of $l = 2$, $k \geq 3$ is a classical NP-hard problem, while the case of $k = 2$, $l \geq 3$ was shown to be NP-hard by Lovász [23]. Thus even the property of a hypergraph being 2-colorable is nontrivial. This property, also called *Property B*, has been studied in the extremal combinatorics literature for a long time. Much work has been done on determining sufficient conditions under which a hypergraph family is 2-colorable and on solving the corresponding algorithmic questions [11, 6, 7, 25, 26, 29, 27].

The hardness of the coloring problem motivates the study of the *approximability* of the graph and hypergraph coloring problems. In the context of these problems, an (l, k, k')-approximation algorithm is one that produces (in polynomial time) a k'-coloring of every k-colorable l-uniform hypergraph for some $k' > k$, with the "approximation" being better as k' gets closer to k. Even this problem turns out to be nontrivial, with the best known algorithms for coloring even 3-colorable graphs requiring $n^{\Omega(1)}$ colors [9, 19], where n is the number of vertices. Similarly, inspired in part by the approximate graph coloring algorithms, several works [1, 10, 22] have provided approximation algorithms for coloring 2-colorable hypergraphs. The best known result for 2-colorable 4-uniform hypergraphs is a polynomial time coloring algorithm that uses $\tilde{O}(n^{3/4})$ colors [1, 10].

To justify the intractability of the approximation versions of the hypergraph coloring problem, one looks for *inapproximability* results. Inapproximability results show that it is NP-hard to achieve the goals of an (l, k, k')-approximation algorithm by producing a polynomial time computable reduction from, say, satisfiability (SAT) to a "gap" problem related to hypergraph coloring. Here we assume a conservative definition of such a reduction, namely, the many-one reduction. The many-one version of such a reduction would reduce a formula φ to an l-uniform hypergraph H such that H is k-colorable if φ is satisfiable and H is not k'-colorable if φ is not satisfiable. Since the existence of an (l, k, k')-approximation algorithm now gives the power to decide if φ is satisfiable or not, this shows that the approximation problem is NP-hard. In what follows, when we say that an (l, k, k')-approximation problem is NP-hard, we always implicitly mean that the "gap version" of the problem is NP-hard.

This methodology combined with the PCP technique has been employed heavily to get hardness results of graph coloring problems. This approach started with the results of [24] and culminates with the essentially tight results of [12], which show that the $(2, n^{\varepsilon}, n^{1-\varepsilon})$-approximation problem is NP-hard under randomized reductions. However, for graphs whose chromatic number is a small constant, the known hardness results are much weaker. For example, for 3-colorable graphs, the best known hardness result rules out only coloring using four colors [20, 16]. This paper is motivated by the quest for strong (superconstant) inapproximability for coloring graphs whose chromatic number is a small constant. We do not get such results for graph coloring but do get such inapproximability results for hypergraph coloring and, in particular, for coloring 4-uniform hypergraphs.

Graph coloring and covering PCPs. In examining the reasons why the current techniques have been unable to show strong hardness results for inapproximability of coloring 3-colorable graphs, a natural question arises: Are PCPs really necessary to show such hardness results, or would something weaker suffice? To date there are no reasons showing that PCPs are necessary. And while the first result showing the intractability of coloring 3-colorable graphs with four colors [20] did use the PCP technique, [16] shows that PCPs are not needed in this result. The starting point of our work is the observation that *covering* PCPs are indeed necessary for showing strong hardness results for graph coloring. Specifically, in Proposition 2.1, we show that, if the $(2, c, \omega(1))$-approximation problem for coloring is NP-hard for $c < \infty$, then NP $\subseteq$ cPCP$_{\gamma,o(1)}[\log, 2]$ for some $\gamma > 0$. (Similar results can also be derived from hardness results for coloring hypergraphs, though we don't do so here.)

Previous approaches have realized this need implicitly but have relied on deriving the required results via PCPs. In particular, they use the trivial containment PCP$_{1,s}[r, q] \subseteq$ cPCP$_{1,s}[r, q]$ and build upon the latter result to derive hardness for coloring. (Notice that we do not have such a simple containment when the completeness parameter is not equal to 1. This special case of $c = 1$ is important in general and is referred to as *perfect completeness*.) For our purposes, however, this trivial containment is too weak. In particular, it is known that PCP$_{c,s}[\log, q] \subseteq$ P for every c, s, q such that $c > s2^q$ (cf. [8, Lemma 10.6]). Thus it is not possible to show NP $\subseteq$ cPCP$_{\gamma,o(1)}[\log, O(1)]$ for any constant $\gamma > 0$, using the trivial containment mentioned above (and such a covering PCP is essential for superconstant hardness results for coloring hypergraphs). Thus it becomes evident that a direct construction of covering PCPs may be more fruitful, and we undertake such constructions in this paper.

Related notions. Typically, every approach that applies PCP to minimization problems has resulted, at least implicitly, in some new complexity measures. Two of those that are close to, and likely to be confused with, the notion of covering complexity are the notions of "multiple assignments" [2] and the "covering parameter" of [12]. Here we clarify the distinctions.

In the former case, the multiple assignments of [2], the proof oracle is expected to respond to each query with an element of a large alphabet (rather than just a bit). When quantifying the "quality" of a proof, however, the oracle is allowed to respond with a subset of the alphabet, rather than just a single element, and the goal of the prover is to pick response sets of small sizes so that, on every random string, the verifier can pick one element from each response set to the different queries so that it leads to acceptance. Once again, we have a notion of covering all random strings with valid proofs, but this time the order of quantifiers is different. The notion of multiple assignments is interesting only when the alphabet of the oracles responses are large, while our notion remains interesting even when the oracle responds with an element of a binary alphabet.

The second related notion is the covering parameter of Feige and Kilian [12]. Since the names are confusingly similar (we apologize for not detecting this at an early stage), we refer to their notion as the FK-covering parameter. In a rather simplified sense, their parameter also allows multiple proofs to be presented to the verifier. However, their notion of coverage requires that, on every random string and every possible accepting pattern of query responses for the verifier, there should exist a proof which gives this accepting pattern (and is hence accepted). For any fixed verifier and input, the FK-covering number is always larger than ours since

we do not need every accepting pattern to be seen among the proofs. Though the notions appear close, the motivation, the application, and the technical challenges posed by the FK-covering parameter and ours are completely different. Both notions arise from an attempt to study graph coloring, but their focus is on general graphs (with high chromatic number), while ours is on graphs of small chromatic number. In their case, separation of the FK-covering parameter is sufficient but not necessary to give inapproximability of coloring. For our parameter, separation is necessary but not sufficient to get the same. Finally, in their constructions, the challenge is to take a traditional PCP and enhance it to have small FK-covering completeness, and they use the PCP directly to argue that the soundness is not large. In our case, the completeness is immediate, and the soundness needs further analysis.

Gadgets and covering complexity. Returning to our notion of covering complexity, while it seems essential to study this to get good hardness results on coloring, the reader should also be warned that this notion is somewhat less robust than usual notions that one deals with in PCPs. Specifically, prior notions were not very sensitive to the predicate applied by the verifier in deciding its final output. They could quantify the power of the verifier by simple parameters such as number of bits read or number of accepting configurations. Here we are forced to pay attention to the verifier's computations and restrict these to get interesting results. It is reasonable to ask why this happens, and we attempt to give some justification below.

In standard PCPs, it is often possible to use "gadgets" (whenever they are available) to convert the acceptance predicate of the verifier from one form to another for only a small loss in the performance. For example, suppose one has a PCP verifier V_1 that reads three bits of a proof and accepts if they are not all equal (NAE). Such a verifier would directly prove the hardness of the "Max NAE-3SAT" problem. However, by application of a gadget, the same verifier can be transformed into one that proves the hardness of the "Max 3SAT" problem. The gadget notices that the function $\mathrm{NAE}(a, b, c)$ for three Boolean variables a, b, c is simply $(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$, which is a conjunction of two 3SAT clauses. Thus a transformed verifier V_2 which picks three bits of the proof as V_1 does and then picks one of the two clauses implied by the check performed by V_1 and verifies just this one clause is now a verifier whose acceptance predicate is a 3SAT condition. Furthermore, if the acceptance probability of V_1 on the same proof is $1 - \alpha$, then the acceptance probability of V_2 on some given proof is exactly $1 - \alpha/2$. Thus, if V_1 proves inapproximability of Max NAE-3SAT, then V_2 proves inapproximability of Max 3SAT.

Unfortunately, a similar transformation does not apply in the case of covering complexity. Notice that two proofs, the oracle that always responds with 0 and the one that responds with 1, always suffice to cover any verifier whose acceptance predicate is 3SAT. Yet there exist NAE 3-SAT verifiers that cannot be covered by any constant number of proofs. (For example, the verifier that picks three of the n bits of the proof uniformly and independently at random and applies the NAE 3-SAT predicate to them needs $\Omega(\log n)$ proofs to be covered.) Thus, even though a gadget transforming NAE 3SAT to 3SAT does exist, it is of no use in preserving covering complexity of verifiers. This nonrobust behavior of covering PCP (cPCP) verifiers forces us to be careful in designing our verifiers, and our two results differ mainly in the predicate applied by the verifier.

Our results. Our first result is a containment of NP in the class $\mathrm{cPCP}_{1,\varepsilon}[O(\log n), 4]$ for every $\varepsilon > 0$. If the randomness is allowed to be slightly superlogarithmic, then the soundness can be reduced to some explicit $o(1)$ function. Technically, this result

is of interest in that it overcomes the qualitative limitation described above of passing through standard PCPs. Furthermore, the proof of this result is also of interest in that it shows how to apply the (by now) standard Fourier-analysis-based techniques to the studying of covering complexity as well. Thus it lays out the hope for applying such analysis to other cPCPs as well.

Unfortunately, the resulting cPCP fails to improve inapproximability of graph coloring or even hypergraph coloring. As noted earlier, covering PCPs are only necessary but not sufficient to get hardness results for hypergraph coloring. In order to get hardness results for hypergraph coloring from covering PCPs, one needs verifiers whose acceptance condition is an NAE SAT predicate (though, in this case, it is also reasonable to allow the responses of the queries to be elements of a nonbinary alphabet, and a result over a q-ary alphabet will give a result for q-colorable hypergraphs).

Keeping this objective in mind, we design a second verifier (whose query complexity is also 4 bits) whose acceptance predicate simply checks if the four queried bits are not all equal. The verifier has perfect completeness, and its covering soundness can be made an arbitrarily large constant (Theorem 4.2). This result immediately yields a superconstant lower bound on coloring 2-colorable 4-uniform hypergraphs: we prove that c-coloring such hypergraphs is NP-hard for any constant c (Theorem 4.4), and, moreover, there exists a constant $c_0 > 0$ such that, unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$), there is no polynomial time algorithm to color a 2-colorable 4-uniform hypergraph using $c_0 \frac{\log\log n}{\log\log\log n}$ colors (Theorem 4.6). A similar hardness result also holds for coloring 2-colorable k-uniform hypergraphs for any $k \geq 5$ by reduction from the case of 4-uniform hypergraphs (Theorem 4.7). Prior to our work, no nontrivial inapproximability results seem to be known for coloring 2-colorable hypergraphs, and, in fact, it was not known if 3-coloring a 2-colorable 4-uniform hypergraph is NP-hard.

We note that we do not have analogous results for the hardness of coloring 2-colorable 3-uniform hypergraphs. The difficulty in capturing the problem stems from the difficulty of analyzing the underlying maximization problem. The natural maximization version of hypergraph 2-coloring is the following: color the vertices with two colors so that a maximum number of hyperedges are nonmonochromatic. For l-uniform hypergraphs, this problem is known as Max l-Set Splitting. For $l = 4$ (the case we study here), a tight hardness result of $7/8 + \varepsilon$ is known [17], and this fact works its way into our analysis. For $k = 3$, a tight hardness result is not known for the maximization version (see [14]), and, in fact, nontrivial approximation algorithms exist [13], and our inability to show hardness results for 3-uniform hypergraphs seems to stem from this fact.

Organization. In section 2, we go over some of the definitions more formally and relate covering complexity to approximability of hypergraph coloring. In section 3, we analyze a simple cPCP verifier that makes four queries and has perfect completeness and $o(1)$ soundness. In section 4, we analyze a more complicated cPCP verifier with similar parameters whose acceptance condition is NAE SAT. This yields the hardness result for coloring 2-colorable, 4-uniform hypergraphs.

This is the complete version of the conference paper [15].

2. Preliminaries.

In this section, we introduce covering PCPs formally and establish a connection (in the wrong direction) between covering PCPs and inapproximability of hypergraph coloring.

2.1. PCPs.

We first give a formal definition of a PCP. Below verifiers are probabilistic oracle Turing machines whose output, on input x and random string r

with oracle O, is denoted $V^O(x,r)$. The output is a bit with 1 denoting acceptance and 0 denoting rejection.

DEFINITION 1. *Let c and s be real numbers such that $1 \geq c > s \geq 0$. A probabilistic polynomial time oracle Turing machine V is a PCP verifier with soundness s and completeness c for a language L iff*

- *for $x \in L$ there exists oracle Π such that $\mathrm{Prob}_r[V^\Pi(x,r)=1] \geq c$,*
- *for $x \notin L$, for all Π, $\mathrm{Prob}_r[V^\Pi(x,r)=1] \leq s$.*

Two parameters of interest in a PCP are the number of random bits used by the verifier and the number of queries it makes to the proof oracle. Most of the time, the symbols of Π are bits, and whenever this is not the case, this is stated explicitly.

DEFINITION 2. *For functions $r, q : \mathbb{Z}^+ \to \mathbb{Z}^+$, a verifier V is (r,q)-restricted if, on any input of length n, it uses at most $r(n)$ random bits and makes at most $q(n)$ queries to Π.*

We can now define classes of languages based on PCPs.

DEFINITION 3 (PCP). *A language L belongs to the class $\mathrm{PCP}_{c,s}[r,q]$ if there is an (r,q)-restricted verifier V for L with completeness c and soundness s.*

Next we have the definition of covering PCP.

DEFINITION 4 (covering PCP). *A language L belongs to the class $\mathrm{cPCP}_{c,s}[r,q]$ if there is an (r,q)-restricted verifier V such that, on input x,*

(i) *if $x \in L$, then there is a set of proofs $\{\Pi_1, \ldots, \Pi_k\}$ for $k \leq 1/c$ such that, for every random string r, there exists a proof Π_i such $V^{\Pi_i}(x,r)=1$; and*

(ii) *if $x \notin L$, then for every set of k proofs $\{\Pi_1, \Pi_2, \ldots, \Pi_k\}$ with $k < 1/s$, there is a random string r for which V rejects* every *Π_i, $1 \leq i \leq k$.*

One usually requires "perfect completeness" ($c = 1$) when seeking PCP characterizations. It is clear from the above definitions that $\mathrm{PCP}_{1,s}[r,q] \subseteq \mathrm{cPCP}_{1,s}[r,q]$, and thus obtaining a PCP characterization for a language class is at least as hard as obtaining a covering PCP characterization with similar parameters.

2.2. Covering PCPs and graph coloring. We now verify our intuition that "good" covering PCPs (i.e., those which have a large gap in covering complexity between the completeness and soundness cases) are necessary for strong lower bounds on approximating the chromatic number. As usual, for a graph G, we denote by $\chi(G)$ its chromatic number, i.e., the minimum number of colors required in a proper coloring of G.

Below, we use the phrase "it is NP-hard to distinguish $f(n)$-colorable graphs from $g(n)$-colorable graphs" to mean that "the $(2, f, g)$-approximation problem is NP-hard." As mentioned in section 1, note that we are using a conservative definition of NP-hardness, and hence this statement implies that there is a many-one reduction from SAT that maps satisfiable instances of SAT to $f(n)$ colorable graphs and maps unsatisfiable instances to graphs that are not $g(n)$-colorable. Under this assumption, we show how to get nice covering PCPs. Below and throughout this paper, the function log denotes logarithms to base two.

PROPOSITION 2.1. *Suppose, for functions $f, g : \mathbb{Z}^+ \to \mathbb{Z}^+$, given a graph G on n vertices, it is NP-hard to distinguish between the cases $\chi(G) \leq f(n)$ and $\chi(G) \geq g(n)$. Then*

$$\mathrm{NP} \subseteq \mathrm{cPCP}_{\lceil \log f(n) \rceil^{-1}, \lceil \log g(n) \rceil^{-1}} \left[O(\log n), 2 \right] .$$

Proof. Let the vertex set of G be $V = \{v_1, v_2, \ldots, v_n\}$. The covering PCP consists of proofs $\Pi_1, \Pi_2, \ldots, \Pi_k$ that correspond to "cuts" $\Gamma_1, \ldots, \Gamma_k$ of G; i.e., each Π_i is n-bits long, with the jth bit being 1 or 0 depending on which side of the cut Γ_i contains

v_j. The verifier simply picks two vertices v_{j_1} and v_{j_2} at random such that they are adjacent in G and then checks if the j_1th and j_2th bits differ in *any* of the k proofs. The minimum number k of proofs required to satisfy the verifier for all its random choices is clearly the *cut cover number* $\kappa(G)$ of G, i.e., the minimum number of cuts that cover all edges of G. It is easy to see that $\kappa(G) = \lceil \log \chi(G) \rceil$, and therefore the claimed result follows. □

One can get a similar result for any base q by letting the proofs be q-ary strings and letting the verifier read two q-ary symbols from the proof. In light of this, we get the following.

COROLLARY 2.2. *Suppose that there exists an $\varepsilon > 0$ such that it is NP-hard, given an input graph G, to distinguish between the cases when G is 3-colorable and when $\chi(G) \geq n^{\varepsilon}$. Then* $\mathrm{NP} \subseteq \mathrm{cPCP}_{1,(\varepsilon \log_3 n)^{-1}}\big[O(\log n), 2\big]$, *where the covering PCP is over a ternary alphabet, and the verifier's action is to simply read two ternary symbols from the proof and check that they are not equal.*

In light of the above corollary, very powerful covering PCP characterizations of NP are necessary in order to get strong hardness results for coloring graphs with small chromatic number. A result similar to Proposition 2.1, with an identical proof, also holds for hypergraph coloring and thus motivates us to look for good covering PCP characterizations of NP in order to prove hardness results for coloring 2-colorable hypergraphs.

PROPOSITION 2.3. *Suppose that there exists a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that, given an input r-uniform hypergraph on n vertices, it is NP-hard to distinguish between the cases when it is 2-colorable and when it is not $f(n)$-colorable. Then* $\mathrm{NP} \subseteq \mathrm{cPCP}_{1,\frac{1}{\log f(n)}}\big[O(\log n), r\big]$. *In particular, if c-coloring 2-colorable r-uniform hypergraphs is NP-hard for every constant c, then* $\mathrm{NP} \subseteq \mathrm{cPCP}_{1,\frac{1}{k}}\big[O(\log n), r\big]$ *for every constant $k \geq 1$.*

3. PCP construction I. We now move on to the constructions of our proof systems. For a reader familiar with PCPs, we first give a preview of our constructions. Both our PCPs (of this section and the next) go through the standard path. We start with strong 2-prover 1-round proof systems of Raz [28], apply the composition paradigm [5], and then use the long code of [8] at the bottom level. One warning: in the literature, it is common to use a variant of the long code—called the "folded long code"—we do not use the folded version. (Readers unfamiliar with the terms above will find elaborations in section 3.1.)

As usual, the interesting aspects of the constructions are the choice of the inner verifiers and the analyses of their soundness. The inner verifiers that we use are essentially from [17]: The inner verifier in section 4 is exactly the same as the one used by [17, section 7] to show hardness of Max 4-Set Splitting, while the one in this section is a small variant. The goals of the analyses are different since we are interested in the number of proofs required to cover all random strings. Despite the difference, we borrow large parts of our analysis from that of [17]. In the current section, our analysis essentially shows that if our verifier, on some fixed input, rejects every proof oracle with probability at least γ, then on any set of k proofs nearly γ^k fraction of random strings end up rejecting all of the proofs. Thus the standard soundness of the verifier we construct *is* of interest, and we analyze this using lemmas from [17]. The analysis of the verifier in section 4 does involve some new components, and we will comment upon these in the next section.

3.1. Preliminaries: Label Cover, long codes, proof composition. Our PCP constructions (also) follow the paradigm of *proof composition* by composing an "outer verifier" with an "inner verifier." In its most modern and easy to apply form, one starts with an *outer proof system* which is a 2-*Prover* 1-*Round proof system* (2P1R) construction for NP. We abstract the 2P1R by a graph-theoretic optimization problem called Label Cover. The specific version of Label Cover we refer to is the maximization version $\text{LabelCover}_{\max}$ discussed in [3] (see [3] for related versions and the history of this problem).

Label Cover. A $\text{LabelCover}_{\max}$ instance $\mathcal{LC}$ consists of a bipartite graph $H = (U, W, F)$ with vertex set $U \cup W$ and edge set F, "label sets" L_U, L_W which represent the possible *labels* that can be given to vertices in U, W, respectively, and *projection functions* $\pi_{u,w} : L_W \to L_U$ for each $u \in U$ and $w \in W$ such that $(u, w) \in F$. The optimization problem we consider is to assign a label $\ell(u) \in L_U$ (resp., $\ell(w) \in L_W$) to each $u \in U$ (resp., $w \in W$) such that the fraction of edges $e = (u', w')$ with $\ell(u') = \pi_{u',w'}(\ell(w'))$ (call such an edge "satisfied") is maximized. The optimum value of a $\text{LabelCover}_{\max}$ instance $\mathcal{LC}$, denoted $\mathsf{OPT}(\mathcal{LC})$, is the maximum fraction of "satisfied" edges in any label assignment. In the language of $\text{LabelCover}_{\max}$, the PCP theorem [5, 4], together with the parallel repetition theorem of Raz [28], yields parts (i)–(iii) of the theorem below. Here we need an additional property that is also used in [17, sections 6, 7]. First we need a definition: For $u \in U, w \in W$, a set $\beta \subseteq L_W$ and $0 < \varepsilon \leq 1$ define

$$\mu_\varepsilon(\beta, u, w) = \sum_{x \in L_U} \min\left\{1, \varepsilon \cdot \left|\pi_{u,w}^{-1}(x) \cap \beta\right|\right\}. \tag{1}$$

The definition above is quite technical (and borrowed directly from [17]), but the intuition is that β projects mostly onto different elements of L_U iff the "measure" μ is large.

THEOREM 3.1 (see [3, 17]). *There exist $d_0, e_0 < \infty$, and $c > 0$ and a transformation that, given a parameter $\delta > 0$, maps instances φ of* SAT *to instances $\mathcal{LC} = (U, W, F, L_U, L_W, \{\pi_{u,w} | (u, w) \in F\})$ of* $\text{LabelCover}_{\max}$*, in time $n^{O(\log \delta^{-1})}$, such that the following hold.*

(i) $|U|, |W| \leq n^{d_0 \log \delta^{-1}}$*, where n is the size of the* SAT *instance φ.*

(ii) $|L_U|, |L_W| \leq \delta^{-e_0}$.

(iii) *If φ is satisfiable, then $\mathsf{OPT}(\mathcal{LC}) = 1$, while, if φ is not satisfiable, then $\mathsf{OPT}(\mathcal{LC}) \leq \delta$.*

(iv) *For every $0 < \varepsilon \leq 1$, $w \in W$, and every $\beta \subseteq L_W$, $|\beta| \geq \varepsilon^{-1}$,*

$$\mathop{\mathbf{E}}_{u \in_R N(w)}\left[\frac{1}{\mu_\varepsilon(\beta, u, w)}\right] \leq (\varepsilon|\beta|)^{-c}, \tag{2}$$

where $N(w) = \{u \in U | (u, w) \in F\}$.

Remark. As mentioned earlier, conditions (i)–(iii) are standard for $\text{LabelCover}_{\max}$. The need for condition (iv) is inherited from some lemmas of [17] that we use (specifically, Lemmas 3.3 and 3.4). This condition is shown in Lemma 6.9 of [17].

To use the hardness of Label Cover, we use the standard paradigm of proof composition. The use of this paradigm requires an error-correcting code, which in our case is again the long code. We define this next.

The long code. We first remark on some conventions and notation we use through the rest of this paper: We represent Boolean values by the set $\{1, -1\}$ with

1 standing for FALSE and -1 for TRUE. This representation has the nice feature that XOR just becomes multiplication. For any domain D, denote by $\mathcal{F}_D$ the space of all Boolean functions $f : D \to \{1, -1\}$. For any set D, $|D|$ denotes its cardinality.

We now describe a very redundant error-correcting code called the *long code*. The long code was first used in [8] and has been very useful in most PCP constructions since.

The long code of an element x in a domain D, denoted $\mathsf{LONG}(x)$, is simply the evaluations of all the $2^{|D|}$ Boolean functions in $\mathcal{F}_D$ at x. If A is the long code of a, then we denote by $A(f)$ the coordinate of A corresponding to function f so that $A(f) = f(a)$.

We note that most of the proofs used in the literature use the "folded long code," which is a code of half the length of the long code, involving evaluations of the elements x at exactly one of the functions f or $-f$ (but not both). For reasons that will become clearer later, we *cannot* use the folded long code here and work with the actual long code.

Constructing a "composed" PCP. Note that Theorem 3.1 implies a PCP where the proof is simply the labels of all vertices in U, W of the $\text{LabelCover}_{\max}$ instance, and the verifier picks an edge $e = (u, w) \in F$ at random and checks if the labels of u and w are "consistent"; i.e., $\pi_{u,w}(\ell(w)) = \ell(u)$. An alternative is to choose a random neighbor w' of u and instead check that $\pi_{u,w}(\ell(w)) = \pi_{u,w'}(\ell(w'))$. By defining $\ell(u)$ to be the most common value of $\pi_{u,w'}(\ell(w'))$, it is easy to see that the probability of acceptance in the latter PCP (that uses w, w' for the check) is at most the probability of acceptance in the former PCP (that uses u, w for the check).

By the properties guaranteed in Theorem 3.1, either PCP uses $O(\log n \log \delta^{-1})$ randomness and has perfect completeness and soundness of at most δ. While the soundness is excellent, the number of bits it reads from the proof in total (from the two "locations" it queries) is large (namely, $O(\log \delta^{-1})$). In order to improve the query complexity, one "composes" this "outer" verification with an "inner" verification procedure. The inner verifier is given as input a projection function $\pi : L_W \to L_U$ and has oracle access to purported encodings, via the encoding function Enc of some error-correcting code, of two labels $a \in L_U$ and $b \in L_W$, and its aim is to check that $\pi(b) = a$ (with "good" accuracy) by making very few queries to $\mathsf{Enc}(a)$ and $\mathsf{Enc}(b)$. The inner verifiers we use have a slightly different character: they are given input two projections π_1 and π_2 (specifically $\pi_{u,w}$ and $\pi_{u,w'}$) and have oracle access to purported encodings $\mathsf{Enc}(b)$ and $\mathsf{Enc}(c)$ of two labels $b, c \in L_W$, and the aim is to test whether $\pi_1(b) = \pi_2(c)$. This interesting feature was part of and necessary for Håstad's construction for set splitting [17], and our PCPs also inherit this feature.

In our final PCP system, the proof is expected to be the encodings of the labels $\ell(w)$ of all vertices $w \in W$ using the encoding Enc. For efficient constructions, the code used is the *long code* of [8], i.e., $\mathsf{Enc} \stackrel{\text{def}}{=} \mathsf{LONG}$. We denote the portion of the (overall) proof that corresponds to w by $\mathsf{LP}(w)$, and, in a "correct" proof, $\mathsf{LP}(w)$ would just be $\mathsf{LONG}(\ell(w))$. (The notation LP stands for "long proof.")

The construction of a PCP now reduces to the construction of a good *inner verifier* that, given a pair of strings B, C which are purportedly long codes and projection functions π_1 and π_2, checks if these strings are the long codes of two "consistent" strings b and c whose respective projections agree (i.e., satisfy $\pi_1(b) = \pi_2(c)$). Given such an inner verifier IV, one can get a "composed verifier" V_{comp} using standard techniques as follows (given formula φ the verifier first computes the $\text{LabelCover}_{\max}$ instance $\mathcal{LC}$ in polynomial time and then proceeds with the verification):

1. Pick $u \in U$ at random and $w, w' \in N(u)$, at random.
2. Run the inner verifier with input $\pi_{u,w}$ and $\pi_{u,w'}$ and oracle access to $\mathsf{LP}(w)$ and $\mathsf{LP}(w')$.
3. Accept iff the inner verifier IV accepts.

We denote by $V_{\text{comp}}(\mathsf{IV})$ the composed verifier obtained using the inner verifier IV. The (usual) soundness analysis of the composed PCP proceeds by saying that, if there is a proof that causes the verifier V_{comp} to accept with large, say, $(s+\varepsilon)$, probability, where s is the soundness we are aiming for, then this proof can be "decoded" into labels for $U \cup W$ that "satisfy" more than a fraction δ of the edges in the $\text{LabelCover}_{\max}$ instance, and by Theorem 3.1, therefore, the original formula φ was satisfiable. In our case, we would like to make a similar argument and say that, if at most k proofs together satisfy all tests of V_{comp}, then these proofs can be "decoded" into labels for $U \cup W$ that satisfy more than δ fraction of edges of $\mathcal{LC}$.

3.2. The inner verifier. We now delve into the specification of our first "inner verifier," which we call Basic-IV4. This inner verifier is essentially the same as the one for 4-set splitting in [17] but has a different acceptance predicate. Recall that the inner verifier is given input two projection functions $\pi_1, \pi_2 : L_W \to L_U$, has oracle access to two tables $B, C : \mathcal{F}_{L_W} \to \{1, -1\}$, and aims to check that B (resp., C) is the long code of b (resp., c) which satisfies $\pi_1(b) = \pi_2(c)$.

`Inner Verifier` Basic-IV4$_p^{B,C}$ (π_1, π_2)
Choose uniformly at random $f \in \mathcal{F}_{L_U}$, $g_1, h_1 \in \mathcal{F}_{L_W}$
Choose at random $g', h' \in \mathcal{F}_{L_W}$ such that $\forall b \in L_W$,
$\mathbf{Pr}[g'(b) = 1] = p$ and $\mathbf{Pr}[h'(b) = 1] = p$
Set $g_2 = -g_1(f \circ \pi_1 \wedge g')$; $h_2 = -h_1(-f \circ \pi_2 \wedge h')$.
Accept iff $(B(g_1) \neq B(g_2)) \vee (C(h_1) \neq C(h_2))$

For a technical reason, as in [17], the final inner verifier needs to run the above inner verifier for the bias parameter p chosen at random from an appropriate set of values. The specific distribution we use is the one used by Håstad [17]. (The constant c used in its specification is the constant from (2) in the statement of Theorem 3.1.)

`Inner Verifier` IV4$_\gamma^{B,C}$ (π_1, π_2)
Set $t = \lceil 1/\gamma \rceil$, $\varepsilon_1 = \gamma$, and $\varepsilon_i = \gamma^{1+2/c}\varepsilon_{i-1}$ for $1 < i \leq t$.
Choose $p \in \{\varepsilon_1, \ldots, \varepsilon_t\}$ uniformly at random.
Run Basic-IV4$_p^{B,C}(\pi_1, \pi_2)$.

Note that the inner verifier above has perfect completeness. Indeed, when B, C are long codes of b, c, where $\pi_1(b) = \pi_2(c) = a$ (say), then, for each $f \in \mathcal{F}_{L_U}$, if $f(a) = 1$, then $B(g_1) = g_1(b)$, while $B(g_2) = B(-g_1(f \circ \pi_1 \wedge g')) = -g_1(b)$, and so these are not equal, and similarly for the case when $f(a) = -1$.

3.3. Covering soundness analysis. Let $X(\gamma)$ be the indicator random variable for the rejection of a particular proof $\Pi = \{\mathsf{LP}(w) : w \in W\}$ by the composed verifier $V_{\text{comp}}(\text{IV4}_\gamma)$ (henceforth $V_1(\gamma)$). The probability that $V_1(\gamma)$ rejects Π taken over its random choices is clearly the expectation

$$\mathop{\mathbf{E}}_{u,w,w',p,f,g_1,h_1,g_2,h_2}\big[X(\gamma)\big] = \mathbf{E}\left[\left(\frac{1 + B(g_1)B(g_2)}{2}\right)\left(\frac{1 + C(h_1)C(h_2)}{2}\right)\right]. \tag{3}$$

Here B, C are shorthand for $\mathsf{LP}(w)$ and $\mathsf{LP}(w')$, respectively, and equal $\mathsf{LONG}(\ell(w))$ and $\mathsf{LONG}(\ell(w'))$, respectively, in a "correct" proof. We wish to say that no k proofs can together satisfy all of the tests which $V_1(\gamma)$ performs. Now, if $X_k(\gamma)$ is the indicator random variable for the rejection of a set of k proofs $\{\mathsf{LP}_i(w) : w \in W\}$, $1 \le i \le k$, by the verifier $V_1(\gamma)$, then the overall probability that $V_1(\gamma)$ rejects all of these k proofs, taken over its random choices, is exactly

$$\mathop{\mathbf{E}}_{u,w,w',p,f,g_1,h_1,g_2}[X_k(\gamma)] = \frac{1}{4^k}\left(\mathbf{E}\left[\prod_{i=1}^{k}\big(1 + B_i(g_1)B_i(g_2)\big)\big(1 + C_i(h_1)C_i(h_2)\big)\right]\right), \tag{4}$$

where we use the shorthand B_i, C_i for $\mathsf{LP}_i(w), \mathsf{LP}_i(w')$, respectively.

We now argue (see Lemma 3.2 below) that, if this rejection probability is much smaller than 4^{-k}, then there is a way to obtain labels $\ell(u)$ for $u \in V \cup W$ by "decoding" Π_1 such that more than δ fraction of the edges (u, w) are satisfied by this labeling; i.e., $\ell(u) = \pi_{u,w}(\ell(w))$. Together with Theorem 3.1, this implies that the rejection probability (from (4)), for any set of k proofs for a false claim of satisfiability (of φ), can be made arbitrarily close to $\frac{1}{4^k}$ and, in particular, is nonzero, and thus the covering soundness of the composed verifier is at most $1/k$.

LEMMA 3.2. *There exists $a' < \infty$ such that, for every integer $k \ge 1$, every ε, $0 < \varepsilon < 4^{-k}$, and all $\gamma \le \varepsilon/8$, if $\mathbf{E}[X_k(\gamma)] < \frac{1}{4^k} - \varepsilon$, then $\mathsf{OPT}(\mathcal{LC}) > \gamma^{-a'\gamma^{-1}}$.*

Before presenting the formal proof of Lemma 3.2, we first highlight the basic approach. The power of arithmetizing the rejection probability for a set of k proofs as in (4) is that one can expand the product and analyze the expectation of

$$\frac{1}{4^k}\sum_{S,T\subseteq[k]} B_S(g_1)B_S(g_2)C_T(h_1)C_T(h_2), \tag{5}$$

where $B_S = \prod_{i\in S} B_i$ and $C_T = \prod_{i\in T} C_i$, where empty products are defined to be 1. A special term is $S = T = \emptyset$, which is the constant 1. We analyze the rest of the terms individually. We can now imagine two new proofs $\tilde{B} = B_S$ and $\tilde{C} = C_T$ which are exclusive-ors of subsets of the k given proofs. Now one can apply existing techniques from [17] to analyze terms involving the tables $\tilde{B}$ and $\tilde{C}$ and show that $\tilde{B}(g_1)\tilde{B}(g_2)$ and $\tilde{C}(h_1)\tilde{C}(h_2)$ cannot be too negative, and similarly if the expectation of $\tilde{B}(g_1)\tilde{B}(g_2)\tilde{C}(h_1)\tilde{C}(h_2)$ is too much below zero; then, in fact, $\mathsf{OPT}(\mathcal{LC})$ is quite large. In short, at a high level, we are saying that, if there exist k proofs such that the verifier accepts at least one of them with good probability, then some exclusive-or of these proofs is also accepted by the verifier with good probability, and we know that this cannot happen by the soundness analysis of [17] for the case of a single proof. This intuition is formalized via the next two lemmas from [17].

Before stating the lemmas, we make a slight digression to point out the relevance of not employing folding here. Folded long codes are typically used as follows: Given a table $A = \mathsf{LP}(w)$ supposedly giving the long code of the encoding of the label assigned to w, conceptually we assume that we have a long proof $A' = \mathsf{FoldedLP}(w)$ which respects the constraints $A'(f) = -A'(-f)$. We generate such an A' for ourselves from A' by setting $A'(f) = A(f)$ if $f(x_0) = 1$ and $A'(-f) = -A(f)$ if $f(x_0) = -1$, where x_0 is some fixed element of the concerned domain (i.e., L_U or L_W as the case might be). Such a table A', which satisfies $A'(-f) = -A'(f)$ for every function f, is said to be *folded*. We then pretend that the verifier works with the long code but carry out the soundness analysis only for folded tables. In our case, we could also do the same to analyze the acceptance of a single proof. However, when faced with

multiple proofs, the intermediate tables we consider, such as B_S above, need not be folded even if the original proofs we were given were folded; in particular, this will be the case when S has even cardinality. Thus our analysis needs to work with nonfolded tables as well. This is why we work with the long code directly.

Now we go back to the technical lemmas.

LEMMA 3.3 (see [17]). *For every $\gamma > 0$ and for all $B : \mathcal{F}_{L_W} \to \{1, -1\}$ and all $w \in W$,*

$$\mathop{\mathbf{E}}_{p,v \in N(w), f, g, g'} \big[B(g_1)B(g_2)\big] \geq -4\gamma,$$

where the distribution of p, f, g_1, g_2 is the same as the one in $\mathrm{IV4}_\gamma$.

This lemma is Lemma 7.9 in [17] combined with the calculation in the first half of Lemma 7.14 in the same paper. Similarly, the next lemma follows from Lemma 7.12 of the same paper and a similar calculation.

LEMMA 3.4 (see [17]). *There exists $a < \infty$ such that, for every $\gamma > 0$ and all proof tables $\{B_w\}$ and $\{C_w\}$, indexed by $w \in W$ with $B_w, C_w : \mathcal{F}_{L_W} \to \{1, -1\}$, we have that $\mathbf{E}\big[B_w(g_1)B_w(g_2)C_{w'}(h_1)C_{w'}(h_2)\big]$ is at least*

$$-7\gamma - \mathsf{OPT}(\mathcal{LC})\gamma^{-a\gamma^{-1}},$$

where the expectation is taken over $p, u, w, w', g_1, h_1, g_2, h_2$, and where the distribution of p, g_1, g_2, h_1, h_2 is the same as the one in $\mathrm{IV4}_\gamma$.

We are now ready to prove Lemma 3.2.

Proof of Lemma 3.2. The proof is actually simple given Lemmas 3.3 and 3.4. We pick a $\gamma > 0$ that satisfies $\gamma < \frac{\varepsilon}{8}$. By (4), if $\mathbf{E}[X_k(\gamma)] < 4^{-k} - \varepsilon$, then there exist subsets S_1, S_2 of $\{1, 2, \ldots, k\}$, $S_1 \cup S_2 \neq \emptyset$, such that

$$\mathbf{E}\big[B_{S_1}(g_1)B_{S_1}(g_2)C_{S_2}(h_1)C_{S_2}(h_2)\big] < -\varepsilon, \tag{6}$$

where B_{S_1} (resp., C_{S_2}) denotes $\Pi_{j \in S_1} B_j$ (resp., $\Pi_{j \in S_2} C_j$).

Suppose one of S_1, S_2 is empty (say, $S_2 = \emptyset$). Lemma 3.3 applied to B_{S_1} (which is a function mapping $\mathcal{F}_{L_W} \to \{1, -1\}$) gives $\mathbf{E}[B_{S_1}(g_1)B_{S_1}(g_2)] \geq -4\gamma$, which, together with (6) above, yields $\gamma > \frac{\varepsilon}{4}$, which is a contradiction since $\gamma \leq \varepsilon/8$.

Now suppose that both S_1 and S_2 are nonempty. Now we apply Lemma 3.4 to B_{S_1} and C_{S_2} to get that the expectation in (6) is at least $-7\gamma - \mathsf{OPT}(\mathcal{LC})\gamma^{-a\gamma^{-1}}$. Together with (6), this yields (using $\varepsilon \geq 8\gamma$)

$$\mathsf{OPT}(\mathcal{LC}) > \gamma\gamma^{a\gamma^{-1}} > \gamma^{a'\gamma^{-1}}$$

for some *absolute* constant a'. □

We are now ready to state and prove the main theorem of this section.

THEOREM 3.5. *For every constant k,* $\mathrm{NP} \subseteq \mathrm{cPCP}_{1, \frac{1}{k}}[\log, 4]$.

Proof. The theorem follows from Lemma 3.2 and Theorem 3.1. Let $\varepsilon = \frac{1}{2} \cdot 4^{-k}$ and $\gamma = \varepsilon/8$, and pick $\delta > 0$ small enough so that $\gamma^{-a'\gamma^{-1}} > \delta$. By Lemma 3.2, we have that $\mathbf{E}[X_k(\gamma)] < \frac{1}{4^k} - \varepsilon = \frac{1}{2 \cdot 4^k}$ implies $\mathsf{OPT}(\mathcal{LC}) > \delta$. Consider the PCP with verifier $V_{\mathrm{comp}}(\mathrm{IV4}_\gamma)$. Using Theorem 3.1, we get that, if the input formula φ is not satisfiable, then the verifier $V_{\mathrm{comp}}(\mathrm{IV4}_\gamma)$ rejects any k proofs with probability at least $\frac{1}{2 \cdot 4^k}$. Since it clearly has perfect completeness and makes only four queries, the claimed result follows. □

Remark on tightness of the analysis. In fact, Lemma 3.2 can be used to show that, for any $\varepsilon > 0$, there exists a (covering) PCP verifier that makes four queries, has

perfect completeness, and rejects any set of k proofs with probability at least $\frac{1}{4^k} - \varepsilon$. Note that this analysis is in fact *tight* for the verifier V_{comp}(IV4) since a random set of k proofs is accepted with probability $1 - 4^{-k}$. It would have been sufficient to prove that, for any k proofs, the set of verifier coins causing the verifier to reject all k proofs is nonempty. We do not know a simpler proof of this weaker statement.

4. PCP construction II and hardness of hypergraph coloring. In the previous section, we gave a PCP construction which made only four queries into the proof and had covering soundness smaller than any desired constant. This is already interesting in that it highlights the power of taking the covering soundness approach (since, as remarked in the introduction, one cannot achieve arbitrarily low soundness using classical PCPs with perfect completeness that make some fixed constant number of queries). We next turn to applying this to get a strong inapproximability result for hypergraph coloring.

The predicate tested by the inner verifier IV4$_\gamma$ is $F(x, y, z, w) = (x \neq y) \vee (z \neq w)$, and, to get a hardness result for hypergraph coloring, we require the predicate to be NAE(x, y, z, w), which is true unless all of x, y, z, w are equal. Note that NAE(x, y, z, w) is true whenever $F(x, y, z, w)$ is true, so one natural approach is to simply replace the predicate F tested by IV4$_\gamma$ by NAE without losing perfect completeness. The challenge, of course, is to prove that the covering soundness does not suffer in this process, and this is exactly what we accomplish. For completeness, we describe the inner verifier below.

`Inner Verifier` IV-NAE4$_\gamma^{B,C}$ (π_1, π_2)
  Pick p as in IV4$_\gamma$.
  Pick $f, g_1, h_1, g', h', g_2, h_2$ as in Basic-IV4$_p$.
  Accept iff not all of $B(g_1), B(g_2), C(h_1), C(h_2)$ are equal.

To analyze the soundness of the resulting composed verifier, we need to understand the not-all-equal predicate NAE. Note that NAE(x, y, z, w) rejects iff

$$\frac{1}{8}(1 + xy + xz + xw + yz + yw + zw + xyzw) = 1,$$

and this sum equals zero otherwise. With similar notation as in the previous section, this implies that, for a given choice of g_1, g_2, h_1, h_2, the verifier rejects all k proofs iff

$$8^{-k} \sum_{S_1,S_2,S_3,S_4,S_1\oplus S_2\oplus S_3\oplus S_4=\emptyset} B_{S_1}(g_1)B_{S_2}(g_2)C_{S_3}(h_1)C_{S_4}(h_2) = 1, \tag{7}$$

where $\oplus$ denotes the exclusive-or of characteristic vectors or, worded differently, the symmetric difference of sets. If the verifier accepts one of the proofs, then the right-hand side of (7) must equal zero. Hence we study the expected value of this quantity.

Before proceeding with the analysis, we shed some insight into the analysis and explain what is new this time. Let $T = S_1 \oplus S_2 = S_3 \oplus S_4$. The terms corresponding to T, being the empty set, are exactly the terms that appeared in the analysis of the verifier of section 3. Let us turn our attention to terms where $T \neq \emptyset$. Typically, when a sum such as the above appears, we would just analyze the individual terms. Unfortunately, it turns out that we are unable to do this in our case. To see why, consider a typical summand above, namely, $B_{S_1}(g_1)B_{S_1\oplus T}(g_2)C_{S_3}(h_1)C_{S_3\oplus T}(h_2)$. These are more general than the terms analyzed in section 3, which were of the form

$B_S(g_1)B_S(g_2)C_{S'}(h_1)C_{S'}(h_2)$. The first two elements of such a product come from an identical distribution, and similarly for the last two elements of the product. This in turn enabled a certain "pairing" up of terms from which a good solution to the Label Cover instance could be extracted (see the analysis in Lemma 7.12 of [17] for more details). But now, since $T \neq \emptyset$, the first two tables, B_{S_1} and $B_{S_1 \oplus T}$, are different, and so are the last two. Therefore, we now have to deal with individual terms which are the product of four elements, each of which comes from a different distribution. It does not seem possible to analyze such a term by itself and extract meaningful solutions to the Label Cover instance.

To cope with this problem, we now bunch together terms $B_{S_1}(g_1)B_{S_1 \oplus T}(g_2)C_{S_3}(h_1)$ $C_{S_3 \oplus T}(h_2)$ that involve the same T but different S_1 and S_3. (Alternatively, one could think of this as fixing T and then picking S_1 and S_3 as random subsets of $[k]$ and considering the expectation of the terms over S_1, S_3 as well.) This makes the distribution of the first pair as a whole identical to that of the second pair and allows us to analyze the terms above. More formally, for each nonempty $T \subseteq [k]$, we define

$$B^T(g_1, g_2) = \sum_S B_S(g_1)B_{S \oplus T}(g_2), \tag{8}$$

and similarly for C. Using this notation, the sum in (7) equals

$$\begin{aligned} 8^{-k} \Bigg(& 1 + \sum_S B_S(g_1)B_S(g_2) + \sum_S C_S(h_1)C_S(h_2) \\ & + \sum_S B_S(g_1)B_S(g_2)C_S(h_1)C_S(h_2) + \sum_{T \neq \emptyset} B^T(g_1, g_2)C^T(h_1, h_2) \Bigg), \end{aligned} \tag{9}$$

where the first four terms correspond to the case where $T = \emptyset$. Lemma 3.3 can be used to lower bound the expectation of the first two sums over g_1, g_2, h_1, h_2, and Lemma 3.4 can be used to lower bound the expectation of the third sum as a function of the optimum of the Label Cover instance. Thus we need only to study the last sum.

We show that, if the last term is too negative, then one can extract an assignment of labels to the provers. The intuition behind the proof is as follows. B^T and C^T are two functions chosen independently from the same distribution. Further, the queried pairs (g_1, g_2) and (h_1, h_2) are also chosen from the same distribution but are not independent of each other (and are related via f). If we ignore this dependence for a moment, then we get

$$\begin{aligned} \mathbf{E}[B^T(g_1, g_2)C^T(h_1, h_2)] &= \mathbf{E}[B^T(g_1, g_2)]\,\mathbf{E}[C^T(h_1, h_2)] \\ &= \mathbf{E}[B^T(g_1, g_2)]\,\mathbf{E}[B^T(g_1, g_2)] \geq 0, \end{aligned}$$

and this would be good enough for us. Unfortunately, (g_1, g_2) and (h_1, h_2) are not independent. The intuition behind the proof of the next inequality is that, if this correlation affects the expectation of $\mathbf{E}[B^T(g_1, g_2)C^T(h_1, h_2)]$, then there is some correlation between the tables for B^T and C^T, and so a reasonable strategy for assigning labels to w and w' can be extracted. Specifically, we get the following lemma.

LEMMA 4.1. *There exists $a' < \infty$ such that the following holds: Let $T \neq \emptyset$, $0 < \varepsilon < 2^{-k}$, and $\gamma \leq \varepsilon/8$ be such that*

$$\mathbf{E}\left[B^T(g_1, g_2)C^T(h_1, h_2)\right] \leq -\varepsilon,$$

where the expectation is taken over the distribution of $u, w, w', p, g_1, h_1, f, g', h', g_2, h_2$ *as in IV-NAE*4_γ. *Then* $\mathsf{OPT}(\mathcal{LC}) \geq \gamma^{a'\gamma^{-1}}$.

As usual, we postpone the proof of the lemma and instead prove the resulting theorem.

THEOREM 4.2. *For every constant* k, $\mathrm{NP} \subseteq \mathrm{cPCP}_{1,\frac{1}{k}}[\log, 4]$, *where, moreover, the predicate verified by the PCP upon reading bits* x, y, z, w *is* $\mathrm{NAE}(x, y, z, w)$.

Proof. We have only to analyze the soundness of the verifier. Let a be the constant from Lemma 3.4 and a' be the constant from Lemma 4.1. Let $b = \max\{a, a'\}$. Let $\gamma = 2^{-(k+5)}$, and let $\delta < \gamma^{b\gamma^{-1}}$. To create a verifier for an instance of SAT, reduce the instance of SAT to an instance of Label Cover using Theorem 3.1 with parameter δ, and then use the verifier based on using IV-NAE4_γ as the inner verifier. To show soundness, we need to show that, if this verifier is covered by k proofs, then the instance of Label Cover has an optimum greater than δ.

Suppose we have k proofs such that the verifier always accepts one of the proofs. This implies that the expectation, over $u, w, w', p, f, g_1, g_2, h_1, h_2$, of (9) is 0. This implies that at least one of summands in (9) is less than or equal to $-2^{-(k+2)}$ in expectation (since there are at most $4 \cdot 2^k$ summands in the expression). If it is a summand in one of the first two sums, then this contradicts Lemma 3.3. If it is a summand in the third sums, then, by Lemma 3.4, we get that $\mathsf{OPT}(\mathcal{LC}) \geq \gamma^{a\gamma^{-1}} > \delta$. If it is a summand in the last sum, then, by Lemma 4.1, we get that $\mathsf{OPT}(\mathcal{LC}) \geq \gamma^{a'\gamma^{-1}} > \delta$. Thus, in the last two cases, we get that the optimum is more than δ as desired. ∎

Before going on to the proof of Lemma 4.1, we discuss the consequences of Theorem 4.2 to hypergraph coloring. Before doing so, we just note that, in fact, one can prove a stronger claim in Theorem 4.2 that, given any k proofs, the probability that the verifier rejects all of them is at least $\frac{1}{8^k} - \varepsilon$ for $\varepsilon > 0$ as small as we seek. The proof is really the same as that of Theorem 4.2 since we have argued that all terms in the expansion (9) are arbitrarily small in the case when the optimum value of the Label Cover instance is very small. Once again, this soundness analysis is tight since a random set of k proofs will, in expectation, satisfy a fraction $1 - \frac{1}{8^k}$ of the verifier's checks.

4.1. Hardness results for hypergraph coloring. Since the predicate used by the PCP of Theorem 4.2 is that of 4-set splitting, we get the following corollary.

COROLLARY 4.3. *For every constant* $k \geq 2$, *given an instance of* 4*-set splitting, it is NP-hard to distinguish between the case when there is a partition of the universe that splits all of the* 4*-sets and when, for every set of* k *partitions, there is at least one* 4*-set which is not split by any of the* k *partitions.*

The above hardness can be naturally translated into a hardness result for coloring 4-uniform hypergraphs, and this gives us our main result.

THEOREM 4.4 (main theorem). *For any constant* $c \geq 2$, *it is NP-hard to color a* 2*-colorable* 4*-uniform hypergraph using* c *colors.*

Proof. The proof follows from the above corollary since a 4-set splitting instance can be naturally identified with a 4-uniform hypergraph whose hyperedges are the 4-sets, and it is easy to see that the minimum number of partitions k needed to split all 4-sets equals $\lceil \log c \rceil$, where c is the minimum number of colors to color the hypergraph such that no hyperedge is monochromatic. ∎

In light of the discussion after the proof of Theorem 4.2, we in fact have the following stronger result.

THEOREM 4.5. *For any constant $c \geq 2$ and every $\varepsilon > 0$, it is NP-hard to color a 2-colorable 4-uniform hypergraph using c colors such that at least a fraction $\left(1 - \frac{1}{8^{\lfloor \log c \rfloor}} + \varepsilon\right)$ of the hyperedges are properly colored (i.e., are not monochromatic).*

THEOREM 4.6. *Assume* NP $\not\subseteq$ DTIME$(n^{O(\log \log n)})$. *Then there exists an absolute constant $c_0 > 0$ such that there is no polynomial time algorithm that can color a 2-colorable 4-uniform hypergraph using $c_0 \frac{\log \log n}{\log \log \log n}$ colors, where n is the number of vertices in the hypergraph.*

Proof. This follows since the covering soundness of the PCP in Theorem 4.2 can be made an explicit $o(1)$ function. Indeed, nothing prevents having a k that is a function of n. We need to have $\gamma = 2^{-(k+5)}$ and to reach a contradiction $\delta < \gamma^{O(\gamma^{-1})}$. The proof size we need is $n^{O(\log \delta^{-1})} 2^{\delta^{-O(1)}}$. We can thus have $n^{O(\log \log n)}$ size proofs by letting $\delta^{-1} = (\log n)^b$ for some small enough constant b. The value of k can then satisfy $2^k = \Omega(\gamma^{-1}) = \Omega(\frac{\log \log n}{\log \log \log n})$. Similarly to Theorem 4.4, this implies that 2^k-coloring a 2-colorable 4-uniform hypergraph is hard unless NP $\subseteq$ DTIME$(n^{O(\log \log n)})$. ∎

We now show that a hardness result similar to Theorem 4.4 also holds for 2-colorable k-uniform hypergraphs for any $k \geq 5$.

THEOREM 4.7. *Let $k \geq 5$ be an integer. For any constant $\ell \geq 2$, it is NP-hard to color a 2-colorable k-uniform hypergraph using ℓ colors.*

Proof. The proof works by reducing from the case of 4-uniform hypergraphs, and the claimed hardness then follows using Theorem 4.4.

Let $\mathcal{H}$ be a 4-uniform hypergraph with vertex set V. Suppose that $k = 4s + t$, where $1 \leq t \leq 4$. Construct a k-uniform hypergraph $\mathcal{H}'$ as follows. The vertex set of $\mathcal{H}'$ is $V^{(1)} \cup V^{(2)} \cup \cdots \cup V^{(s\ell+1)}$, where the sets $V^{(j)}$ are independent copies of V. On each $V^{(j)}$, take a collection $\mathcal{F}^{(j)}$ of 4-element subsets of $V^{(j)}$ that correspond to the hyperedges in $\mathcal{H}$. A hyperedge of $\mathcal{H}'$ (which is a $(4s + t)$-element subset of $\bigcup_j V^{(j)}$) is now given by the union of s 4-sets belonging to s different $\mathcal{F}^{(j)}$'s, together with t vertices picked from a 4-set belonging to yet another $\mathcal{F}^{(j)}$. More formally, for every set of $(s+1)$ distinct indices $j_1, j_2, \ldots, j_{s+1}$, every choice of elements $e_{j_i} \in \mathcal{F}^{(j_i)}$ for $i = 1, \ldots, s + 1$, and every t-element subset $f_{j_{s+1}}$ of $e_{j_{s+1}}$, there is a hyperedge $(e_{j_1} \cup \cdots \cup e_{j_s} \cup f_{j_{s+1}})$ in $\mathcal{H}'$.

If $\mathcal{H}$ is 2-colorable, then clearly any 2-coloring of it induces a 2-coloring of $\mathcal{H}'$, and hence $\mathcal{H}'$ is 2-colorable as well.

Suppose $\mathcal{H}$ is not ℓ-colorable and that we are given an ℓ-coloring of $\mathcal{H}'$. Since $\mathcal{H}$ is not ℓ-colorable, each $\mathcal{F}^{(j)}$, for $1 \leq j \leq s\ell + 1$, must contain a monochromatic set g_j. By the pigeonhole principle, there must be a color c such that $(s + 1)$ different g_j's have color c. The hyperedge of $\mathcal{H}'$ constructed from those $(s + 1)$ sets is then clearly monochromatic (all its vertices have color c), and we conclude that $\mathcal{H}'$ is not ℓ-colorable.

Since the reduction runs in polynomial time when k and ℓ are constants, the proof is complete. ∎

4.2. Discrete Fourier transforms. Before going on to the proof of Lemma 4.1, we now introduce a tool that has been crucial in the analysis on inner verifiers. This was hidden so far from the reader but already used in the proofs of Lemmas 3.3 and 3.4 in [17]. Now we need to introduce them explicitly.

In general, we consider functions mapping D to $\{1, -1\}$. For $\alpha \subseteq D$ and $f \in \mathcal{F}_D$, let $\chi_\alpha(f) = \prod_{x \in \alpha} f(x)$. Notice that $\chi_{\{x\}}$ is the long code of x. For any function A

mapping $\mathcal{F}_D$ to the reals, we have the corresponding Fourier coefficients

$$\hat{A}_\alpha = 2^{-2^{|D|}} \sum_f A(f)\chi_\alpha(f),$$

where $\alpha \subseteq D$. We have the Fourier inversion formula given by

$$A(f) = \sum_\alpha \hat{A}_\alpha \chi_\alpha(f)$$

and Plancherel's equality that states that

$$\sum_\alpha \hat{A}_\alpha^2 = 2^{-2^{|D|}} \sum_f A(f)^2.$$

In the case when A is a Boolean function, the latter sum is clearly 1.

We think of an arbitrary table A as being somewhat close to (or coherent with) the long code of x if there exists a small set α containing x such that $\hat{A}_\alpha$ is nonnegligibly large. Thus, when viewing the long proofs of w and w', our goal is to show that the $\mathsf{LP}(w)$ and $\mathsf{LP}(w')$ have coherence with the long codes of strings x and y such that $\pi_{u,w}(x)$ and $\pi_{u,w'}(y)$ are equal.

4.3. Proof of Lemma 4.1. Fix $T \subseteq [k]$. Throughout this section, the quantities that we define depend on T, but we do not include it as a parameter explicitly.

Recall that we need to show that, if the expectation, over $u, w, w', p, g_1, g_2, h_1, h_2$ of $B^T(g_1, g_2)C^T(h_1, h_2)$, is too negative (less than $-\varepsilon$), then we can assign labels to the Label Cover problem with acceptance probability more than δ. Recall that $g_2 = g_1(f \circ \pi_1 \wedge g')$ is defined in terms of other random variables f and g' and similarly h_2 in terms of f and h'. For brevity, we let X denote the quantity $B^T(g_1, g_2)C^T(h_1, h_2)$. Notice that X is a random variable depending on all of the variables above. We first analyze the expectation of X over g_1 and h_1 (for fixed choice of u, w, w', p, f, g', and h'). Next we calculate the expectation over f, g', and h'. In both stages, we get exact expressions. Finally, we make some approximations for the expectation over u, w, w'. (The careful reader may observe that we do not take expectations over p—in fact, the lemma holds for every choice of p of the inner verifier IV-NAE4$_\gamma$.)

The crux of this proof is the functions $B^*, C^* : \mathcal{F}_{L_W} \to \{1, -1\}$, defined as follows: We let

$$B^*(g) = \mathop{\mathbf{E}}_h \left[\sum_S B_S(h) B_{S \oplus T}(gh) \right]$$

and

$$C^*(g) = \mathop{\mathbf{E}}_h \left[\sum_S C_S(h) C_{S \oplus T}(gh) \right].$$

Note that, for a fixed choice of f and g', we have $\mathbf{E}_{g_1}[B^T(g_1, g_2)] = B^*(-((f \circ \pi_1) \wedge g'))$. We get a similar expression for C^T, and thus we get

$$\mathop{\mathbf{E}}_{g_1, h_1}[X] = B^*(-((f \circ \pi_1) \wedge g'))C^*(-((-f \circ \pi_2) \wedge h')).$$

Let us call the above quantity Y.

In what follows, we rely crucially on the properties of the Fourier coefficients of B^* and C^*. Let $\hat{F}_\beta$ and $\hat{G}_\beta$ denote the Fourier coefficients of B^* and C^*, respectively. From the definitions and some standard manipulation, we get

$$\hat{F}_\beta = \sum_S \hat{B}_{\beta,S}\hat{B}_{\beta,S\oplus T} \quad \text{and} \quad \hat{G}_\beta = \sum_S \hat{C}_{\beta,S}\hat{C}_{\beta,S\oplus T}.$$

Using simple Fourier expansion, we can rewrite the quantity we are analyzing as

$$\begin{aligned} Y &= B^*(-((f\circ\pi_1)\wedge g'))C^*(-((-f\circ\pi_2)\wedge h')) \\ &= \sum_{\beta,\beta'} \hat{F}_\beta\hat{G}_{\beta'}\chi_\beta(-((f\circ\pi_1)\wedge g'))\chi_{\beta'}(-((-f\circ\pi_2)\wedge h')) \\ &= \sum_{\beta,\beta'} \hat{F}_\beta\hat{G}_{\beta'}\prod_{y\in\beta}(-(f(\pi_1(y))\wedge g'(y)))\prod_{z\in\beta'}(-(-f(\pi_2(z))\wedge h'(z))). \end{aligned}$$

The main property about the Fourier coefficients of B^* and C^* is that their L_1 norm is bounded. Specifically, we have

$$\sum_\beta |\hat{F}_\beta| \le \sum_{\beta,S} |\hat{B}_{\beta,S}\hat{B}_{\beta,S\oplus T}| \le \sum_S \left(\sum_\beta \hat{B}^2_{\beta,S}\right)^{1/2}\left(\sum_\beta \hat{B}^2_{\beta,S\oplus T}\right)^{1/2} \le 2^k. \tag{10}$$

We start by defining the strategy we use to assign labels and prove that, if the expectation (of Y) is large, then the labels give an assignment to the Label Cover instance with objective of at least $\delta = \gamma^{a'\gamma^{-1}}$.

Strategy. Given $w \in W$ and tables $B_1,\dots,B_k$ corresponding to $\mathsf{LP}(w)$ in k different proofs, compute B_S for every $S \subseteq [k]$, B^* and its Fourier coefficients. Pick a nonempty set $\beta \subseteq L_W$ with probability $2^{-k}|\hat{F}_\beta|$, and assign as label to w, an element $x \in \beta$ chosen uniformly at random. With remaining probability, since $\sum_{\beta\neq\emptyset}|\hat{F}_\beta|$ may be less than 2^k, assign no label to w.

Preliminary analysis. We now give a preliminary expression for the success probability of the strategy. Consider picking u, w, and w' (and the associated π_1 and π_2) at random and checking for the event $\pi_1(\ell(w)) = \pi_2(\ell(w'))$. The probability of this event is lower bounded by the probability that $\pi_1(\beta)$ and $\pi_2(\beta')$ intersect, and we assign the elements corresponding to this intersection to w and w'. The probability of these events is at least

$$\mathop{\mathbf{E}}_{u,w,w'}\left[\sum_{\beta,\beta':\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset}\frac{2^{-2k}}{|\beta|\cdot|\beta'|}|\hat{F}_\beta||\hat{G}_{\beta'}|\right]. \tag{11}$$

Below we show that this quantity is large if the expectation of Y is too small. We now return to the expectation of Y.

An exact expression for the expectation of Y. We start with some notation. Fix u, w, w', p, and π_1 and π_2. For $x \in L_U$ and $\beta, \beta' \subseteq L_W$, let $s_x(\beta) = |\beta \cap \pi_1^{-1}(x)|$, and let $t_x(\beta') = |\beta' \cap \pi_2^{-1}(x)|$. Since the argument of s_x is always β and the argument of t_x is always β', we use the shorthand s_x for $s_x(\beta)$ and t_x for $t_x(\beta')$. Further, for real p and nonnegative integers s, t, let $\alpha(p,s,t) = \frac{1}{2}\left((-1)^s(1-2p)^t + (1-2p)^s(-1)^t\right)$, and let $\eta(p,s) = \frac{1}{2}\left((-1)^s + (1-2p)^s\right)$. Next we show that

$$\mathop{\mathbf{E}}_{f,g',h'}[Y] = \sum_{\beta,\beta'}\hat{F}_\beta\hat{G}_{\beta'}\prod_{x\in L_U}\alpha(p,s_x,t_x). \tag{12}$$

To prove the above, it suffices to show that

$$\mathop{\mathbf{E}}_{f,g',h'}\left[\prod_{y\in\beta}\left[-(f(\pi_1(y))\wedge g'(y))\right]\prod_{z\in\beta'}\left[-(-f(\pi_2(z))\wedge h'(z))\right]\right]=\prod_{x\in L_U}\alpha(p,s_x,t_x).$$

Factors corresponding to y and z with different projections on L_U are independent, and thus the expectation can be broken down into a product of expectations, one for each $x\in L_U$. Fix $x\in L_U$, and consider the term

$$\mathop{\mathbf{E}}_{f,g',h'}\left[\prod_{y\in\beta\cap\pi_1^{-1}(x)}(-(f(\pi_1(y))\wedge g'(y)))\prod_{z\in\beta'\cap\pi_2^{-1}(x)}(-(-f(\pi_2(z))\wedge h'(z)))\right]$$
$$=\mathop{\mathbf{E}}_{f,g',h'}\left[\prod_{y\in\beta\cap\pi_1^{-1}(x)}(-(f(x)\wedge g'(y)))\prod_{z\in\beta'\cap\pi_2^{-1}(x)}(-(-f(x)\wedge h'(z)))\right].$$

If $f(x)=1$ (or "false"), the first product equals $(-1)^{s_x}$, and the second equals $(1-2p)^{t_x}$. Similarly, if $f(x)=-1$, the first product equals $(1-2p)^{s_x}$, and the second equals $(-1)^{t_x}$. The events happen with probability $1/2$ each and thus give that the expectation above (for fixed x) equals

$$\frac{1}{2}\left((-1)^{s_x}(1-2p)^{t_x}+(1-2p)^{s_x}(-1)^{t_x}\right)=\alpha(p,s_x,t_x).$$

Taking the product over all x's gives (12).

Inequalities on E[*Y*]. For every u, we now show how to lower bound the expectation of Y over w,w',f,g',h' in terms of a sum involving only β's and β' that *intersect* in their projections. This brings us much closer to the expression derived in our preliminary analysis of the success probability of our strategy for assigning labels, and we lower bound a closely related quantity. Specifically, we now use the inequality $\mathbf{E}_{w,w',f,g',h'}[Y]\le-\varepsilon$ (as guaranteed by the lemma statement) to show

$$\mathop{\mathbf{E}}_{u,w,w'}\left[\sum_{\beta,\beta'|\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset}2|\hat{F}_\beta\hat{G}_{\beta'}|e^{-\mu_p(\beta)/2}\right]\ge\varepsilon, \tag{13}$$

where $\mu_p(\beta)=\mu_p(\beta,u,w)=\sum_x\min\{1,ps_x\}$ is the quantity defined in (1). (A similar inequality with $\mu_p(\beta')$ in the exponent follows by symmetry.)

To prove the above, consider the following expression, which is closely related to the expectation of Y as given by (12):

$$Y_1=\left(\sum_\beta\hat{F}_\beta\prod_x\eta(p,s_x)\right)\left(\sum_{\beta'}\hat{G}_{\beta'}\prod_x\eta(p,t_x)\right).$$

First we note that $\mathbf{E}_{w,w'}[Y_1]=(\mathbf{E}_w[(\sum_\beta\hat{F}_\beta\prod_x\eta(p,s_x))])^2\ge0$. (Here we are using the fact that the tables B^T and C^T are chosen from the same distribution.) Next we note that the difference between Y and Y_1 arises only from terms involving β,β' such that $\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset$. To verify this, note that, if $s=0$ or $t=0$, then $\alpha(p,s,t)=$

$\eta(p,s)\cdot\eta(p,t)$. Since either s_x or t_x equals 0 for every x, if $\pi_1(\beta)\cap\pi_2(\beta')=\emptyset$, we get that terms corresponding to such pairs of β,β' vanish in $Y-Y_1$. We conclude that

$$\mathop{\mathbf{E}}_{w,w',f,g',h'}[Y]=\mathop{\mathbf{E}}_{w,w'}[Y_1]$$

$$+\sum_{\beta,\beta':\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset}\hat{F}_\beta\hat{G}_{\beta'}\left(\prod_x\alpha(p,s_x,t_x)-\prod_x(\eta(p,s_x)\cdot\eta(p,t_x))\right).$$

Using $\mathbf{E}_{w,w',f,g',h'}[Y]\le-\varepsilon$, $\mathbf{E}[Y_1]\ge0$ and taking absolute values, we get

$$\begin{aligned}\varepsilon&\le\left|\sum_{\beta,\beta':\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset}\hat{F}_\beta\hat{G}_{\beta'}\left(\prod_x\alpha(p,s_x,t_x)-\prod_x(\eta(p,s_x)\cdot\eta(p,t_x))\right)\right|\\&\le\sum_{\beta,\beta':\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset}|\hat{F}_\beta\hat{G}_{\beta'}|\left(\prod_x|\alpha(p,s_x,t_x)|+\prod_x|\eta(p,s_x)|\right),\end{aligned}\tag{14}$$

where the last inequality uses $|\eta(p,t)|\le1$ for every $t\ge0$.

Next we simplify the terms of the left-hand side above. First we show that, for every $t\ge0$,

$$|\alpha(p,s,t)|,|\eta(p,s)|\le e^{-\min\{1,ps\}/2}.\tag{15}$$

First we note that both $|\alpha(p,s,t)|$ and $|\eta(p,s)|$ are upper bounded by $\frac12(1+(1-2p)^s)\le\frac12(1+e^{-2ps})$. If $p\ge s^{-1}$, then we have $\frac12(1+e^{-2ps})\le\frac12(1+e^{-2})\le e^{-1/2}$ as required. If $p\le\frac1s$, let us set $\lambda(z)=2e^{-z/2}-(1+e^{-2z})$. To show (15), we need to prove that $\lambda(z)\ge0$ for $z\in[0,1]$. We have $\lambda''(z)=\frac12e^{-z/2}-4e^{-2z}$, and hence $\lambda''(z)\le0$ in the interval in question, and we have only to check the inequality at the end points. We have $\lambda(0)=0$ and $\lambda(1)=2e^{-1/2}-(1+e^{-2})>0$.

Using (15), we conclude that

$$\prod_x|\alpha(p,s_x,t_x)|+\prod_x|\eta(p,s_x)|\le2\prod_xe^{-\min\{1,ps_x\}}=2e^{-\mu_p(s_x)}.$$

Substituting the above into (14) gives (13).

Based on (13), we want to prove that the strategy for assigning labels is a good one. First we prove that large sets β do not contribute much to the left-hand side of the sum in (13). Define

$$K=p^{-1}(\varepsilon^{-1}2^{2k+4}(4k+8+2\log\varepsilon^{-1}))^{1/c}.$$

We have the following lemma.

LEMMA 4.8. *We have*

$$\mathbf{E}\left[\sum_{\pi(\beta)\cap\pi(\beta')\neq\emptyset,|\beta|\ge K}2|\hat{F}_\beta\hat{G}_{\beta'}|e^{-\mu_p(\beta)/2}\right]\le\varepsilon/4.$$

Proof. By property (iv) of Theorem 3.1, we have that the probability that $\mu_p(\beta)\le(4k+8+2\log\varepsilon^{-1})$ is at most $\varepsilon2^{-(2k+4)}$. A similar chain of inequalities to (10) shows

that $\sum |\hat{G}_{\beta'}| \le 2^k$. The sum in the lemma can hence be estimated as

$$\begin{aligned} 2^{k+1} \sum_{|\beta|\ge K} |\hat{F}_\beta| \mathop{\mathbf{E}}_u[e^{-\mu_p(\beta)/2}] &\le 2^{k+1} \sum_{|\beta|\ge K} |\hat{F}_\beta| \big(2^{-(2k+4)}\varepsilon + e^{-(2k+4+\log \varepsilon^{-1})}\big) \\ &\le 2^{2k+1}(2^{-(2k+4)}\varepsilon + 2^{-(2k+4)}\varepsilon) \\ &\le \varepsilon/4, \end{aligned}$$

and the lemma follows. □

By the same argument applied to β' of size at least K, together with (13), we get

$$\mathbf{E}\left[\sum_{\pi(\beta)\cap\pi(\beta')\neq\emptyset, |\beta|,|\beta'|\le K} 2|\hat{F}_\beta \hat{G}_{\beta'}|\right] \ge \varepsilon/2. \tag{16}$$

We now relate to the probability of success of our strategy for assigning labels. From (11) we know this quantity is at least

$$\begin{aligned} &\mathop{\mathbf{E}}_{u,w,w'}\left[\sum_{\beta,\beta':\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset} \frac{2^{-2k}}{|\beta|\cdot|\beta'|}|\hat{F}_\beta||\hat{G}_{\beta'}|\right] \\ &\quad\ge \mathop{\mathbf{E}}_{u,w,w'}\left[\sum_{\pi_1(\beta)\cap\pi_2(\beta')\neq\emptyset, |\beta|,|\beta'|\le K} \frac{2^{-2k}}{K^2}|\hat{F}_\beta||\hat{G}_{\beta'}|\right] \\ &\quad\ge \frac{\varepsilon 2^{-(2k+2)}}{K^2}, \end{aligned}$$

where the last inequality uses (16). (It is easy to convert this randomized strategy for assigning labels to a deterministic one that does equally well.) The dominating factor in the expression is the term $p^{-O(1)}$ (from the definition of K), which can be calculated to be $\gamma^{O(\gamma^{-1})}$, and the proof of Lemma 4.1 is complete. □

4.3.1. Comparison to previous proof of Theorem 4.4. We point out that the conference version of this paper [15] contained a different proof of Theorem 4.4. The current proof is significantly simpler, and, furthermore, it is only a minor adjustment of similar proofs in [17]. The key observation to make the current proof possible is the insight that we should treat the terms of (7) in the collections given by $B^T(g_1, g_2)C^T(h_1, h_2)$, as it does not seem possible to handle them one by one in an efficient manner. The previous proof did not make this observation explicitly and ended up being significantly more complicated. This "simplicity" in turn has already enabled some further progress on the hypergraph coloring problem; in particular, using this style of analysis, Khot [21] shows a better superconstant hardness for a-colorable 4-uniform hypergraphs for $a \ge 7$.

4.3.2. Subsequent related work. In a very recent work, Holmerin [18] showed that the vertex cover problem considered on 4-uniform hypergraphs is NP-hard to approximate within a factor of $(2-\varepsilon)$ for arbitrary $\varepsilon > 0$. (A subset S of vertices of a hypergraph H is said to be a vertex cover if every hyperedge of H intersects S.) He proves this by modifying the soundness analysis of Håstad's 4-set splitting verifier (which is also the verifier we use in section 4) to show that any proof which sets only a fraction ε of bits to -1 will cause some 4-tuple tested by the verifier to consist of only 1's. This in turn shows that, for every constant $\varepsilon > 0$, given a 2-colorable

4-uniform hypergraph, it is NP-hard to find an independent set that consists of a fraction ε of vertices. Note that this result is stronger, as a small independent set implies a large chromatic number, and it thus immediately implies the hardness of coloring such a 2-colorable 4-uniform hypergraph with $1/\varepsilon$ colors, and hence our main result (Theorem 4.4). We stress that the verifier in Holmerin's paper is the same as the one in this paper; however, the analysis in [18] obtains our result without directly referring to covering complexity.

Acknowledgments. We would like to thank the anonymous referees and Oded Goldreich for useful comments on the presentation of the paper.

REFERENCES

[1] N. Alon, P. Kelsen, S. Mahajan, and H. Ramesh, *Coloring 2-colorable hypergraphs with a sublinear number of colors*, Nordic J. Comput., 3 (1996), pp. 425–439.
[2] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331.
[3] S. Arora and C. Lund, *Hardness of approximations*, in Approximation Algorithms for NP-Hard Problems, Dorit Hochbaum, ed., PWS Publishing, Boston, 1996.
[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
[5] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.
[6] J. Beck, *On 3-chromatic hypergraphs*, Discrete Math., 24 (1978), pp. 127–137.
[7] J. Beck, *An algorithmic approach to the Lovász local lemma*, Random Structures Algorithms, 2 (1991), pp. 343–365.
[8] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.
[9] A. Blum and D. R. Karger, *Improved approximation for graph coloring*, Inform. Process. Lett., 61 (1997), pp. 49–53.
[10] H. Chen and A. Frieze, *Coloring bipartite hypergraphs*, in Proceedings of the 5th IPCO, Lecture Notes in Comput. Sci. 1084, Springer-Verlag, New York, 1996, pp. 345–358.
[11] P. Erdös, *On a combinatorial problem*, Nordisk Mat. Tidskr., 11 (1963), pp. 5–10.
[12] U. Feige and J. Kilian, *Zero-knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199.
[13] M. Goemans and D. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.
[14] V. Guruswami, *Inapproximability results for set splitting and satisfiability problems with no mixed clauses*, in Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 1913, Springer-Verlag, Berlin, 2000, pp. 155–166; Algorithmica, to appear.
[15] V. Guruswami, J. Håstad, and M. Sudan, *Hardness of approximate hypergraph coloring*, in Proceedings of 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, pp. 149–158.
[16] V. Guruswami and S. Khanna, *On the hardness of 4-coloring a 3-colorable graph*, in Proceedings of the 15th Annual IEEE Conference on Computational Complexity, Florence, 2000, pp. 188–197.
[17] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.
[18] J. Holmerin, *Vertex cover on 4-regular hyper-graphs is hard to approximate within* $(2-\varepsilon)$, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 544–552.
[19] D. R. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring using semidefinite programming*, J. ACM, 45 (1998), pp. 246–265.
[20] S. Khanna, N. Linial, and S. Safra, *On the hardness of approximating the chromatic number*, Combinatorica, 20 (2000), pp. 393–415.
[21] S. Khot, *Hardness results for approximate hypergraph coloring*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 351–359.

[22] M. Krivelevich and B. Sudakov, *Approximate coloring of uniform hypergraphs*, in Proceedings of the 6th European Symposium on Algorithms, Lecture Notes in Comput. Sci. 1461, Springer-Verlag, New York, 1998, pp. 477–489.
[23] L. Lovász, *Coverings and colorings of hypergraphs*, in Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Utilitas Mathematica Publishing, Winnipeg, 1973, pp. 3–12.
[24] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.
[25] C. McDiarmid, *A random recoloring method for graphs and hypergraphs*, Combin. Probab. Comput., 2 (1993), pp. 363–365.
[26] C. McDiarmid, *Hypergraph coloring and the Lovász local lemma*, Discrete Math., 167/168 (1997), pp. 481–486.
[27] J. Radhakrishnan and A. Srinivasan, *Improved bounds and algorithms for hypergraph* 2*-coloring*, Random Structures Algorithms, 16 (2000), pp. 4–32.
[28] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[29] J. H. Spencer, *Coloring* n*-sets red and blue*, J. Combin. Theory Ser. A, 30 (1981), pp. 112–113.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1687–1722

PHASE CHANGE OF LIMIT LAWS IN THE QUICKSORT RECURRENCE UNDER VARYING TOLL FUNCTIONS*

HSIEN-KUEI HWANG† AND RALPH NEININGER‡

Abstract. We characterize all limit laws of the quicksort-type random variables defined recursively by $\mathcal{L}(X_n) = \mathcal{L}(X_{I_n} + X^*_{n-1-I_n} + T_n)$ when the "toll function" T_n varies and satisfies general conditions, where $(X_n), (X^*_n), (I_n, T_n)$ are independent, I_n is uniformly distributed over $\{0, \dots, n-1\}$, and $\mathcal{L}(X_n) = \mathcal{L}(X^*_n)$. When the "toll function" T_n (cost needed to partition the original problem into smaller subproblems) is small (roughly $\limsup_{n\to\infty} \log E(T_n)/\log n \le 1/2$), X_n is asymptotically normally distributed; nonnormal limit laws emerge when T_n becomes larger. We give many new examples ranging from the number of exchanges in quicksort to sorting on a broadcast communication model, from an in-situ permutation algorithm to tree traversal algorithms, etc.

Key words. quicksort, binary search trees, analysis of algorithms, limit distribution, method of moments, contraction method

AMS subject classifications. Primary, 68W40, 68Q25; Secondary, 60F05, 11B37

PII. S009753970138390X

1. Quicksort recurrence.

Quicksort, invented by Hoare [37], is one of the most widely used general-purpose sorting algorithms and was selected to be among the top ten most influential algorithms in science and engineering in the 20th century; see JaJa [41]. For more information on the practical implementation and recent development of quicksort, see, for example, [5, 49, 75]. Assume that the input comes from a sequence of independent and identically distributed random variables with a common continuous distribution, the cost measures, say, X_n, on quicksort can generally be described by $X_0 = 0$, and, for $n \ge 1$,

$$X_n \stackrel{d}{=} X_{I_n} + X^*_{n-1-I_n} + T_n, \tag{1}$$

where $(X_n), (X^*_n), (T_n, I_n)$ are independent, $X_n \stackrel{d}{=} X^*_n$, and I_n is uniformly distributed over $\{0, \dots, n-1\}$. Here the symbol $\stackrel{d}{=}$ denotes equivalence in distribution, and T_n is either a deterministic function of n or a random variable depending on I_n or not. Throughout this paper, we call T_n the *toll function*. Note that this description implicitly assumes that the randomness is preserved for each subfile after partitioning, a property enjoyed by many partitioning schemes but easily violated if carelessly implemented; see Sedgewick [71] for a detailed discussion. Our aim in this paper is to develop a distribution theory for X_n based on the stochastic behavior of T_n.

The motivation of such a study is multifold. First, the model is simple yet prototypical of many sophisticated divide-and-conquer schemes. Viewing this recurrence

*Received by the editors January 22, 2001; accepted for publication (in revised form) April 3, 2002; published electronically September 5, 2002.

http://www.siam.org/journals/sicomp/31-6/38390.html

†Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan (hkhwang@sinica.edu.tw). The work of this author was done while he was visiting the School of Computer Science, McGill University. He thanks the school for their hospitality and support.

‡School of Computer Science, McGill University, 3480 University Street, Montreal H3A 2K6, Canada (neiningr@cs.magill.ca). The work of this author was supported by NSERC grant A3450 and the Deutsche Forschungsgemeinschaft.

from an equally important binary search tree perspective, we can study a large number of extensions and variants (see Devroye [19] and Gonnet and Baeza-Yates [31]). Second, the inherent phase change of the limit laws from normal to nonnormal is a new, interesting phenomenon, which should also occur in many other structures. Third, how sensitive is the limit law of the cost with respect to the toll function? For such a simple structure, a certain "robustness" is expected. Also, the extent to which the normal law persists is helpful in giving a deeper understanding of the associated algorithms; roughly, the variance is increasing as the toll function grows, and the algorithms become less useful in practice if the variance is too large. Fourth, a complete characterization of the limit law under varying toll functions is still lacking in the literature. Fifth, the diverse examples we collected were the catalysts that stimulated our study.

The most studied special case is $T_n = n + O(1)$, which corresponds to the number of comparisons used by quicksort to sort a random input or, equivalently, to the total path length of a random binary search tree. It is known that

$$\frac{X_n - E(X_n)}{n} \xrightarrow{d} Y,$$

where "$\xrightarrow{d}$" denotes convergence in distribution. Here Y satisfies

$$Y \stackrel{d}{=} UY + (1-U)Y^* + 2U\log U + 2(1-U)\log(1-U) + 1, \tag{2}$$

where $Y \stackrel{d}{=} Y^*$, U is a uniform random variable over the unit interval, and Y, Y^*, and U are independent; see Rösler [66], Régnier [64], and Fill and Janson [26].

Other known cases leading to a normal limit law are

- the number of leaves in a random binary search tree for which $T_n = \delta_{n1}$, the Kronecker symbol (see Devroye [18, 20] and Flajolet, Gourdon, and Martínez [28]);
- the log-product of subtree sizes for which $T_n = \log n$ (see Fill [25]);
- the number of occurrences of any fixed pattern with T_n equal to the probability of the pattern when n is equal to the size of the pattern (see Flajolet, Gourdon, and Martínez [28]);
- the number of occurrence of subtrees of a given fixed size (see Aldous [1] and Devroye [18, 20]);
- the number of nodes whose subtree sizes are larger than a given page size $b \geq 1$ (see Flajolet, Gourdon, and Martínez [28]).

The case when $T_n = n^\alpha$, where $\alpha > 1$, was studied by Neininger [54]; this case leads again to nonnormal limit laws.

The rough picture reflected by these sporadic examples is that, if the toll function is small such as $\log n$ or $O(1)$, then the limit law of the total cost is normal, and that for the large toll function such as n it is nonnormal. However, when does the limit law of the total cost fail to be normal? We show, under general conditions, that $\sqrt{n}$ is roughly the separating line between normal and nonnormal limit laws; this is intuitively in accordance with the classical law of errors. For, from a structural point of view, if the toll function is small, then the contribution from each subproblem is not dominating so that the normal limit law is quite expected. (In vivid terms, the situation resembles the democratic system.) On the other hand, if the toll function is large, then the main contribution comes from a few subproblems of large size, rendering large variance and thus nonnormal law in the limit (totalitarian system?).

An even more intuitive guess is that, if $\mathrm{Var}(T_n) = o(\mathrm{Var}(X_n))$, then X_n would be asymptotically normally distributed; otherwise, the limit law would be nonnormal. This guess, although false in general, is true for the conditions we consider.

These examples will turn out to be special cases of our general results. We will discuss more new examples in section 6.

We give two different approaches, based, respectively, on the contraction method (see Rösler and Rüschendorf [69]) and the method of moments, to prove the different limit laws for several reasons. First, we propose the two approaches in a consistent and synthetic way so that they are likely to be applied to other algorithmic problems. Indeed, almost all asymptotic properties of the moments are encapsulated into an "asymptotic transfer" lemma, which relates the asymptotic behavior of the toll function T_n to that of the total cost X_n. Such a transfer also clarifies the sensitivity of the total cost with respect to the toll function (see also Devroye [20]). Second, each approach has its own advantages and inconveniences; we give them for more methodological interests. Third, both approaches can more or less be classified as "computational," in contrast to the "probabilistic" approach used by Devroye in the companion paper [20].

The contraction method, first introduced by Rösler [66] for the analysis in distribution of the quicksort algorithm (namely, (1) with $T_n = n - 1$), starts from a recursive equation satisfied by the random variable in question. Then one computes the first or the second moments, scales properly, proves that the scaled recurrence stabilizes in the limit, and chooses a suitable probability metric so that the stabilized equation defines a map of measures that is a contraction in this metric and has a unique fixed-point in some space of probability measures. The weak convergence of the scaled random variables to this fixed-point then follows from the contraction properties; see Rösler [67], Rachev and Rüschendorf [62], and Rösler [68] for more information and Rösler and Rüschendorf [69] for a survey. This approach is especially simple if the limiting map has contraction properties in the minimal L_2 metric. In this case, only knowledge of the first moment is required for the application of the method. This property will become clear in the case of "large" toll functions (very roughly, $E(T_n) \gg \sqrt{n}$). For "small" toll functions, the limiting equation necessitates the use of a probability metric that is ideal of order larger than two as well as information on the variance. In either case, a feature of the contraction method is that the dependence between T_n and I_n can be succinctly handled. For other applications of the contraction method, see [17, 50, 52, 56].

The method of moments, one of the most classical ways of deriving limit distributions, has been widely applied to problems in diverse fields (see, for example, Billingsley [8, section 30] and Diaconis [21]). It consists in first computing the mean and variance, properly scaling the random variable, computing by induction the higher moments of the scaled random variable, applying Carleman's criterion to justify the unicity of the limit law, and then concluding the convergence in distribution and of all moments (or convergence in L_p for all $p > 0$) by the Fréchet–Shohat moment convergence theorem (see Loève [44]). While the method of moments is usually used as the "last weapon" for proving limit laws, it does have some advantages: first, it provides more information than weak convergence; second, it is more transparent and self-contained and requires less advanced theory. We systematize the use of this method so that all major tasks boil down to the asymptotic transfer from the toll function to the total cost. Previously, this method was applied by Hennequin in his Ph.D. thesis [36, section IV.4] to characterize the limit laws of his generalized quick-

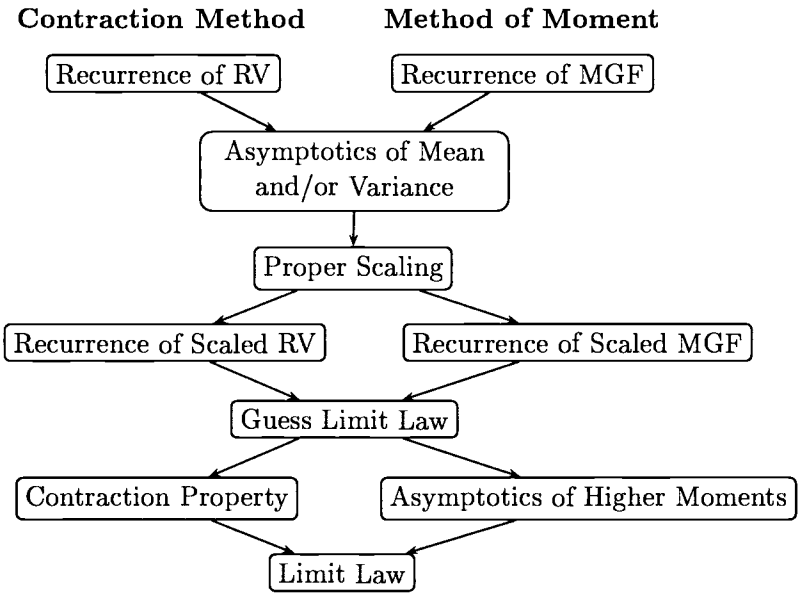


FIG. 1. *Main steps used by the contraction method and the method of moments. Here RV denotes "random variable," and MGF denotes "moment generating function."*

sort (covering, in particular, the quicksort with median-of-$(2k+1)$). His proof is, however, incomplete in that his Abelian lemma [36, p. 79] gives only an estimate inside the unit circle for the generating function in question so that his application of the singularity analysis (see Flajolet and Odlyzko [29]) is not fully justified. We use a different approach, more elementary in nature, to link the asymptotics of the toll function and that of the total cost. For recent applications of the method of moments to similar problems, see Fill [25], Flajolet, Poblete, and Viola [30], Dobrow and Fill [22], and Schachinger [70].

A schematic diagram illustrating the two approaches is given in Figure 1.

Typically, the method of moments requires more assumptions on the moments of the toll function than the contraction method, and the results obtained are stronger. On the other hand, it is also possible to obtain the convergence of all moments by the contraction method based on moment generating functions; see Rösler [66] for details. For another approach to recursive random variables, which we might term the "inductive approximation approach," see Pittel [58] and the references therein. See also [14, 56] for an interesting example for which the method of moments applies but the contraction method fails (the space requirement of random m-ary search trees when $m > 26$).

Viewing our results as bridging the transition from normal to "the quicksort law" (2), we can investigate other kinds of transitions by looking at different recurrences (or algorithms). A closely related recurrence to (1) is the one-sided quicksort recurrence

$$X_n \stackrel{d}{=} X_{I_n} + T_n, \tag{3}$$

for which we can vary the toll function to bridge the normal law and the Dickman

distribution; see [40]. Roughly, our results say that, if the toll function is of logarithmic order, then the limit law is normal; the limit law is nonnormal for larger toll functions; see section 7 for more precise results and examples.

For the class of problems we study in this paper and many others, an important feature distinguishing normal and nonnormal limit laws is the effect of cancellation caused by centering the random variable. Roughly, *the more cancellations of higher moments, the more likely the limit law is normal*. Since our settings cover almost all practical variations of the toll functions, the cancellation effect will be more "visible" in different cases, especially in the method of moments.

We give the main asymptotic transfer results in the next section. Then we prove the phase change of the limit laws in sections 3 and 4. We first give a more straightforward proof by the method of moments under stronger assumptions; then we apply the contraction method under more general settings. Continuities of the variation of the limit laws are discussed in section 5. We discuss many examples in section 6. In particular, the number of exchanges used by quicksort gives an intriguing example of T_n depending on I_n. Section 7 addresses a similar distribution theory for the recurrence (3); this is included because it is closely related.

Notation. Throughout this paper, X_n and T_n are related by (1). We use consistently the following notation: $x_n := E(X_n)$, $t_n := E(T_n)$, $P_n(y) := E(e^{X_n y})$, $Q_n(y) := E(e^{T_n y})$, $H_n := \sum_{1\le k\le n} 1/k$. The symbols U and $N(0,1)$ always represent a uniform $[0,1]$ and a standard normal random variable, respectively. The symbol n_0 denotes a suitable nonnegative integer whose value may vary from one occurrence to another. All unspecified limits (including O, o, $\sim$) are taken to be $n \to \infty$.

Slowly varying functions. A nonnegative function $L(n)$ defined for $n \ge n_0 \ge 0$ and not identically zero is called *slowly varying* if, for all real $\lambda > 0$,

$$L(n) \sim L(\lfloor \lambda n \rfloor) \qquad (n \to \infty).$$

If $n_0 > 0$, we define $L(n) = 0$ for $0 \le n < n_0$. Typical slowly varying functions include any powers of $\log n$ and $\log\log n$, $e^{(\log n)^\alpha (\log\log n)^\beta}$, where $0 \le \alpha, \beta < 1$, and $e^{\log n/\log\log n}$.

2. Mean and asymptotic transfers. We develop the main elementary tools in this section that will be used later. While the same results can be obtained via differential equations and suitable analytic tools, we content ourselves with the elementary approach due to the simplicity of the recurrence. See [15] for more general recurrences of quicksort type.

The mean $x_n := E(X_n)$ satisfies, by (1), $x_0 = 0$ and

$$x_n = \frac{2}{n} \sum_{0\le k<n} x_k + t_n \qquad (n \ge 1), \tag{4}$$

where $t_n := E(T_n)$.

LEMMA 1. *Let $\{b_n\}_{n\ge1}$ be a given sequence, and define a_n by $a_0 := 0$, and*

$$a_n = \frac{2}{n} \sum_{0\le k<n} a_k + b_n \qquad (n \ge 1). \tag{5}$$

Then, for $n \ge 1$,

$$a_n = b_n + 2(n+1) \sum_{1\le k<n} \frac{b_k}{(k+1)(k+2)}. \tag{6}$$

Proof. Take the difference $(n+1)a_n - na_n$, and then iterate the resulting recurrence. □

From this lemma, we obtain the exact solution for $E(X_n)$,

$$E(X_n) = t_n + 2(n+1)\sum_{1\le k<n}\frac{t_k}{(k+1)(k+2)}, \tag{7}$$

for $n \ge 1$; see Devroye [20] for a concrete interpretation of each term on the right-hand side of (7).

The main tool we need is the following lemma linking the asymptotic behavior of the toll function to that of the total cost.

LEMMA 2 (asymptotic transfers). *Assume that a_n satisfies* (5). (i) *The conditions $b_n = o(n)$ and $\sum_k b_k/k^2 < \infty$ are both necessary and sufficient for*

$$a_n \sim \Upsilon[b]n, \qquad \Upsilon[b] := 2\sum_{k\ge1}\frac{b_k}{(k+1)(k+2)};$$

(ii) *if $b_n \sim nL(n)$, then*

$$a_n \sim \begin{cases} \Upsilon[b]n & if\ \ \sum_{k\ge1} L(k)/k < \infty, \\ 2n\sum_{k\le n}\dfrac{L(k)}{k} & if\ \ \sum_{k\le n} L(k)/k \to \infty; \end{cases}$$

(iii) *if $b_n \sim n^\alpha L(n)$, where $\alpha > 1$, then*

$$a_n \sim \frac{\alpha+1}{\alpha-1}\, n^\alpha L(n).$$

Proof. The sufficiency part of (i) follows directly from the exact solution (6). For the necessary part, assume that $a_n \sim cn$ for some constant c. Then, by (4),

$$b_n = a_n - \frac{2}{n}\sum_{0\le k<n} a_k = o(n).$$

From this and (6), we deduce that $c = \Upsilon[b] < \infty$.

Part (ii) also results from (6) and the estimate (see Bingham, Goldie, and Teugels [9, Proposition 1.5.9a])

$$L(n) = o\left(\sum_{k\le n}\frac{L(k)}{k}\right). \tag{8}$$

For part (iii), we have

$$a_n \sim n^\alpha L(n) + 2n\sum_{1\le k\le n} k^{\alpha-2}L(k).$$

However,

$$\sum_{1\le k\le n} k^{\alpha-2}L(k) \sim L(n)\sum_{1\le k\le n} k^{\alpha-2} \sim \frac{n^{\alpha-1}}{\alpha-1}\, L(n);$$

see Proposition 1.5.8 of Bingham, Goldie, and Teugels [9, p. 26]. □

Remarks. 1. If $b_n = o(\sqrt{n})$, then $a_n = \Upsilon[b]n + o(\sqrt{n})$.

2. If $b_n \sim n^\alpha L(n)$, where $\alpha \geq 1/2$, $\alpha \neq 1$, then

$$a_n = \Upsilon[b]n + \frac{\alpha+1}{\alpha-1} n^\alpha L(n)(1+o(1)). \tag{9}$$

3. If we replace the two $\sim$'s for b_n in the lemma by $O(.)$ (or $o(.)$) in cases (ii) and (iii), then the same results hold by replacing $\sim$ for a_n by $O(.)$ (or $o(.)$).

3. Limit laws. I. Method of moments. We study the limit laws of X_n. Briefly, we derive weak convergence and convergence of all moments of X_n (properly normalized) to some Y when estimates for moments of T_n are available. We consider mainly the case when T_n is independent of I_n. The case when T_n depends on I_n requires a straightforward extension of the method; we will discuss briefly the dependence extension for small toll functions; the large toll functions will be discussed via examples in section 6.

Let $P_n(y) := E(e^{X_n y})$. Then from (1) and independence

$$P_n(y) = \frac{Q_n(y)}{n} \sum_{0\leq k<n} P_k(y) P_{n-1-k}(y) \qquad (n \geq 1),$$

with $P_0(y) := 1$, where $Q_n(y) := E(e^{T_n y})$.

Before going further, we need to discard the special case when $T_n = c$ for $n \geq 1$, which yields $X_n = cn$ for $n \geq 1$.

LEMMA 3. *Assume that I_n and T_n are independent for $n \geq 1$. The variance of X_n is zero for $n \geq 1$ iff $T_n = c$ for $n \geq 1$ and for some constant c.*

Proof. Let $\phi_n(y) := e^{-x_n y} P_n(y)$. Then $\phi_n'(0) = 0$, and

$$\phi_n''(0) = \mathrm{Var}(X_n) = \frac{2}{n} \sum_{0\leq k<n} \phi_k''(0) + \psi_n \qquad (n \geq 2),$$

where, defining $\Delta_{n,k} = x_k + x_{n-1-k} - x_n$,

$$\begin{aligned} \psi_n &= Q_n''(0) + \frac{2}{n} t_n \sum_{0\leq k<n} \Delta_{nk} + \frac{1}{n} \sum_{0\leq k<n} \Delta_{nk}^2 \\ &= E(T_n^2) - (E(T_n))^2 + \frac{1}{n} \sum_{0\leq k<n} \Delta_{nk}^2 - \left(\frac{1}{n} \sum_{0\leq k<n} \Delta_{nk} \right)^2. \end{aligned} \tag{10}$$

The assertion of the lemma follows from the Cauchy–Schwarz inequality and induction. □

Define $Y_\alpha = Y_\alpha(T)$ by

$$Y_\alpha \stackrel{d}{=} \begin{cases} U^\alpha Y_\alpha + (1-U)^\alpha Y_\alpha^* + T & \text{if } \alpha > 1/2,\ \alpha \neq 1, \\ UY + (1-U)Y^* + h(T,U) & \text{if } \alpha = 1, \end{cases} \tag{11}$$

where $Y \stackrel{d}{=} Y^*$ and Y, Y^*, T, U are independent and $h(T,U) = 2U\log U + 2(1-U)\log(1-U) + T$. Here T is essentially the limit distribution of T_n/t_n. It will turn out that Y_α is the limit law of X_n after properly normalized. From this defining

equation, it follows that the mth moment of Y_α, denoted by η_m, satisfies, if it exists, the recurrence $\eta_0 = 1$ and, for $m \geq 1$,

$$\eta_m = \begin{cases} \displaystyle\sum_{a+b+c=m} \binom{m}{a,b,c} \tau_a \eta_b \eta_c B(b\alpha+1, c\alpha+1) & \text{if } \alpha > 1/2,\ \alpha \neq 1, \\ \displaystyle\sum_{a+b+c+d=m} \binom{m}{a,b,c,d} \tau_a \eta_b \eta_c \int_0^1 x^{b\alpha}(1-x)^{c\alpha} \Lambda(x)^d \,\mathrm{d}x & \text{if } \alpha = 1, \end{cases} \tag{12}$$

where $\tau_m = E(T^m)$, $\Lambda(x) := 2x\log x + 2(1-x)\log(1-x)$, and $B(u,v)$ denotes the beta integral

$$B(u,v) := \int_0^1 x^{u-1}(1-x)^{v-1}\,\mathrm{d}x = \frac{\Gamma(u)\Gamma(v)}{\Gamma(u+v)} \qquad (u,v>0),$$

Γ being the Gamma function. Also, the moment generating function $\eta(z) := E(e^{Y_\alpha z})$ satisfies

$$\eta(z) = \begin{cases} \tau(z) \int_0^1 \eta(x^\alpha z)\eta((1-x)^\alpha z)\,\mathrm{d}x & \text{if } \alpha > 1/2,\ \alpha \neq 1, \\ \tau(z) \int_0^1 \eta(x^\alpha z)\eta((1-x)^\alpha z) e^{\Lambda(x)z}\,\mathrm{d}x & \text{if } \alpha = 1, \end{cases}$$

provided that both $\eta(z)$ and $\tau(z) := E(e^{Tz})$ exist.

LEMMA 4. *Assume that $\alpha > 0$. If the moment generating function of T exists, then $\{\eta_m\}_m$ characterizes uniquely the distribution $\mathcal{L}(Y_\alpha)$.*

Proof. Assume that the series $\sum_m \tau_m z^m/m!$ converges for $|z| \leq \delta$ for some $\delta > 0$. We show that $|\eta_m| \leq m! K^m$ for a sufficiently large K. By induction, using (12), we have

$$\begin{aligned} \frac{|\eta_m|}{m!} &\leq \frac{m\alpha+1}{m\alpha-1} \sum_{0\leq b<m} \int_0^1 x^{b\alpha} \sum_{0\leq a\leq m-b} \frac{|\tau_a|}{a!} K^{m-a}(1-x)^{(m-a-b)\alpha}\,\mathrm{d}x \\ &\leq K^m \left(\sum_{a\geq 0} \frac{|\tau_a|}{a!}K^{-a}\right) \frac{m\alpha+1}{m\alpha-1} \sum_{0\leq b<m} \int_0^1 x^{b\alpha}(1-x)^{(m-b)\alpha}\,\mathrm{d}x \\ &\leq K^m \left(\sum_{a\geq 0} \frac{|\tau_a|}{a!}K^{-a}\right) \frac{m\alpha+1}{m\alpha-1} \left(\frac{1}{m\alpha+1} + (m-1)\int_0^1 x^\alpha(1-x)^{(m-1)\alpha}\,\mathrm{d}x\right) \\ &= K^m \left(\sum_{a\geq 0} \frac{|\tau_a|}{a!}K^{-a}\right) \left[\frac{1}{m\alpha-1} + \frac{(m\alpha+1)(m-1)\Gamma(\alpha+1)\Gamma(m\alpha+1-\alpha)}{(m\alpha-1)(m\alpha+1)\Gamma(m\alpha+1)}\right]. \end{aligned}$$

Take first K so large that the series $\sum_a |\tau_a| K^{-a}/a!$ converges. Then, since the terms in brackets tend to zero as $m \to \infty$, there exists an $m_0 > 0$ such that

$$\left(\sum_{a\geq 0} \frac{|\tau_a|}{a!}K^{-a}\right) \left[\frac{1}{m\alpha-1} + \frac{(m\alpha+1)(m-1)\Gamma(\alpha+1)\Gamma(m\alpha+1-\alpha)}{(m\alpha-1)(m\alpha+1)\Gamma(m\alpha+1)}\right] < 1$$

for $m > m_0$. On the other hand, $|\eta_m| \leq m! K^m$ for $m \leq m_0$ if K was chosen sufficiently large. We conclude that $|\eta_m| \leq m! K^m$, and the required assertion then follows from Carleman's criterion stating that *the moment sequence $\{\eta_m\}_m$ uniquely characterizes a distribution if* $\sum_m \eta_{2m}^{-1/(2m)} = \infty$. ∎

The condition we impose (that $\tau(z)$ exists) is certainly far from optimal but is sufficient for practical applications.

Define

$$\Upsilon[t] := 2\sum_{k\geq 1}\frac{t_k}{(k+1)(k+2)}.$$

THEOREM 1 (large toll functions). *Let (X_n) be given by (1), where T_n is independent of I_n. If*

$$E(T_n)\sim n^\alpha L(n) \quad\text{and}\quad E\left(\frac{T_n}{t_n}\right)^m \to \tau_m \qquad (m=2,3,\dots), \tag{13}$$

where $\alpha > 1/2$, and $\tau(z) := \sum_m \tau_m z^m/m!$ exists, then

$$\frac{X_n-\xi_n}{n^\alpha L(n)} \xrightarrow{d} Y_\alpha,$$

with convergence of all moments, where $Y_\alpha = Y_\alpha(T)$ is defined as above and

$$\xi_n = \begin{cases} \Upsilon[t]n & \text{if } 1/2<\alpha<1,\\ E(X_n) & \text{if } \alpha=1,\\ 0 & \text{if } \alpha>1. \end{cases}$$

For small toll functions, we distinguish two overlapping cases: (i)

$$t_n = O(\sqrt{n}/(\log n)^{1/2+\varepsilon}) \quad\text{and}\quad E(T_n^m)=O(t_n^m) \tag{14}$$

for $m=2,3,\dots$; and (ii)

$$t_n\sim\sqrt{n}L(n),\quad E(T_n^2)\sim\tau_2 t_n^2,\quad\text{and}\quad E(T_n^m)=O(t_n^m) \tag{15}$$

for $m=3,4,\dots$. More general conditions can be studied, but we content ourselves with these two for simplicity of presentation.

In the first case, define

$$s^2(n) := \sigma^2 n, \qquad \sigma^2 := \Upsilon[\psi] = 2\sum_{k\geq 1}\frac{\psi_k}{(k+1)(k+2)}, \tag{16}$$

where ψ_k is given in (10). In the second case, define $s(n)$ as in (16) if $\sum_{k\geq 1} L^2(k)/k < \infty$ and

$$s^2(n) := \left(\frac{9}{2}\pi - 16 + 2\tau_2\right) n\sum_{k\leq n}\frac{L^2(k)}{k} \tag{17}$$

if $\sum_{k\leq n} L^2(k)/k\to\infty$. Note that $\sigma>0$ by (10) and (16), and the leading constant $\frac{9}{2}\pi - 16 + 2\tau_2$ is positive since $\frac{9}{2}\pi > 14$ and $\tau_2\geq 1$ by $E(T_n^2)\geq t_n^2$.

THEOREM 2 (small toll functions). *Let (X_n) be given by (1), where T_n is independent of I_n. If T_n satisfies either (14) or (15), then*

$$\frac{X_n-\Upsilon[t]n}{s(n)} \xrightarrow{d} N(0,1),$$

with mean and variance satisfying $E(X_n)\sim\Upsilon[t]n$ and $\operatorname{Var}(X_n)\sim s^2(n)$. The limit holds with convergence of all moments.

The proof uses the method of moments and the asymptotic transfer lemma.

3.1. Large toll functions. For simplicity of presentation, we split the proof of Theorem 1 into three cases, $1/2 < \alpha < 1$, $\alpha = 1$, and $\alpha > 1$, although we can easily encapsulate them into one.

Case L1. $1/2 < \alpha < 1$. In this case, $\xi_n = \Upsilon[t]n$, and, by (7),

$$E(X_n) \sim \Upsilon[t]n + \frac{\alpha+1}{\alpha-1} n^\alpha L(n).$$

Shift the mean by $\Upsilon[t]n$ by defining $\Pi_n(y) := P_n(y)e^{-\Upsilon[t]ny}$. Then $\Pi_0(y) = 1$, and

$$\Pi_n(y) = \frac{Q_n(y)e^{-\Upsilon[t]y}}{n} \sum_{0\le k<n} \Pi_k(y)\Pi_{n-1-k}(y) \qquad (n \ge 1). \tag{18}$$

Taking m times derivatives with respect to y on both sides and then substituting $y = 0$, we have, by defining $\Pi_{n,m} := \Pi_n^{(m)}(0) = E((X_n - \Upsilon[t]n)^m)$,

$$\Pi_{n,m} = \frac{2}{n} \sum_{0\le k<n} \Pi_{k,m} + R_{n,m} \qquad (n \ge 1), \tag{19}$$

with $\Pi_{0,m} = 0$, where

$$R_{n,m} := \sum_{\substack{a+b+c+d=m\\ b,c<m}} \binom{m}{a,b,c,d} Q_n^{(a)}(0) \frac{(-\Upsilon[t])^d}{n} \sum_{0\le k<n} \Pi_{k,b}\Pi_{n-1-k,c}.$$

By assumption (13),

$$Q_n^{(m)}(0) = E(T_n^m) \sim \tau_m n^{m\alpha} L^m(n) \qquad (m \ge 1).$$

For convenience, define $\tau_0 = \tau_1 = 1$. We proceed by induction. Assume

$$\Pi_{n,m} \sim g_m n^{m\alpha} L^m(n). \tag{20}$$

This holds true for $m = 1$ by (9) with $g_1 = (\alpha+1)/(\alpha-1)$. By induction and slow variation of $L(n)$, we deduce that, for $m \ge 2$,

$$\begin{aligned}
&R_{n,m}\\
&\sim \sum_{\substack{a+b+c=m\\ b,c<m}} \binom{m}{a,b,c} \tau_a g_b g_c n^{a\alpha-1} L^a(n) \sum_{0\le k<n} k^{b\alpha} L^b(k)(n-1-k)^{c\alpha} L^c(n-1-k)\\
&\sim L^m(n) \sum_{\substack{a+b+c=m\\ b,c<m}} \binom{m}{a,b,c} \tau_a g_b g_c n^{a\alpha-1} \sum_{0\le k<n} k^{b\alpha}(n-1-k)^{c\alpha}\\
&\sim n^{m\alpha} L^m(n) \sum_{\substack{a+b+c=m\\ b,c<m}} \binom{m}{a,b,c} \tau_a g_b g_c B(b\alpha+1, c\alpha+1).
\end{aligned}$$

It follows, by the asymptotic transfer lemma, that

$$\Pi_{n,m} \sim \frac{m\alpha+1}{m\alpha-1} n^{m\alpha} L^m(n) \sum_{\substack{a+b+c=m\\ b,c<m}} \binom{m}{a,b,c} \tau_a g_b g_c B(b\alpha+1, c\alpha+1) \qquad (m \ge 2).$$

Thus, if we define g_m recursively by

$$g_m = \frac{m\alpha+1}{m\alpha-1} \sum_{\substack{a+b+c=m\\ b,c<m}} \binom{m}{a,b,c} \tau_a g_b g_c B(b\alpha+1, c\alpha+1) \qquad (m \ge 2),$$

then (20) holds for all $m \ge 1$. Note that $g_m = \eta_m$ for $m \ge 1$; see (12). We conclude, by the Fréchet–Shohat moment convergence theorem (see [44]) and Lemma 4, that $\{\eta_m\}$ is the sequence of moments of some distribution function and that $(X_n - \Upsilon[t]n)/(n^\alpha L(n))$ converges in distribution to Y_α.

Case L2. $\alpha = 1$. Define this time $\Pi_n(y) := P_n(y)e^{-x_n y}$, where $x_n = E(X_n)$. Then $\Pi_0(y) = 1$, and

$$\Pi_n(y) = \frac{Q_n(y)}{n} \sum_{0\le k<n} \Pi_k(y)\Pi_{n-1-k}(y) e^{\Delta_{n,k}y} \qquad (n \ge 1),$$

where $\Delta_{n,k} = x_k + x_{n-1-k} - x_n$. Observe first, by (7), that

$$\begin{aligned}\Delta_{n,k} = t_k + t_{n-1-k} - t_n &- 2(k+1) \sum_{k\le j<n} \frac{t_j}{(j+1)(j+2)} \\ &- 2(n-k) \sum_{n-k<j<n} \frac{t_j}{(j+1)(j+2)};\end{aligned}$$

from this and $t_n \sim nL(n)$, we deduce that, for $k = \lfloor xn \rfloor$,

$$\Delta_{n,k} \sim \Lambda(x) nL(n), \qquad \Lambda(x) := 2x\log x + 2(1-x)\log(1-x), \tag{21}$$

uniformly for $0 \le x \le 1$.

Write as above $\Pi_{n,m} = \Pi_{n,m}(0)$. Then $\Pi_{n,m}$ satisfies (19) with

$$R_{n,m} := \sum_{\substack{a+b+c+d=m\\ b,c<m}} \binom{m}{a,b,c,d} \frac{Q_n^{(a)}(0)}{n} \sum_{0\le k<n} \Pi_{k,b}\Pi_{n-1-k,c}\Delta_{n,k}^d.$$

Note that $\Pi_{n,1} = 0$. We prove by induction that

$$\Pi_{n,m} \sim g_m n^m L^m(n). \tag{22}$$

The case $m = 1$ is true with $g_1 = 0$. For $m \ge 2$, we have, similarly as above,

$$\begin{aligned} R_{n,m} &\sim \sum_{\substack{a+b+c+d=m\\ b,c<m}} \binom{m}{a,b,c,d} \tau_a g_b g_c n^{a-1} L^a(n) \\ &\qquad \times \sum_{0\le k<n} k^b L^b(k)(n-1-k)^c L^c(n-1-k)\Delta_{n,k}^d \\ &\sim n^m L^m(n) \sum_{\substack{a+b+c+d=m\\ b,c<m}} \binom{m}{a,b,c,d} \tau_a g_b g_c \int_0^1 x^b(1-x)^c \Lambda^d(x)\,\mathrm{d}x. \end{aligned}$$

It follows, by Lemma 2, that (22) holds with

$$g_m = \frac{m+1}{m-1} \sum_{\substack{a+b+c+d=m \\ b,c<m}} \binom{m}{a,b,c,d} \tau_a g_b g_c \int_0^1 x^b (1-x)^c \Lambda^d(x)\,\mathrm{d}x \qquad (m \ge 2).$$

Thus convergence in distribution follows as in Case L1.

Note that $\mathrm{Var}(X_n) \sim g_2 n^2 L^2(n)$, where

$$g_2 = 7 - \frac{2}{3}\pi^2 + 3\mathrm{Var}(T). \tag{23}$$

Case L3. $\alpha > 1$. In this case, no centering is needed since $\xi_n = 0$. We apply mutatis mutandis the same argument in Case L1 for $P_n(y)$. The proof is similar and is omitted here.

3.2. Small toll functions. When t_n is small, namely, $t_n = O(\sqrt{n}L(n))$, $E(X_n)$ is linear, so we center X_n as in Case L1 above by defining $\Pi_n(y) := P_n(y)e^{-\Upsilon[t]ny}$. Write again $\Pi_{n,m} := \Pi_n^{(m)}(0)$. Then $\Pi_n(y)$ satisfies (18) and

$$\Pi_{n,1} = \Pi_n'(0) = t_n - 2n\sum_{j\ge n} t_j j^{-2} + O(1).$$

Variance. By (18), the sequence $\Pi_{n,2}$ satisfies (19) with

$$R_{n,2} = Q_n''(0) - t_n^2 + (t_n - \Upsilon[t])(2\Pi_{n,1} - t_n + \Upsilon[t]) + \frac{2}{n}\sum_{0\le k<n} \Pi_{k,1}\Pi_{n-1-k,1}$$

(the recurrence of $\Pi_{n,1}$ being used to simplify).

Thus, by the asymptotic transfer lemma, if $\sum_k R_{k,2}/k^2 < \infty$, then

$$\mathrm{Var}(X_n) \sim \sigma^2 n, \qquad \sigma^2 := 2\sum_{k\ge 1} \frac{R_{k,2}}{(k+1)(k+2)}. \tag{24}$$

However, the condition $\sum_k R_{k,2}/k^2 < \infty$ is not so transparent. We thus consider two simple overlapping cases.

Case S1. $t_n = O(\sqrt{n}(\log n)^{-1/2-\varepsilon})$. In this case, we have

$$\Pi_{n,1} = O(\sqrt{n}(\log n)^{-1/2-\varepsilon})$$

and, by (14),

$$R_{n,2} = O(n(\log n)^{-1-2\varepsilon});$$

thus

$$\mathrm{Var}(X_n) \sim \sigma^2 n.$$

Case S2. $t_n \sim \sqrt{n}L(n)$. By (9),

$$\Pi_{n,1} \sim -3\sqrt{n}L(n),$$

from which we deduce, using (15), that

$$R_{n,2} \sim \left(\frac{9}{4}\pi - 8 + \tau_2\right) nL^2(n).$$

Applying Lemma 2 yields

$$\mathrm{Var}(X_n) \sim \begin{cases} \sigma^2 n & \text{if } \sum_k L^2(k)/k < \infty, \\ \left(\dfrac{9}{2}\pi - 16 + 2\tau_2\right) n \displaystyle\sum_{k\le n} \frac{L^2(k)}{k} & \text{if } \sum_k L^2(k)/k = \infty. \end{cases}$$

Note that the definition of σ^2 in (24) can be shown to be identical to (16).

Asymptotic normality. For higher moments, we again use (19) but split $R_{n,m}$ into two parts: $R_{n,m} = R^{(1)}_{n,m} + R^{(2)}_{n,m}$, where

$$R^{(1)}_{n,m} := \sum_{1\le b<m} \binom{m}{b} \frac{1}{n} \sum_{0\le k<n} \Pi_{k,b}\Pi_{n-1-k,m-b},$$

$$R^{(2)}_{n,m} := \sum_{\substack{a+b+c+d=m \\ a+d\ge 1 \\ b,c<m}} \binom{m}{a,b,c,d} Q^{(a)}_n \frac{(-\Upsilon[t])^d}{n} \sum_{0\le k<n} \Pi_{k,b}\Pi_{n-1-k,c}.$$

Case S1. $t_n = O(\sqrt{n}(\log n)^{-1/2-\varepsilon})$. Assume that, for $m \ge 1$,

$$\begin{cases} \Pi_{n,2m} \sim g_m n^m, \\ \Pi_{n,2m-1} = o(n^{m-1/2}). \end{cases}$$

This is true for $m = 1$ with $g_1 = \sigma^2$. By induction, we have

$$R^{(2)}_{n,m} = O(n^{m/2}(\log n)^{-1/2-\varepsilon}).$$

The main contribution for even moments comes from $R^{(1)}_{n,m}$.

$$\begin{aligned} R^{(1)}_{n,2m} &\sim \sum_{1\le j<m} \binom{2m}{2j} \frac{1}{n} \sum_{0\le k<n} \Pi_{k,2j}\Pi_{n-1-k,2m-2j} \\ &\sim \sum_{1\le j<m} \binom{2m}{2j} g_j g_{m-j} \frac{1}{n} \sum_{0\le k<n} k^j (n-1-k)^{m-j} \\ &\sim \frac{n^m}{m+1} \sum_{1\le j<m} \frac{\binom{2m}{2j}}{\binom{m}{j}} g_j g_{m-j}. \end{aligned}$$

By Lemma 2,

$$\Pi_{n,2m} \sim \frac{n^m}{m-1} \sum_{1\le j<m} \frac{\binom{2m}{2j}}{\binom{m}{j}} g_j g_{m-j} \qquad (m \ge 2).$$

Thus we take g_m so that $g_1 = \sigma^2$ and

$$g_m = \frac{1}{m-1} \sum_{1\le j<m} \frac{\binom{2m}{2j}}{\binom{m}{j}} g_j g_{m-j} \qquad (m \ge 2).$$

The solution is given by

$$g_m = \frac{(2m)!}{2^m m!}\, \sigma^{2m} \qquad (m \ge 1),$$

which denotes the $2m$th moment of the normal distribution with mean zero and variance σ^2.

Similarly, for $m \geq 2$,

$$R^{(1)}_{n,2m-1} = o(m^{m-1/2}),$$

and thus, by the o-version of Lemma 2,

$$\Pi_{n,2m-1} = o(n^{m-1/2}).$$

The asymptotic normality follows.

Case S2. $t_n \sim \sqrt{n}L(n)$. In this case, noting that $L^2(n)$ is also slowly varying, we have (see (8))

$$Q_n''(0) \sim \tau_2 t_n^2 \sim \tau_2 n L^2(n) = o\left(n \sum_{k \leq n} \frac{L^2(k)}{k}\right).$$

In particular, if $\sum_k L^2(k)/k < \infty$, then $t_n = o(\sqrt{n}\,)$. The proof then follows the same line of argument as in Case S1.

3.3. T_n depends on I_n. In this case, we have $P_0(y) = 1$ and

$$P_n(y) = \frac{1}{n} \sum_{0 \leq k < n} P_k(y) P_{n-1-k}(y) Q_{n,k}(y) \qquad (n \geq 1),$$

where $Q_{n,k}(y)$ is the moment generating function of T_n conditioned on $I_n = k$.

First, Lemma 3 still holds since ψ_n satisfies

$$\psi_n = \frac{1}{n} \sum_{0 \leq k < n} \left(Q''_{n,k}(0) - Q'_{n,k}(0)^2\right) + \frac{1}{n} \sum_{0 \leq k < n} \left(Q'_{n,k}(0) + \Delta_{n,k}\right)^2, \tag{25}$$

and the same argument applies.

A full extension of the limit laws of X_n to this case requires more assumptions on the asymptotic behavior of $Q_{n,k}(y)$. There is, however, a special case for which the extension is trivial: Case S1, namely, when T_n satisfies (14). The asymptotic normality holds without any additional assumptions. Intuitively, this is the case when each toll summand has only limited contribution to the total cost; thus whether T_n depends on I_n or not does not change the "democratic" nature of the problem, rendering the same law of errors to take effect. Case S2 needs one more condition (26), and the extension is also straightforward.

Define $s(n)$ as in (16) with ψ_k replaced by (25) when T_n satisfies (14). In the case when $t_n \sim \sqrt{n}L(n)$, we need, in addition to (15), the following estimate:

$$E(T_n \sqrt{I_n} L(I_n)) + E(T_n \sqrt{n-1-I_n} L(n-1-I_n)) \sim \tau_2' n L^2(n). \tag{26}$$

Define $s(n)$ by

$$s^2(n) := \left(2\tau_2 - 6\tau_2' + \frac{9}{2}\pi\right) n \sum_{k \leq n} \frac{L^2(k)}{k}$$

if $\sum_k L^2(k)/k$ diverges and $s^2(n) := \sigma^2 n$ otherwise.

THEOREM 2′ (small toll functions—T_n dependent on I_n). *Let* (X_n) *be given by* (1). *If* T_n *satisfies either* (14) *or the two estimates* (15) *and* (26), *then*

$$\frac{X_n - \Upsilon[t]n}{s(n)} \xrightarrow{d} N(0,1),$$

with mean and variance satisfying $E(X_n) \sim \Upsilon[t]n$ *and* $\mathrm{Var}(X_n) \sim s^2(n)$. *In either case, convergence of all moments holds.*

The proof of Theorem 2 requires only minor modifications, and the $R_{n,2}$ there should be replaced by

$$\begin{aligned} R_{n,2} = \frac{1}{n}\sum_{0\le k<n} Q''_{n,k}(0) - 2\Upsilon[t]\Pi_{n,1} - \Upsilon[t]^2 \\ + \frac{2}{n}\sum_{0\le k<n} Q'_{n,k}(0)\left(\Pi_{k,1} + \Pi_{n-1-k,1}\right) \\ + \frac{2}{n}\sum_{0\le k<n} \Pi_{k,1}\Pi_{n-1-k,1}. \end{aligned}$$

We leave aside the discussions of large toll functions since (i) such cases can be succinctly incorporated in the settings by the contraction method, (ii) Theorem 2′ covers most practical applications, and (iii) we will describe one such example in section 6.

4. Limit laws. II. Contraction method. We consider the limit laws of X_n using the contraction method in this section. An advantage of this approach is that dependence of T_n on I_n can be easily handled.

4.1. Outline of the method. According to our discussions in the previous section, we first introduce the standardized versions (Y_n) of (X_n) by $Y_0 := 0$ and

$$Y_n := \frac{X_n - x_n}{s(n)} \qquad (n \ge 1),$$

where $s(n) > 0$ is an appropriate scaling to be defined later.

We first sketch the method of proof. The first step is to transform the original recurrence (1) into a modified recurrence for the scaled quantities (Y_n),

$$Y_n \overset{d}{=} A_1^{(n)} Y_{I_n} + A_2^{(n)} Y^*_{n-1-I_n} + b_n \qquad (n \ge 1), \tag{27}$$

where, for $n \ge 1$, $A_1^{(n)} = s(I_n)/s(n)$, $A_2^{(n)} = s(n-1-I_n)/s(n)$, and

$$b_n = \frac{1}{s(n)}\left(x_{I_n} + x_{n-1-I_n} - x_n + T_n\right) =: h_n(T_n, I_n). \tag{28}$$

According to (1), (Y_n), (Y^*_n), and $(A_1^{(n)}, A_2^{(n)}, b_n)$ are independent, and $Y_n \overset{d}{=} Y^*_n$ for all $n \ge 0$. Note that the value $s(0)$ can be chosen arbitrarily since $Y_0 = 0$.

If the coefficients $A_1^{(n)}$, $A_2^{(n)}$, and the additive term b_n converge as $n \to \infty$, say, to A_1, A_2, and b, respectively, and we expect that (Y_n) converges in distribution, then the weak limit Y of (Y_n) should satisfy the limiting equation corresponding to (27):

$$Y \overset{d}{=} A_1 Y + A_2 Y^* + b, \tag{29}$$

where Y, Y^*, and (A_1, A_2, b) are independent and $Y \stackrel{d}{=} Y^*$.

The contraction method then proceeds by showing that a fixed-point equation like (29) has exactly one solution in a certain space of probability measures and that the scaled random variables under consideration converge in distribution to this fixed-point.

Usually, the existence and uniqueness of a fixed-point of the limiting equation in a subspace of probability distributions is shown by endowing the subspace with a metric and by proving that the limiting equation defines a contraction map on this space. Then the existence of a unique fixed-point is implied by Banach's fixed-point theorem in the case of a complete metric or an appropriate substitute in the incomplete case.

In particular, the minimal L_2-metric ℓ_2 is often used, where ℓ_r-metrics are defined on the spaces $\mathcal{M}_r$ of probability measures on the Borel σ-algebra of $\mathbb{R}$ with finite absolute rth moment by

$$\ell_r(\nu, \varrho) := \inf\{\|X - Y\|_r \,:\, X \stackrel{d}{=} \nu, Y \stackrel{d}{=} \varrho\} \qquad (\nu, \varrho \in \mathcal{M}_r)$$

for $r \geq 1$. We denote by $\mathcal{M}_r(0) \subset \mathcal{M}_r$ the subspace of the centered probability measures in $\mathcal{M}_r$. The metric spaces $(\mathcal{M}_r, \ell_r)$ and $(\mathcal{M}_r(0), \ell_r)$ are complete, and convergence in the ℓ_r-metric is equivalent to weak convergence and convergence of the rth absolute moment. For simplicity, we write $\ell_r(X, Y) := \ell_r(\mathcal{L}(X), \mathcal{L}(Y))$. The infimum in the definition of ℓ_r is attained for all $\nu, \varrho \in \mathcal{M}_r$, and (X, Y) are called optimal couplings of ν, ϱ if $\ell_r(\nu, \varrho) = \|X - Y\|_r$; see Bickel and Freedman [7], Rachev [60], and Rachev and Rüschendorf [63] for more properties of the minimal L_r-metric.

The existence of a unique fixed-point $\mathcal{L}(Y)$ in $\mathcal{M}_2(0)$ for (29) and the convergence in ℓ_2 of (Y_n) given by (27) to Y holds particularly if the following properties are satisfied (see Rösler [68]):

(a) $E(b_n) = E(b) = 0, \quad E(b^2) < \infty$.
(b) $\|(A_1^{(n)}, A_2^{(n)}, b_n) - (A_1, A_2, b)\|_2 \to 0$.
(c) $E(A_1^2) + E(A_2^2) < 1$.
(d) For all $n_1 \in \mathbb{N}$, $E[\mathbf{1}_{\{I_n \leq n_1\}}(A_1^{(n)})^2] + E[\mathbf{1}_{\{n-1-I_n \leq n_1\}}(A_2^{(n)})^2] \to 0$.

This is the line we will follow for large toll functions T_n. In the case of small toll functions, we will end up with a well-known limiting equation that is not a contraction on $(\mathcal{M}_2(0), \ell_2)$ and has the normal distributions as solutions. In this case, asymptotic normality will be derived by a change of the metric as used in Rachev and Rüschendorf [62]. The metric used later is ideal of order larger than two, which implies the contraction properties of the limiting equation with respect to this metric on the appropriate space.

4.2. Large toll functions. Assume that

$$E(T_n) \sim n^\alpha L(n) \quad \text{and} \quad \left(\frac{T_n}{E(T_n)}, \frac{I_n}{n}\right) \xrightarrow{L_2} (T, U),$$

where $\alpha > 1/2$, $L(n)$ is slowly varying, and T is square-integrable. In particular, T_n may depend on I_n, and T may depend on U. For our applications to quicksort and binary search trees, this U comes up (essentially) as the first partitioning element of quicksort (or the root of the associated binary search tree). Therefore, I_n has, conditioned on $U = u$, the binomial $B(n-1, u)$ distribution, and $I_n/n \to U$ holds in L_p for all $p > 0$.

For the scaling factor, we assume at the moment that the variance of X_n admits an expansion of the form

$$\operatorname{Var}(X_n) \sim \sigma^2 n^{2\alpha} L^2(n),$$

where $\sigma = \sigma(\alpha, (T, U))$ is a positive constant given later in Corollary 1. This will later turn out to be true (up to degenerate cases). Therefore, we use the scaling $s(n) := n^\alpha L(n)$ and define $Y_0 := 0$ and

$$Y_n := \frac{X_n - x_n}{n^\alpha L(n)} \qquad (n \geq 1).$$

Note that the initial values of L have to be chosen such that $L(n) > 0$ for all $n \geq 1$. We have, for $n \geq 1$,

$$\begin{aligned} Y_n \stackrel{d}{=} \left(\frac{I_n}{n}\right)^\alpha \frac{L(I_n)}{L(n)} Y_{I_n} + \left(\frac{n-1-I_n}{n}\right)^\alpha \frac{L(n-1-I_n)}{L(n)} Y^*_{n-1-I_n} \\ + h_n(\alpha, (T_n, I_n)), \end{aligned} \tag{30}$$

where $h_n(\alpha, (T_n, I_n)) := (x_{I_n} + x_{n-1-I_n} - x_n + T_n)/(n^\alpha L(n))$.

Observe that our formal L_2-convergence assumption on I_n/n is equivalent to $I_n/n \to U$ in L_p for all $p \geq 0$. Using this and the estimate

$$\frac{L(I_n)}{L(n)} \xrightarrow{L_2} 1,$$

we obtain

$$\begin{aligned} \left\| \left(\frac{I_n}{n}\right)^\alpha \frac{L(I_n)}{L(n)} - U^\alpha \right\|_2 &\leq \left\| \left(\frac{I_n}{n}\right)^\alpha - U^\alpha \right\|_2 + \left\| \left(\frac{I_n}{n}\right)^\alpha \left(\frac{L(I_n)}{L(n)} - 1\right) \right\|_2 \\ &= o(1) + \left\| \frac{L(I_n)}{L(n)} - 1 \right\|_2 \to 0. \end{aligned} \tag{31}$$

Analogously,

$$\left\| \left(\frac{n-1-I_n}{n}\right)^\alpha \frac{L(n-1-I_n)}{L(n)} - (1-U)^\alpha \right\|_2 \to 0.$$

Finally, $h_n(\alpha, (T_n, I_n))$ also converges:

$$h_n(\alpha, (T_n, I_n)) \stackrel{L_2}{\to} h(\alpha, (T, U)), \tag{32}$$

where, for $\alpha > 1/2$,

$$h(\alpha, (T, U)) := \begin{cases} \dfrac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1) + T & \text{if} \quad \alpha \neq 1, \\ 2U \log U + 2(1-U)\log(1-U) + T & \text{if} \quad \alpha = 1. \end{cases} \tag{33}$$

For $\alpha = 1$, (32) is proved by the relation (see (21))

$$\frac{x_{I_n} + x_{n-1-I_n} - x_n}{nL(n)} \xrightarrow{L_2} 2U \log U + 2(1-U)\log(1-U)$$

and our assumption $T_n/t_n \to T$ in L_2. The case $\alpha \neq 1$ is established by using the asymptotic expansions (9) for $1/2 < \alpha < 1$ and Lemma 2 for $\alpha > 1$, respectively:

$$\begin{aligned}
&\frac{1}{n^\alpha L(n)}(x_{I_n} + x_{n-1-I_n} - x_n + T_n)\\
&= \frac{1}{n^\alpha L(n)}\frac{\alpha+1}{\alpha-1}(I_n^\alpha L(I_n) + (n-1-I_n)^\alpha L(n-1-I_n) - n^\alpha L(n) + T_n) + o(1)\\
&\to \frac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1) + T \quad \text{in } L_2,
\end{aligned}$$

where $o(1)$ depends on the randomness but the convergence is uniform. This establishes the stabilization of the modified recursion (30) to the limiting equation

$$Y \stackrel{d}{=} U^\alpha Y + (1-U)^\alpha Y^* + h(\alpha,(T,U)). \tag{34}$$

Note that this equation coincides for independent T, U with (11) in the case $\alpha = 1$; for $\alpha > 1/2$, $\alpha \neq 1$, (34) is a translated version of (11) in the sense that Y is a fixed-point of (34) iff $Y + (\alpha+1)/(\alpha-1)$ is a fixed point of (11). This is because, in Theorem 1, the random variable is not centered for $\alpha \neq 1$ by the exact mean so that the mean of Y_α there equals $(\alpha+1)/(\alpha-1)$, while our Y has mean zero.

The limiting equation (34) defines a map $S_{\alpha,(T,U)}$ on $\mathcal{M}_2$:

$$S_{\alpha,(T,U)} : \mathcal{M}_2 \to \mathcal{M}_2, \quad \nu \mapsto \mathcal{L}(U^\alpha Z + (1-U)^\alpha Z^* + h(\alpha,(T,U))), \tag{35}$$

where $Z, Z^*, (T,U)$ are independent, $Z \stackrel{d}{=} Z^* \stackrel{d}{=} \nu$, and $h(\alpha,(T,U))$ is given by (33).

THEOREM 3. *Let (X_n) be given by (1). Assume that*

$$E(T_n) \sim n^\alpha L(n) \quad \textit{and} \quad \left(\frac{T_n}{E(T_n)}, \frac{I_n}{n}\right) \xrightarrow{L_2} (T,U),$$

where $\alpha > 1/2$, and that T is square-integrable. Then

$$\ell_2\left(\frac{X_n - E(X_n)}{n^\alpha L(n)}, Y_{\alpha,(T,U)}\right) \to 0,$$

where $\mathcal{L}(Y_{\alpha,(T,U)})$ is the unique fixed-point in $\mathcal{M}_2(0)$ of the map $S_{\alpha,(T,U)}$ defined in (35).

Proof. First, we show that the restriction of $S_{\alpha,(T,U)}$ to $\mathcal{M}_2(0)$ is a map into $\mathcal{M}_2(0)$. Let $\nu \in \mathcal{M}_2(0)$. Then $S_{\alpha,(T,U)}(\nu)$ has a finite second moment because of independence and the same property of the coefficients. The assumption $T_n/E(T_n) \to T$ in L_2 implies that $E(T) = 1$, and, therefore, $E(h(\alpha,(T,U))) = 0$ for all $\alpha > 1/2$. This implies $E(S_{\alpha,(T,U)}(\nu)) = 0$, and thus $S_{\alpha,(T,U)}(\nu) \in \mathcal{M}_2(0)$.

By Theorem 3 in Rösler [67] or Lemma 1 in Rösler and Rüschendorf [69], $S_{\alpha,(T,U)}$ is Lipschitz continuous on $(\mathcal{M}_2(0), \ell_2)$, where the Lipschitz constant $\operatorname{lip}(S_{\alpha,(T,U)})$ satisfies

$$\operatorname{lip}(S_{\alpha,(T,U)}) \le (E(U^{2\alpha}) + E((1-U)^{2\alpha}))^{1/2}.$$

Since $\alpha > 1/2$, we have $\operatorname{lip}(S_{\alpha,(T,U)}) \le \sqrt{2/(2\alpha+1)} < 1$; thus $S_{\alpha,(T,U)}$ is a contraction on $\mathcal{M}_2(0)$. By Banach's fixed-point theorem, $S_{\alpha,(T,U)}$ has a unique fixed-point $\mathcal{L}(Y_{\alpha,(T,U)})$ in $\mathcal{M}_2(0)$.

By (30) and (34), the standardized variables $Y_n = (X_n - E(X_n))/n^\alpha L(n)$ and $Y_{\alpha,(T,U)}$ satisfy, respectively,

$$Y_n \stackrel{d}{=} A_1^{(n)} Y_{I_n} + A_2^{(n)} Y^*_{n-1-I_n} + b_n$$

and

$$Y_{\alpha,(T,U)} \stackrel{d}{=} A_1 Y_{\alpha,(T,U)} + A_2 Y^*_{\alpha,(T,U)} + b.$$

It remains to check the conditions (a)–(d).

First, by taking expectations in (30) and (34), respectively, we obtain $E(b_n) = E(b) = 0$; also, $E(b^2) < \infty$ since T is square-integrable. Thus (a) is satisfied. Condition (b) is established in (31) and (32), and condition (c) is the contraction property of $S_{\alpha,(T,U)}$. Finally, condition (d) follows from $|s(I_n)/s(n)|, |s(n-1-I_n)/s(n)| < 1$ since

$$\begin{aligned} &E(\mathbf{1}_{\{I_n \le n_1\}}(A_1^{(n)})^2) + E(\mathbf{1}_{\{n-1-I_n \le n_1\}}(A_2^{(n)})^2) \\ &\le P(I_n \le n_1) + P(n-1-I_n \le n_1) \\ &= \frac{2n_1}{n} \to 0 \end{aligned}$$

for all $n_1 \in \mathbb{N}$. We complete the proof by applying Rösler's theorem [68]. □

Note that, if $h(\alpha,(T,U)) = 0$, then the limit distribution $\mathcal{L}(Y_{\alpha,(T,U)})$ is degenerate, namely, $Y_{\alpha,(T,U)} = 0$ almost surely. In this case, more knowledge on the asymptotics of T_n is necessary, and a scaling other than $n^\alpha L(n)$ should be used. (Our limit law yields merely $\mathrm{Var}(X_n) = o(n^\alpha L(n))$.)

COROLLARY 1. *If $h(\alpha,(T,U)) \neq 0$ (see (33)), then the sequence (X_n) of Theorem 3 satisfies*

$$\mathrm{Var}(X_n) \sim \sigma^2 n^{2\alpha} L^2(n),$$

where $\sigma^2 = \sigma^2(\alpha,(T,U))$ is defined by

$$\sigma^2 = \begin{cases} \dfrac{\alpha(\alpha+1)^2 B(\alpha,\alpha) + 2(\alpha^2 - 2\alpha - 1)}{(2\alpha-1)(\alpha-1)^2} + C & \text{if } \alpha \neq 1, \\ 7 - \dfrac{2\pi^2}{3} + C & \text{if } \alpha = 1, \end{cases}$$

with $C = C(\alpha,(T,U))$ given by

$$C = \begin{cases} \dfrac{2\alpha+1}{2\alpha-1}\left(\mathrm{Var}(T) + 2\dfrac{\alpha+1}{\alpha-1} E\left[T(U^\alpha + (1-U)^\alpha)\right] - \dfrac{4}{\alpha-1}\right) & \text{if } \alpha \neq 1, \\ 3(\mathrm{Var}(T) + 4E[T(U\log U + (1-U)\log(1-U))] + 2) & \text{if } \alpha = 1. \end{cases}$$

Proof. By Theorem 3, $\mathrm{Var}(X_n) = \mathrm{Var}(n^\alpha L(n) Y_n) \sim E(Y^2_{\alpha,(T,U)}) n^{2\alpha} L^2(n)$; thus $\sigma^2 = E(Y^2_{\alpha,(T,U)})$. Since $Y_{\alpha,(T,U)}$ solves (34), we deduce, by taking squares and expectations, that

$$E(Y^2_{\alpha,(T,U)}) = \frac{2\alpha+1}{2\alpha-1} E(h^2(\alpha,(T,U))),$$

which leads to the expressions in the corollary. □

If T is independent of U, then $C = (2\alpha+1)\mathrm{Var}(T)/(2\alpha-1)$, which coincides with (23) for $\alpha = 1$. Moreover, $C = 0$ if $T = 1$, which holds particularly if the toll functions (T_n) are all deterministic.

4.3. Small toll functions. In this section, we consider small toll functions by the contraction method, assuming again that T_n and I_n may be dependent. We choose the scaling $s(n) := \sqrt{\mathrm{Var}(X_n)}$. As in the analysis by the method of moments, we consider two cases:

$$t_n = O(\sqrt{n}/(\log n)^{1/2+\epsilon}), \quad E(T_n^2) = O(t_n^2), \quad \text{and} \quad E\left(\frac{T_n}{s(n)}\right)^{2+\delta} \to 0, \tag{36}$$

where $0 < \delta \le 1$; and

$$\begin{cases} t_n \sim \sqrt{n}L(n), \quad E(T_n^2) \sim \tau_2 n L^2(n), \quad E\left(\dfrac{T_n}{s(n)}\right)^{2+\delta} \to 0, \quad \text{and} \\ E(T_n\sqrt{I_n}L(I_n)) + E(T_n\sqrt{n-1-I_n}L(n-1-I_n)) \sim \tau_2' n L^2(n). \end{cases} \tag{37}$$

In particular, if we assume (14) or (15), then (36) or (37) hold, respectively.

We first look for convergence in (27) in order to derive a limiting equation. In the case (36), we have (see (16)) $s(n)^2 \sim \sigma^2 n$; thus

$$A_1^{(n)} = \frac{s(I_n)}{s(n)} \to U^{1/2} \quad \text{in } L_{2+\delta}. \tag{38}$$

Similarly,

$$A_2^{(n)} = \frac{s(n-1-I_n)}{s(n)} \to (1-U)^{1/2} \quad \text{in } L_{2+\delta}. \tag{39}$$

For the additive term in (27), we obtain $(x_{I_n} + x_{n-1-I_n} - x_n)/s(n) \to 0$ in $L_{2+\delta}$ by the expansion $E(X_n) = \Upsilon[t]n + o(\sqrt{n}\,)$. This, together with $T_n/s(n) \to 0$, gives

$$\frac{x_{I_n} + x_{n-1-I_n} - x_n + T_n}{s(n)} \to 0 \quad \text{in } L_{2+\delta}. \tag{40}$$

The recursion (27) for $Y_n = (X_n - E(X_n))/s(n)$ and $Y_n = 0$ if $s(n) = 0$ now leads to the limiting equation

$$Y \stackrel{d}{=} U^{1/2}Y + (1-U)^{1/2}Y^*. \tag{41}$$

The conditions (38)–(40) are also satisfied in the case of (37) using the corresponding expansions for $s(n)$. Briefly, the conditions (38) and (39) are proved by $(\sum_{k\le I_n} L^2(k)/k)/(\sum_{k\le n} L^2(k)/k) \to 1$. For (40), if $\sum_k L^2(k)/k < \infty$, then $L(k) \to 0$, implying that $(x_{I_n} + x_{n-1-I_n} - x_n)/s(n) \to 0$ in $L_{2+\delta}$. If $\sum_k L^2(k)/k = \infty$, the same $L_{2+\delta}$ convergence follows from $L^2(n) = o(\sum_{k\le n} L^2(k)/k)$; see (8).

In all cases, we obtain the limiting equation (41). Therefore, we cannot follow the line as for large toll functions since (41) has no contraction properties on $(\mathcal{M}_2(0), \ell_2)$ and is not a contraction for any ℓ_r-metric. This is well known and is discussed in Rachev and Rüschendorf [62] and Rösler and Rüschendorf [69]. Thus we have to choose a metric that is $(r,+)$-ideal, where $r > 2$, and to refine the work space $\mathcal{M}_2(0)$ in order to obtain contraction properties for (41).

The situation here is similar to the size of random tries discussed in Rachev and Rüschendorf [62]. We obtain weak convergence of (Y_n) to a normal distribution by applying similar arguments; see also Rösler and Rüschendorf [69].

We define, for $r = m + 1/p$ with $m \in \mathbb{N}$ and $p \in [1, \infty)$,

$$\mathcal{F}_r := \{f \in C^{m+1} : \|f^{(m+1)}\|_q \le 1\},$$

where $1/p + 1/q = 1$ and $f^{(m+1)}$ denotes the $(m+1)$st derivative of the function $f : \mathbb{R} \to \mathbb{R}$. Then we will use the metric

$$\mu_r(X, Y) := \sup_{f \in \mathcal{F}_r} |E[f(X) - f(Y)]|,$$

which was introduced and studied in Maejima and Rachev [46]; see also Rachev and Rüschendorf [61].

We briefly state the properties of μ_r, which are used subsequently. The metric μ_r is $(r, +)$-ideal; i.e., $\mu_r(cX, cY) = |c|^r \mu_r(X, Y)$ for $c \neq 0$ and $\mu_r(X + Z, Y + Z) \le \mu_r(X, Y)$ if Z is independent of X, Y. An upper estimate for μ_r in Zolotarev's metric ζ_r and corresponding properties for the metric ζ_r (see Zolotarev [76]) imply that $\mu_r(X, Y) < \infty$ if $E(X^j) = E(Y^j)$ for all $j = 1, \dots, m$ and $E(|X|^r), E(|Y|^r) < \infty$. Convergence in μ_r implies convergence in distribution since a lower estimate in Levy's metric L is valid: $(L(X, Y))^{r+1} \le C(r)\mu_r(X, Y)$ for some constant $C(r) < \infty$. We will also use the fact that convergence in ℓ_r implies convergence in μ_r. This follows from the upper estimate $\mu_r(X, Y) \le C'(r)\kappa_r(X, Y)$ with some constant $C'(r) < \infty$ and the difference pseudomoment κ_r and the fact that κ_r and ℓ_r are topologically equivalent (see Rachev [60, p. 301]).

The following proof of asymptotic normality is based on the approach used in Rachev and Rüschendorf [62] mentioned above. The differences here are that we derive convergence in $\mu_{2+\delta}$ rather than only weak convergence and that the estimate of the additive term $h_n(T_n, I_n)$ is simplified. These improvements are due to the fact that more information on the moments is known in our case.

THEOREM 4. *Let (X_n) be given by* (1). *If T_n satisfies either* (36) *or* (37), *then, for $0 < \delta \le 1$,*

$$\mu_{2+\delta}\left(\frac{X_n - E(X_n)}{\sqrt{\mathrm{Var}(X_n)}}, N(0, 1)\right) \to 0.$$

Proof. Let $r := 2 + \delta$, and denote $Y_n := (X_n - x_n)/s(n)$ if $s(n) = \sqrt{\mathrm{Var}(X_n)} > 0$ and $Y_n := 0$ otherwise. The key idea of the proof is to introduce a mixed quantity that combines the structure of the modified recursion with the normal distribution; see [62] and [69, section 6]. We denote by N, N^* two independent standard normal random variables that are also independent of all other quantities.

Then we define the distributions of our mixtures M_n by $M_n := 0$ if $s(n) = 0$ and, for $s(n) > 0$, by

$$\begin{aligned} M_n &\stackrel{d}{:=} \frac{s(I_n)}{s(n)} N + \frac{s(n-1-I_n)}{s(n)} N^* + h_n(T_n, I_n) \\ &\stackrel{d}{=} \left[\left(\frac{s(I_n)}{s(n)}\right)^2 + \left(\frac{s(n-1-I_n)}{s(n)}\right)^2\right]^{1/2} N + h_n(T_n, I_n), \end{aligned} \tag{42}$$

with $h_n(T_n, I_n)$ given in (28). A comparison with (27) shows that $E(M_n) = 0$, $E(M_n^2) = 1$, and $E|M_n|^r < \infty$ if $s(n) > 0$; thus μ_r-distances between Y_n, M_n,

and $N(0,1)$ are finite. We have $s(n) > 0$ for all n sufficiently large, say, for all $n \geq n_0$. For these n, we estimate

$$\mu_r(Y_n, N(0,1)) \leq \mu_r(Y_n, M_n) + \mu_r(M_n, N(0,1)).$$

By (38) and (39), the factor between the brackets in (42) converges to 1 in L_r; this, together with the L_r-convergence of $h_n(T_n, I_n)$ to 0, yields $\ell_r(M_n, N(0,1)) \to 0$ and, therefore, $\mu_r(M_n, N(0,1)) \to 0$. Here we used the estimates (38)–(40).

For the estimate of the term $\mu_r(Y_n, M_n)$, we have, for $n \geq n_0$,

$$Y_n \stackrel{d}{=} \frac{s(I_n)}{s(n)} Y_{I_n} + \frac{s(n-1-I_n)}{s(n)} Y^*_{I_n} + h_n(T_n, I_n).$$

We denote by λ_n the joint distribution of (T_n, I_n). By the $(r,+)$-ideality of μ_r, we have, for $n \geq n_0$,

$$\begin{aligned}
\mu_r(Y_n, M_n) &= \sup_{f \in \mathcal{F}_r} |E(f(Y_n) - f(M_n))| \\
&= \sup_{f \in \mathcal{F}_r} \left| \int E\left[f\left(\frac{s(k)}{s(n)} Y_k + \frac{s(n-1-k)}{s(n)} Y^*_{n-1-k} + h_n(t,k) \right) \right.\right. \\
&\qquad \left.\left. - f\left(\frac{s(k)}{s(n)} N + \frac{s(n-1-k)}{s(n)} N^* + h_n(t,k) \right) \right] \mathrm{d}\lambda_n(t,k) \right| \\
&\leq \int \mu_r\left(\frac{s(k)}{s(n)} Y_k + \frac{s(n-1-k)}{s(n)} Y^*_{n-1-k} + h_n(t,k), \right. \\
&\qquad \left. \frac{s(k)}{s(n)} N + \frac{s(n-1-k)}{s(n)} N^* + h_n(t,k) \right) \mathrm{d}\lambda_n(t,k) \\
&\leq \frac{1}{n} \sum_{k=0}^{n-1} \left(\mu_r\left(\frac{s(k)}{s(n)} Y_k, \frac{s(k)}{s(n)} N \right) \right. \\
&\qquad \left. + \mu_r\left(\frac{s(n-1-k)}{s(n)} Y^*_{n-1-k}, \frac{s(n-1-k)}{s(n)} N^* \right) \right) \\
&\leq \frac{2}{n} \sum_{k=0}^{n-1} \left(\frac{s(k)}{s(n)} \right)^r \mu_r(Y_k, N).
\end{aligned} \tag{43}$$

Thus we obtain the reduction inequality

$$\mu_r(Y_n, N(0,1)) \leq \frac{2}{n} \sum_{k=0}^{n-1} \left(\frac{s(k)}{s(n)} \right)^r \mu_r(Y_k, N(0,1)) + o(1).$$

By (38) and (39),

$$\frac{2}{n} \sum_{k=0}^{n-1} \left(\frac{s(k)}{s(n)} \right)^r = 2E\left(\frac{s(I_n)}{s(n)} \right)^r \to 2E(U^{r/2}) = \frac{2}{r/2+1} < 1.$$

From this and the reduction inequality, we deduce by a bootstrapping argument (see Rösler [66, p. 94] or Rachev and Rüschendorf [62, p. 786]) that $\mu_r(Y_n, N(0,1)) \to 0$. (First prove that $\mu_r(Y_n, N(0,1))$ remains bounded; then refine the approximation.) □

5. Continuous change of limits. The limit distributions $\mathcal{L}(Y_{\alpha,(T,U)})$ in Theorem 3 are continuous in the parameters $(\alpha,(T,U))$ as $\alpha > 1/2$. This continuity still holds as $\alpha \downarrow 1/2$ in the case of deterministic toll functions and in the random case under appropriate assumptions.

THEOREM 5. *Let $\alpha \to \beta > 1/2$ and $T = T(\alpha) \to V$ in L_2 for a square-integrable V. Then*

$$\ell_2(Y_{\alpha,(T,U)}, Y_{\beta,(V,U)}) \to 0.$$

Let $\alpha \downarrow 1/2$ and $T = T(\alpha)$ satisfy $\|T\|_{2+\delta} = o(\sigma(\alpha,(T,U)))$ as $\alpha \downarrow 1/2$ with $0 < \delta \le 1$. If T is independent of U, then

$$\mu_{2+\delta}\left(\frac{Y_{\alpha,(T,U)}}{\sigma(\alpha,(T,U))}, N(0,1)\right) \to 0.$$

The property still holds if T, U are dependent, provided that (i) *$h(\alpha,(T,U)) \neq 0$ for h given in* (33) *and* (ii) *$\sigma(\alpha,(T,U))$ in Corollary* 1 *is properly divergent.*

Proof (sketch). Consider the special case $\beta = 1$. For $\alpha > 1/2$, we have, by definition,

$$Y_{\alpha,(T,U)} \stackrel{d}{=} U^\alpha Y_{\alpha,(T,U)} + (1-U)^\alpha Y^*_{\alpha,(T,U)} + \frac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1) + T,$$
$$Y_{1,(V,U)} \stackrel{d}{=} U Y_{1,(V,U)} + (1-U) Y^*_{1,(V,U)} + 2U\log U + 2(1-U)\log(1-U) + V,$$

where $(Y_{\alpha,(T,U)}, Y_{1,(V,U)})$, $(Y^*_{\alpha,(T,U)}, Y^*_{1,(V,U)})$, (T,U,V) are independent, and the variates $(Y_{\alpha,(T,U)}, Y_{1,(V,U)})$, $(Y^*_{\alpha,(T,U)}, Y^*_{1,(V,U)})$ are optimal couplings of $\mathcal{L}(Y_{\alpha,(T,U)})$ and $\mathcal{L}(Y_{1,(V,U)})$. To match these two fixed-point equations, we use the Taylor expansion

$$x^\alpha = x + (\alpha-1)x\log x + \int_1^\alpha (\alpha - y)(x^{y-1} + x^y(\log x)^2)\,\mathrm{d}y, \tag{44}$$

where $x \in (0,1)$ and $\alpha > 0$. Using the representations of $Y_{\alpha,(T,U)}, Y_{1,(V,U)}$ given in the coupled fixed-point equations in the estimate $\ell_2(Y_{\alpha,(T,U)}, Y_{1,(V,U)}) \le \|Y_{\alpha,(T,U)} - Y_{1,(V,U)}\|_2$, we obtain, after tedious calculations, that

$$\ell_2(Y_{\alpha,(T,U)}, Y_{1,(V,U)}) \ll \max\{|\alpha-1|, \sqrt{|\alpha-1|\|T-V\|_2}, \|T-V\|_2\} \tag{45}$$

as $\alpha \to 1$ and $T \to V$ in L_2. In particular, we used the expansion

$$B(\alpha,\alpha) = 1 - 2(\alpha-1) + (4-\pi^2/6)(\alpha-1)^2 + O((\alpha-1)^3) \tag{46}$$

to derive $\sigma(\alpha,(T,U)) \to \sigma(1,(V,U))$ as $\alpha \to 1$ and $T \to V$ in L_2. This implies the assertion for $\beta = 1$. The general case $\beta > 1/2$ can be treated by the same approach and is indeed simpler since the expansions (44) and (46) are not needed.

For the second part, we denote $r := 2 + \delta$ and $Z_{\alpha,(T,U)} := Y_{\alpha,(T,U)}/\sigma(\alpha,(T,U))$. These rescaled quantities satisfy the fixed-point equation

$$Z_{\alpha,(T,U)} \stackrel{d}{=} U^\alpha Z_{\alpha,(T,U)} + (1-U)^\alpha Z^*_{\alpha,(T,U)} + \frac{1}{\sigma(\alpha,(T,U))}\left[\frac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1) + T\right],$$

where $Z_{\alpha,(T,U)}$, $Z^*_{\alpha,(T,U)}$, (U,T) are independent and $Z_{\alpha,(T,U)} \stackrel{d}{=} Z^*_{\alpha,(T,U)}$. We denote by N, N^* two independent standard normal distributed random variables independent of the other quantities. Then

$$N(0,1) \stackrel{d}{=} U^{1/2}N + (1-U)^{1/2}N^*.$$

Moreover, we define, similarly to (42), the mixtures

$$M_{\alpha,(T,U)} \stackrel{d}{=} U^\alpha N + (1-U)^\alpha N^* + \frac{1}{\sigma(\alpha,(T,U))}\left[\frac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1) + T\right].$$

Then $E(M_{\alpha,(T,U)}) = 0$, $E(M^2_{\alpha,(T,U)}) = 1$, and $E|M_{\alpha,(T,U)}|^r < \infty$; thus the μ_r distances between $Z_{\alpha,(T,U)}$, $N(0,1)$, and $M_{\alpha,(T,U)}$ are finite. It follows that

$$\mu_r(Z_{\alpha,(T,U)}, N(0,1)) \le \mu_r(Z_{\alpha,(T,U)}, M_{\alpha,(T,U)}) + \mu_r(M_{\alpha,(T,U)}, N(0,1)).$$

A calculation similar to (43) implies, for $\alpha > 1/2$, that

$$\begin{aligned}\mu_r(Z_{\alpha,(T,U)}, M_{\alpha,(T,U)}) &\le 2E(U^{r\alpha})\mu_r(Z_{\alpha,(T,U)}, N(0,1)) \\ &\le \frac{2}{1+r/2}\mu_r(Z_{\alpha,(T,U)}, N(0,1)).\end{aligned}$$

Note that the assumptions (i) and (ii) for the dependent case are also satisfied in the case when T and U are independent (see Corollary 1). The asymptotic normality for dependent and independent cases can be derived under conditions (i) and (ii) by proving $\mu_r(M_{\alpha,(T,U)}, N(0,1)) = o(1)$ as $\alpha \downarrow 1/2$. This follows from the convergence in ℓ_r, which is obtained using the fixed-point equations for $N(0,1)$, $M_{\alpha,(T,U)}$, $\|T\|_r = o(\sigma(\alpha,(T,U)))$ and that $\sigma(\alpha,(T,U))$ is properly divergent, giving

$$\begin{aligned}&\ell_r(M_{\alpha,(T,U)}, N(0,1)) \\ &\le 2\,\|U^{1/2} - U^\alpha\|_r \|N\|_r \\ &\quad + \frac{1}{\sigma(\alpha,(T,U))}\left[\left\|\frac{\alpha+1}{\alpha-1}(U^\alpha + (1-U)^\alpha - 1)\right\|_r + \|T\|_r\right],\end{aligned}$$

which tends to zero as $\alpha \downarrow 1/2$ under our assumptions. It follows that

$$\mu_r(Z_{\alpha,(T,U)}, N(0,1)) \le \frac{2}{1+r/2}\mu_r(Z_{\alpha,(T,U)}, N(0,1)) + o(1);$$

thus $2/(1+r/2) < 1$ implies $\mu_r(Z_{\alpha,(T,U)}, N(0,1)) \to 0$. □

Note that, in the case $\alpha \downarrow 1/2$ and $T = 1$, which holds especially for deterministic toll functions, all conditions of the theorem are satisfied.

We may endow $(1/2,\infty) \times L_2$ with the metric $d((\alpha,T),(\beta,V)) := |\alpha - \beta| + \|T - V\|_2$. Then, for fixed U, the map $Y : (1/2,\infty) \times L_2 \to \mathcal{M}_2(0)$, $(\alpha,T) \mapsto \mathcal{L}(Y_{\alpha,(T,U)})$ is locally Lipschitz continuous with respect to d and ℓ_2. This follows by making all the constants explicit in the estimate (45) and in the corresponding one for general $\beta > 1/2$.

6. Examples. In this section, we discuss many examples, most of which are new.

The number of exchanges of quicksort. The number of exchanges used by quicksort satisfies (1) with T_n dependent on I_n. While Theorem 1 does not apply, its proof does. The starting point is the recurrence $P_0(y) = 1$ and, for $n \geq 1$,

$$P_n(y) = \frac{1}{n} \sum_{0 \leq k < n} P_k(y) P_{n-1-k}(y) \sum_{0 \leq j \leq \min\{k, n-1-k\}} \pi_{n,k,j} e^{jy},$$

where $\pi_{n,k,j}$ denotes the probability that there are exactly j exchanges when the rank of the pivot element is $k+1$ so that (see Sedgewick [71, p. 55])

$$\pi_{n,k,j} = \frac{\binom{k}{j}\binom{n-1-k}{j}}{\binom{n-1}{k}}. \tag{47}$$

Note that the exact number of exchanges used depends on implementation details, and we count only the essential random part.

Using the identity

$$\sum_{j \geq 1} \pi_{n,k,j} j(j-1)\cdots(j-v+1) = \frac{(n-v-1)!k!(n-1-k)!}{(n-1)!(k-v)!(n-k-1-v)!}, \tag{48}$$

valid for $v = 0, 1, 2, \ldots$ and (7), we easily obtain

$$E(X_n) = \frac{n+1}{3} H_n - \frac{7}{9} n + \frac{1}{18} \qquad (n \geq 2).$$

For higher moments, we proceed as in section 3 ($\alpha = 1$) by defining $\Pi_n(y) := P_n(y) e^{-x_n y}$ and $\Pi_{n,m} := \Pi_n^{(m)}(0)$. Then, by the same approach, we deduce that

$$\Pi_{n,m} \sim g_m n^m \qquad (n \geq 2),$$

where $g_0 = 1$, $g_1 = 0$, and, for $n \geq 2$,

$$g_m = \sum_{a+b+c=m} \binom{m}{a, b, c} g_a g_b \times \int_0^1 x^a (1-x)^b \left(\frac{x}{3} \log x + \frac{1-x}{3} \log(1-x) + x(1-x) \right)^c \, \mathrm{d}x.$$

Thus

$$\frac{X_n - x_n}{n} \xrightarrow{d} Y$$

as well as convergence of all moments, where

$$Y \stackrel{d}{=} UY + (1-U)Y^* + \frac{U}{3} \log U + \frac{1-U}{3} \log(1-U) + U(1-U),$$

with $Y \stackrel{d}{=} Y^*$ and Y, Y^*, U independent.

On the other hand, Theorem 3 applies by establishing

$$\frac{T_n}{n/6} \xrightarrow{L_2} 6U(1-U).$$

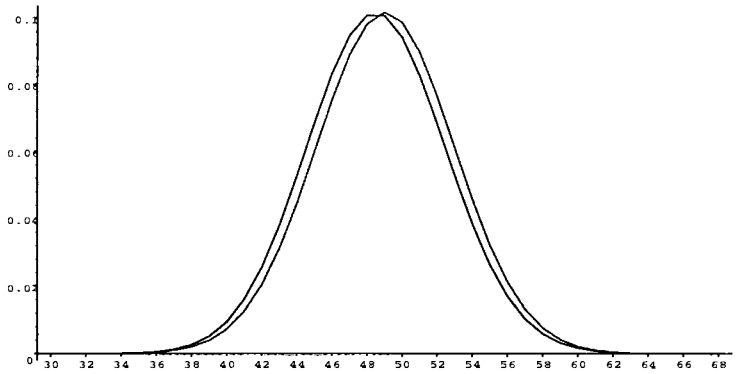


FIG. 2. *The histogram of* $P(X_{60} = k)$ *for* k *from* 30 *to* 68 *and the corresponding normal curve* $e^{-(k-E(X_{60})-1/2)^2/(2\mathrm{Var}(X_{60}))}/\sqrt{2\pi \mathrm{Var}(X_{60})}$. *We shifted the normal density by* 1/2*; otherwise, the two curves will be almost indistinguishable.*

This follows from (48).

In particular, by the recurrence of g_m or by Corollary 1, we obtain $\mathrm{Var}(X_n) \sim (\frac{11}{60} - \frac{\pi^2}{54})n^2$.

Note that, by (48),

$$\begin{aligned} E(T_n^k) &\sim E(T_n(T_n-1)\cdots(T_n-k+1)) \\ &= \frac{k!k!(n-k-1)!}{(2k+1)!(n-2k-1)!} \\ &\sim \frac{k!k!}{(2k+1)!}\, n^k \end{aligned}$$

for $k \geq 1$. Thus T_n/n has in the limit a beta distribution:

$$P\left(\frac{T_n}{n} < x\right) \to 1 - \sqrt{1-4x} \qquad (0 < x < 1/4).$$

Unlike the number of comparisons, which has quadratic worst-case behavior, the number of exchanges is at most of order $n \log n$. Also, it is interesting to note that the histograms of $P(X_n = i)$ are very close to normal curves for n small; see Figure 2. An explanation of this phenomenon is that the leading constant of the variance (as well as g_3) is very small: $\frac{11}{60} - \frac{\pi^2}{54} \approx 0.00056288$. The "nonnormality character" of Y will emerge for large enough n.

The limit law (49) is different from that of the number of comparisons (2); however, the limit distributions are related by their defining fixed-point equations. Indeed, the correlation of the number of comparisons and the number of exchanges is asymptotic to

$$\frac{\sqrt{5}(39-4\pi^2)}{2\sqrt{(21-2\pi^2)(99-10\pi^2)}} \approx -0.864042\ldots.$$

This can be proved by the bivariate limit law of both variates that can be derived by a multivariate extension of the contraction method (see Neininger [53] for details).

Thus *the number of comparisons and the number of exchanges are highly negatively correlated*. Intuitively, when the shape of the corresponding binary search tree is very skewed, few key exchanges are needed; on the other hand, the number of exchanges reaches its maximum when the pivot element is around $n/2$ (see (47)). Roughly, the more "balanced" the permutation, the more exchanges are needed. The situation here is more or less the same when one uses the median-of-$(2t+1)$ quicksort: while the number of comparisons decreases with t, the number of exchanges increases. We might say that we trade off the number of exchanges for the number of comparisons.

Note that the same limit law (2) for $T_n = n + O(1)$ persists for $T_n = n + \omega(n)$, where $\omega(n) = o(n)$ and $\sum_n \omega(n)/n^2 < \infty$; this reflects the "robustness" of the limit laws.

Paged trees. Fix a page (or bucket) size $b \geq 1$. Cut all nodes with subtree sizes $\leq b$. The resulting tree is called the b-index of the tree; see Flajolet, Gourdon, and Martínez [28] and Mahmoud [48]. What is the size of a random b-index? And what is the total path length? Obviously, both random variables satisfy (1) (with different initial conditions). The asymptotic normality of the size was established for fixed b by Flajolet, Gourdon, and Martínez [28] with mean equal to $2(n+1)/(b+2)-1$. The variance is equal to (the expression given in [28] being wrong)

$$2\left(4H_{2b+2} - 4H_{b+1} - \frac{(b+1)(5b+2)}{(2b+3)(b+2)}\right)\frac{n+1}{b+2} \qquad (n \geq 2b+2).$$

Indeed, we can prove that the asymptotic normality holds for $2 \leq b = o(n)$. This does not follow directly from our results but is easily amended by truncating the first b terms in our exact and asymptotic expressions (6) and by applying the same arguments.

If we vary b such that $2 \leq b = o(n)$, then the path length of the b-index gives an interesting example with mean of order n/b, which varies from linear to any function tending to infinity. Thus the limit laws change from nonnormal to normal when b increases.

This variation of path length suggests in turn a variation of quicksort: stop subfiles of size less than or equal to b, where b can vary with n. We can show that the limit law of the total number of comparisons used in the quicksort partitioning stages does not change as long as $b = o(n)$. This images another "robustness" of the limit laws.

Leaves and patterns in binary search trees. Our Theorem 2 can be applied to the number of times a given pattern appears in a random binary search tree; see Devroye [18] and Flajolet, Gourdon, and Martínez [28]. The number of times a subtree of size k appears is also asymptotically normally distributed; see Aldous [1] and Devroye [20]. By the correspondence between increasing trees (or binary recursive trees) and permutations, some patterns on trees like the number of leaves also lead to well-known distributions in random permutations; see Bergeron, Flajolet, and Salvy [6] and Flajolet, Gourdon, and Martínez [28].

Analysis of tree traversal algorithms. Binary search trees can be implemented in several different ways: two-pointers, threaded with or without flag, triply linked (with a pointer to parent), etc.; and the nodes can be traversed in different orders: inorder, preorder, postorder, breadth-first, depth-first, etc.; see [2, 10, 11, 12, 13, 23, 24, 32, 51, 65] and [31]. The analysis of the cost of these algorithms then reduces to the calculation of certain parameters on trees such as the number of nodes with null (or nonnull) left (or right) branches, the number of nodes with both nonnull left and right branches, and the number of nodes that are a left child and whose right branch

is not empty. All these quantities can be systematically analyzed by applying our results; see Brinck and Foo [11] and Brinck [10] for analysis of the mean of some cost measures.

For example, the major cost (number of pointer operations) needed to traverse a threaded binary search tree in preorder and in inorder is essentially given by (neglecting minor parameters)

$$X_n \stackrel{d}{=} X_{I_n} + X^*_{n-1-I_n} + T_{I_n},$$

where

$$Q_n(y) := E(e^{T_n y}) = \prod_{1\le k\le n} \frac{k-1+e^y}{k} \qquad (n \ge 1), \tag{49}$$

essentially the Stirling numbers of the first kind (enumerating the number of records in independent and identically distributed sequences, the number of cycles in random permutations, etc.). The distribution of X_n is asymptotically normal. Likewise, the moment generating function $P_n(y)$ of the cost for postorder traversal satisfies

$$P_n(y) = \frac{e^y}{n} \sum_{0\le k\le n-2} P_k(y) P_{n-1-k}(y) Q_k^2(y) V_k(y) + \frac{e^y}{n} P_{n-1} Q_{n-1}^2(y),$$

where $Q_n(y)$ is defined as in (49) and $V_n(y)$ denotes the moment generating function for the depth of the first node in postorder; see (53). The mean was derived by Brinck [10]. Indeed, the exact forms of these generating functions are immaterial because our results are strong enough to prove the asymptotic normality of the cost within a large range of variation for the toll function; see also section 7 for the asymptotic normality of the depth of the first node in postorder.

Secondary parameters of quicksort. If we always sort smaller files first, then the number of stack pushes and pops used to sort a random input satisfies $P_n(y) = 1$ for $n \le 4$ and

$$P_n(y) = \frac{e^y}{n} \sum_{0\le k<n} P_k(y) P_{n-1-k}(y) + \frac{2}{n}(1-e^y)\left(P_{n-1}(y) + P_{n-2}(y)\right) \qquad (n \ge 5).$$

Our results apply, and the number of stack pushes is asymptotically normally distributed. If we stop sorting subfiles of sizes less than a certain given value and then use a final insertionsort to complete the sorting, then the number of comparisons and exchanges used by the insertionsort is again normal in the limit. For more information on analysis of quicksort, see Sedgewick [71], Hennequin [35], and Chern and Hwang [14].

Sorting on a broadcast communication model. The model consists of n processors sharing a common channel for communications, allowing one processor to broadcast at each time epoch. To each processor a certain number is attached. (The numbers are distinct.) The sorting problem is to order these numbers in increasing order. The algorithm proposed in Shiau and Yang [72] is as follows. Select first a loser (a preferable term being "a leader") by the coin-flipping procedure in Prodinger [59]. Split the processors into two subsets containing, respectively, smaller and larger numbers; then sort recursively by the same approach; see Shiau and Yang [72] for details. The number of rounds of coin-tossings (in order to resolve the conflict for using the channel)

satisfies (1) with T_n given by ($Q_n(y) := E(e^{T_n y})$)

$$Q_n(y) = \frac{e^y}{2^n} \sum_{1 \le k \le n} \binom{n}{k} Q_k(y) + \frac{e^y}{2^n} Q_n(y) \qquad (n \ge 2),$$

with $Q_1(y) = 1$. The mean of X_n is studied by Grabner and Prodinger [33]. By the results of Fill, Mahmoud, and Szpankowski [27], our results apply, and X_n is asymptotically normal.

In-situ permutation algorithm. The problem in question is as follows: Given a sequence of numbers $\{a_1, \dots, a_n\}$ and a permutation $\{\pi_1, \dots \pi_n\}$, output $\{a_{\pi_1}, \dots, a_{\pi_n}\}$ using at most $O(1)$ space. An algorithm was given by MacLeod [45] and analyzed by Knuth [43]. Kirschenhofer, Prodinger, and Tichy [42] showed that the major cost X_n of the algorithm satisfies the quicksort recurrence (1) with $T_n = I_n$. They extended Knuth's analysis of the first two moments by computing the asymptotics of all moments (noncentered).

Theorem 3 applies, and we obtain

$$\frac{X_n - E(X_n)}{n} \xrightarrow{d} Y, \tag{50}$$

where $Y \stackrel{d}{=} UY + (1-U)Y^* + U \log U + (1-U)\log(1-U) + U$. Note that

$$E(X_n) \sim n \log n, \qquad \operatorname{Var}(X_n) \sim \sigma^2(1, (2U, U))\, n^2 = \left(2 - \frac{\pi^2}{6}\right) n^2.$$

We can indeed prove convergence of all moments using the same approach in section 3 starting from $P_0(y) = 1$ and

$$P_n(y) = \frac{1}{n} \sum_{0 \le k < n} e^{ky} P_k(y) P_{n-1-k}(y) \qquad (n \ge 1). \tag{51}$$

Note that X_n can also be viewed as the *left path length* of random binary search trees (by counting only left branches). In general, one may consider *weighted path length* by assigning weight α to each left branch and β to each right branch in a random binary search tree; our tools apply.

Recursive trees. Interestingly, the limit distribution (50) also appears as the limit distribution of the total path length of random recursive trees; see Dobrow and Fill [22] and Mahmoud [47]. This can be explained in two ways. First, by a well-known transformation from multiway trees to binary trees (see Cormen, Leiserson, and Rivest [16]), we can actually prove a bijection between the total path length of a recursive tree of n nodes and the left path length of a random binary search tree of $n-1$ nodes, the latter having the same distribution as the major cost of the in-situ permutation algorithm.

Second, the underlying recurrence for total path length of recursive trees is almost identical to (51)

$$X_n \stackrel{d}{=} X_{J_n} + X_{n-J_n} + J_n,$$

where J_n is uniformly distributed over $\{1, 2, \dots, n-1\}$.

This connection makes it possible to derive the limit laws of other parameters on recursive trees by our approaches (up to minor modifications) like the number of

leaves, the number of nodes with a specified degree, etc.; see Smythe and Mahmoud [73] for a survey of recursive trees. Note that the number of leaves satisfies $P_0(y) = 1$, $P_1(y) = e^y$, and

$$P_n(y) = \frac{1}{n-1} \sum_{1\le k\le n-2} P_k(y)P_{n-1-k}(y) + \frac{P_{n-1}(y)}{n-1} \qquad (n \ge 2),$$

the underlying distribution being essentially the Eulerian numbers; see Bergeron, Flajolet, and Salvy [6].

Superlinear toll functions. The Wiener index of a graph is defined as the sum of the distances between all pairs of nodes. This index plays an important role in connection with physico-chemical properties (like boiling point, heat of information, crystal defects) of chemical structures; see Gutman, Klavzar, and Mohar [34] and Trinajstić [74]. The Wiener index of a random binary search tree satisfies, neglecting the independence assumptions, (1) with

$$T_n = 2I_n(n-1-I_n) + Z_n + I_n Z^*_{n-1-I_n} + (n-1-I_n)Z'_{I_n},$$

where Z_n denotes the total path length, which satisfies (1) with $T_n = n-1$. The mean is easily seen to be

$$E(X_n) = 2n^2H_n - 6n^2 + 8nH_n - 10n + 6H_n \qquad (n \ge 1).$$

However, our results fail since Z_n and X_n are not independent. The variance satisfies $\mathrm{Var}(X_n) \sim (\frac{20}{3} - \frac{2}{3}\pi^2)n^4$, and the characterization of the limit law of X_n necessitates a multivariate extension of our approach; see Neininger [55] for details.

Other examples. For other examples of the quicksort type leading to an asymptotically normal distribution, see Fill [25], Hofri and Shachnai [38], Panholzer and Prodinger [57], and Chern, Hwang, and Tsai [15].

7. One-sided quicksort recurrence. In this section, we briefly discuss the recurrence (3). Assume that T_n is independent of I_n. Then the moment generating function of X_n satisfies $P_0(y) = 1$ and, for $n \ge 1$,

$$P_n(y) = \frac{Q_n(y)}{n} \sum_{0\le k<n} P_k(y),$$

which, by considering the difference $nP_n(y) - (n-1)P_{n-1}(y)Q_n(y)/Q_{n-1}(y)$, can be easily solved, giving

$$P_n(y) = Q_n(y) \prod_{1\le k<n} \frac{k + Q_k(y)}{k+1} \qquad (n \ge 1).$$

Thus $X_n - T_n$ is the sum of independent mixed random variables. The asymptotic transfer from the toll function to the total cost in this case is much simpler.

LEMMA 5. *Define* $a_0 = 0$ *and, for* $n \ge 1$,

$$a_n = b_n + \frac{1}{n} \sum_{0\le k<n} a_k. \tag{52}$$

Then

$$a_n = b_n + \sum_{1\le k<n} \frac{b_k}{k+1} \qquad (n \ge 1).$$

Proof. The proof is omitted. □

LEMMA 6 (asymptotic transfer). *Assume a_n satisfies* (52). *If $b_n \sim n^\alpha L(n)$, where $L(n)$ is slowly varying, then*

$$a_n \sim \begin{cases} \displaystyle\sum_{1\le k<n} \frac{L(k)}{k+1} & \text{if } \alpha = 0, \\ \displaystyle\frac{\alpha+1}{\alpha}\, n^\alpha L(n) & \text{if } \alpha > 0. \end{cases}$$

Proof. The proof is omitted. □

For the limit laws, we have roughly

$$\begin{aligned} \frac{P_n(y)}{Q_n(y)} &= \prod_{1\le k<n} \left(1 + \frac{Q_k(y)-1}{k+1}\right) \\ &\approx \exp\left(\sum_{1\le k<n} \frac{Q_k(y)-1}{k+1}\right) \\ &\approx \exp\left(y \sum_{1\le k<n} \frac{Q_k'(0)}{k+1} + \frac{y^2}{2} \sum_{1\le k<n} \frac{Q_k''(0)}{k+1} + O\left(|y|^3 \sum_{1\le k<n} \frac{|Q_k'''(0)|}{k+1}\right)\right). \end{aligned}$$

Thus, for small toll functions, if

$$\sum_{1\le k<n} \frac{Q_k''(0)}{k+1} \to \infty$$

and

$$\left(\sum_{1\le k<n} \frac{Q_k''(0)}{k+1}\right)^{-3/2} \sum_{1\le k<n} \frac{|Q_k'''(0)|}{k+1} \to 0,$$

then X_n is asymptotically normally distributed.

On the other hand, for larger toll functions, if $T_n/t_n \xrightarrow{d} T$, then roughly

$$\begin{aligned} \frac{P_n(y)}{Q_n(y)} &\approx \exp\left(\sum_{1\le k\le n} \frac{Q_k(y)-1}{k+1}\right) \\ &\approx \exp\left(\int_0^x \frac{Q(v)-1}{v}\, \mathrm{d}v\right), \end{aligned}$$

where $Q(y)$ denotes the moment generating function of T.

Instead of making these heuristics rigorous, we state a simpler result, describing mainly the phase change from normal to nonnormal laws.

THEOREM 6. *Let X_n satisfy* (3), *where T_n is independent of I_n. Assume that*

$$E(T_n) \sim n^\alpha L(n) \quad \text{and} \quad E\left(\frac{T_n}{t_n}\right)^m \to \tau_m \qquad (m \ge 1),$$

where $\alpha > 0$, and that $Q(z) := \sum_{m\ge 0} \tau_m z^m/m!$ has a nonzero radius of convergence. Then

$$\frac{X_n}{n^\alpha L(n)} \xrightarrow{d} X,$$

with convergence of all moments, where $G(z) := E(e^{zX})$ *satisfies*

$$G(z) = \int_0^1 \exp\left(\frac{1}{\alpha}\int_0^{w^\alpha z} \frac{Q(v)-1}{v}\,\mathrm{d}v\right)\mathrm{d}w$$

for sufficiently small z*. On the other hand, if*

$$t_n \sim L(n) \quad \textit{and} \quad E(|T_n|^m) = O(t_n^m) \qquad (m = 2, 3)$$

and

$$s^2(n) := \sum_{1\le k<n} \frac{Q_k''(0)}{k+1} \to \infty,$$

then

$$\frac{X_n - \sum_{1\le k<n} Q_k'(0)/(k+1)}{s(n)} \xrightarrow{d} N(0,1).$$

Proof (sketch). The proof of the asymptotic normality follows from the above argument using moment generating functions and Curtiss's continuity theorem. For large toll functions, we use the method of moments as above by proving

$$P_n^{(m)}(0) \sim g_m n^{m\alpha} L^m(n),$$

where $g_0 = 1$ and, for $m \ge 1$,

$$g_m = \sum_{0\le j\le m} \binom{m}{j} \frac{g_j}{j\alpha+1}\,\tau_{m-j}.$$

The required result follows from the same arguments we used for (1). □

When $\alpha > 0$, the contraction method gives another access to the limit law, where T_n may depend on I_n.

THEOREM 7. *Let* (X_n) *be given by* (3)*. Assume that*

$$E(T_n) \sim n^\alpha L(n) \quad \textit{and} \quad \left(\frac{T_n}{E(T_n)}, \frac{I_n}{n}\right) \xrightarrow{L_2} (T, U),$$

where $\alpha > 0$ *and* T *is square-integrable. Then*

$$\ell_2\left(\frac{X_n}{n^\alpha L(n)}, X_{\alpha,(T,U)}\right) \to 0,$$

where $\mathcal{L}(X_{\alpha,(T,U)})$ *is the unique fixed-point of the map*

$$S_{\alpha,(T,U)} : \mathcal{M}_2 \to \mathcal{M}_2, \quad \nu \mapsto \mathcal{L}(U^\alpha Z + T),$$

with $Z, (T, U)$ *independent and* $\mathcal{L}(Z) = \nu$*.*

Proof. The proof is omitted. □

If $T \ne (\alpha+1)(1-U^\alpha)/\alpha$, then

$$\mathrm{Var}(X_n) \sim \sigma^2 n^{2\alpha} L^2(n),$$

where $\sigma = \sigma(\alpha, (T, U))$ is defined by

$$\sigma^2 = \frac{1}{2\alpha} + \frac{2\alpha+1}{2\alpha}\left(\mathrm{Var}(T) + \frac{2(\alpha+1)}{\alpha}E(TU^\alpha) - \frac{2}{\alpha}\right).$$

When $T = (\alpha+1)(1-U^\alpha)/\alpha$, $\mathrm{Var}(X_n) = o(n^{2\alpha}L^2(n))$.

Tree traversals. The simplest example is when $T_n = 1$ for $n \geq 1$. The distribution is essentially the Stirling numbers of the first kind; see (49). This classical example also appears in a large number of problems; see Bai, Hwang, and Liang [3] for some examples. This distribution also has another concrete interpretation: the depth of the first node in inorder traversal.

Interestingly, the depth of the first node in postorder traversal of a random binary search tree satisfies a slightly different recurrence: $P_0(y) = P_1(y) = 1$, and, for $n \geq 2$,

$$P_n(y) = \frac{e^y}{n}\sum_{1\leq k<n} P_k(y) + \frac{e^y}{n}P_{n-1}(y), \tag{53}$$

which can be asymptotically solved as

$$P_n(y) = \frac{n^{e^y-1}}{\Gamma(y)}\varpi(e^y)\left(1+O(n^{-1})\right) + O(n^{-1}),$$

uniformly for $|y| \leq \delta$, where

$$\varpi(y) = e^y + \int_0^1 w^y e^{yw}\left(1-y-yw^{-1}\right)\mathrm{d}w.$$

This is derived by applying singularity analysis (see [29]) to the generating function $P(z, e^y) = \sum_n P_n(y)z^n$, which satisfies

$$P(z,y) = (1-z)^{-y}e^z + (1-z)^{-y}e^{-y(1-z)}\int_{1-z}^1 w^y e^{yw}\left(1-y-yw^{-1}\right)\mathrm{d}w.$$

Therefore, the distribution of X_n is asymptotically Poisson with parameter $\log n$ and thus asymptotically normal; see [39]. The mean was discussed by Brinck [10].

Quickselect. The number of comparisons and exchanges used by quickselect to find the smallest (or the largest) elements satisfies (3) with toll functions of linear mean. Our theorems apply and, in particular, the limit law of the number of comparisons is Dickman. The same limit law actually persists for selecting the mth smallest (or largest) element when $m = o(n)$; see Hwang and Tsai [40] for more details.

The Stirling distribution also naturally appears as the number of partitioning stages used by quickselect to find the smallest or the largest element. This gives yet another addition to the large list of concrete interpretations of the Stirling numbers of the first kind.

Logarithmic product of cycle sizes in random permutation. Permutations can be decomposed into a set of cycles. Given a random permutation of n elements, let $\sigma_1 \leq \cdots \leq \sigma_k$ denote the cycle sizes. Define $X_n := \sum_{1\leq j\leq k}\log\sigma_j$, which appeared as a good approximation to the logarithmic order of a random permutation. Then X_n satisfies (1) with $T_n = \log n$ and

$$E(e^{X_n y}) = \prod_{1\leq k\leq n}\left(1+\frac{y^k-1}{k}\right).$$

Our result gives the well-known asymptotic normality of X_n with mean $\frac{1}{2}\log^2 n$ and variance $\frac{1}{3}\log^3 n$; see Barbour and Tavaré [4] for further information.

Acknowledgment. We thank Uwe Rösler for helpful suggestions.

REFERENCES

[1] D. Aldous, *Asymptotic fringe distributions for general families of random trees*, Ann. Appl. Probab., 1 (1991), pp. 228–266.
[2] A. Andersson, *A note on the expected behaviour of binary tree traversals*, Comput. J., 33 (1990), pp. 471–472.
[3] Z.-D. Bai, H.-K. Hwang, and W.-Q. Liang, *Normal approximations of the number of records in geometrically distributed random variables*, Random Structures Algorithms, 13 (1998), pp. 319–334.
[4] A. D. Barbour and S. Tavaré, *A rate for the Erdős-Turán law*, Combin. Probab. Comput., 3 (1994), pp. 167–176.
[5] J. L. Bentley and M. D. McIlroy, *Engineering a sort function*, Software—Practice and Experience, 23 (1993), pp. 1249–1265.
[6] F. Bergeron, P. Flajolet, and B. Salvy, *Varieties of increasing trees*, in CAAP'92 (Rennes, 1992), Lecture Notes in Comput. Sci. 581, Springer-Verlag, Berlin, 1992, pp. 24–48.
[7] P. J. Bickel and P. A. Freedman, *Some asymptotic theory for the bootstrap*, Ann. Statist., 9 (1981), pp. 1196–1217.
[8] P. Billingsley, *Probability and Measure*, 3rd ed., John Wiley and Sons, New York, 1995.
[9] N. H. Bingham, C. M. Goldie, and J. L. Teugels, *Regular Variation*, Cambridge University Press, Cambridge, UK, 1989.
[10] K. Brinck, *The expected performance of traversal algorithms in binary trees*, Comput. J., 28 (1985), pp. 426–432.
[11] K. Brinck and N. Y. Foo, *Analysis of algorithms on threaded trees*, Comput. J., 24 (1981), pp. 148–155.
[12] W. A. Burkhard, *Nonrecursive traversals of trees*, Comput. J., 18 (1975), pp. 227–230.
[13] W. A. Burkhard, *Corrigendum: Nonrecursive traversals of trees*, Comput. J., 20 (1977), p. 352.
[14] H.-H. Chern and H.-K. Hwang, *Phase changes in random m-ary search trees and generalized quicksort*, Random Structures Algorithms, 19 (2001), pp. 316–358.
[15] H.-H. Chern, H.-K. Hwang, and T.-H. Tsai, *An asymptotic theory for Cauchy-Euler differential equations with applications to the analysis of algorithms*, J. Algorithms, to appear.
[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
[17] M. Cramer and L. Rüschendorf, *Analysis of recursive algorithms by the contraction method*, in Proceedings of the Athens Conference on Applied Probability and Time Series Analysis, Athens, Greece, 1995, C. C. Heyde et al., eds., Lecture Notes in Statist. 114, Springer-Verlag, New York, 1996, pp. 18–33.
[18] L. Devroye, *Limit laws for local counters in random binary search trees*, Random Structures Algorithms, 2 (1991), pp. 303–315.
[19] L. Devroye, *Universal limit laws for depths in random trees*, SIAM J. Comput., 28 (1998), pp. 409–432.
[20] L. Devroye, *Limit laws for sums of functions of subtrees of random binary search trees*, SIAM J. Comput., to appear.
[21] P. Diaconis, *Application of the method of moments in probability and statistics*, in Moments in Mathematics (San Antonio, TX, 1987), AMS, Providence, RI, 1987, pp. 125–142.
[22] R. P. Dobrow and J. A. Fill, *Total path length for random recursive trees*, Combin. Probab. Comput., 8 (1999), pp. 317–333.
[23] B. Dwyer, *Simple algorithms for traversing a tree without an auxiliary stack*, Inform. Process. Lett., 2 (1974), pp. 143–145.
[24] T. I. Fenner and G. Loizou, *A note on traversal algorithms for triply linked trees*, BIT, 21 (1981), pp. 153–156.
[25] J. A. Fill, *On the distribution of binary search trees under the random permutation model*, Random Structures Algorithms, 8 (1996), pp. 1–25.
[26] J. A. Fill and S. Janson, *Smoothness and decay properties of the limiting quicksort density function*, in Mathematics and Computer Science: Algorithms, Trees, Combinatorics, and Probabilities, D. Gardy and A. Mokkadem, eds., Birkhäuser, Basel, 2000, pp. 53–64.
[27] J. A. Fill, H. Mahmoud, and W. Szpankowski, *On the distribution for the duration of a randomized leader election algorithm*, Ann. Appl. Probab., 6 (1996), pp. 1260–1283.

[28] P. Flajolet, X. Gourdon, and C. Martínez, *Patterns in random binary search trees*, Random Structures Algorithms, 11 (1997), pp. 223–244.
[29] P. Flajolet and A. Odlyzko, *Singularity analysis of generating functions*, SIAM J. Discrete Math., 3 (1990), pp. 216–240.
[30] P. Flajolet, P. Poblete, and A. Viola, *On the analysis of linear probing hashing*, Algorithmica, 22 (1998), pp. 490–515.
[31] G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1991.
[32] D. Gordon, *Eliminating the flag in threaded binary search trees*, Inform. Process. Lett., 23 (1986), pp. 209–214.
[33] P. Grabner and H. Prodinger, *Sorting algorithms for broadcast communications: Mathematical analysis*, Theoret. Comput. Sci., to appear.
[34] I. Gutman, S. Klavzar, and B. Mohar, eds., *Fifty years of the Wiener index*, Match, 35 (1997), pp. 1–259.
[35] P. Hennequin, *Combinatorial analysis of quicksort algorithm*, RAIRO Inform. Théor. Appl., 23 (1989), pp. 317–333.
[36] P. Hennequin, *Analyse en moyenne d'algorithme, tri rapide et arbres de recherche*, Ph.D. thesis, Ecole Polytechnique, Palaiseau, France, 1991.
[37] C. A. R. Hoare, *Quicksort*, Comput. J., 5 (1962), pp. 10–15.
[38] M. Hofri and H. Shachnai, *Efficient reorganization of binary search trees*, in Algorithms and Complexity (Rome, 1994), Lecture Notes in Comput. Sci. 778, Springer-Verlag, Berlin, 1994, pp. 152–166.
[39] H.-K. Hwang, *Asymptotics of Poisson approximation to random discrete distributions: An analytic approach*, Adv. in Appl. Probab., 31 (1999), pp. 448–491.
[40] H.-K. Hwang and T.-H. Tsai, *Quickselect and Dickman function*, Combin. Probab. Comput., to appear.
[41] J. JaJa, *A perspective on quicksort*, Computing in Science and Engineering, 2 (2000), pp. 43–49.
[42] P. Kirschenhofer, H. Prodinger, and R. F. Tichy, *A contribution to the analysis of in situ permutation*, Glas. Mat. Ser. III, 22 (1987), pp. 269–278.
[43] D. E. Knuth, *Mathematical analysis of algorithms*, in Information Processing '71, Proceedings of the IFIP Congress, Ljubljana, 1971, Vol. 1: Foundations and systems, North–Holland, Amsterdam, 1972, pp. 19–27.
[44] M. Loève, *Probability Theory* I, 4th ed., Springer-Verlag, New York, 1977.
[45] I. D. G. MacLeod, *An algorithm for in-situ permutation*, Austral. Comput. J., 2 (1970), pp. 16–19.
[46] M. Maejima and S. T. Rachev, *An ideal metric and the rate of convergence to a self similar process*, Ann. Probab., 15 (1987), pp. 708–727.
[47] H. M. Mahmoud, *Limiting distributions for path lengths in recursive trees*, Probab. Engrg. Inform. Sci., 5 (1991), pp. 53–59.
[48] H. M. Mahmoud, *Evolution of Random Search Trees*, John Wiley and Sons, New York, 1992.
[49] H. M. Mahmoud, *Sorting. A Distribution Theory*, Wiley-Interscience, New York, 2000.
[50] H. M. Mahmoud, R. Modarres, and R. T. Smythe, *Analysis of quickselect: An algorithm for order statistics*, RAIRO Inform. Théor. Appl., 29 (1995), pp. 255–276.
[51] J. M. Morris, *Traversing binary trees simply and cheaply*, Inform. Process. Lett., 9 (1979), pp. 197–200.
[52] R. Neininger, *Limit Laws for Random Recursive Structures Algorithms*, Ph.D. Dissertation, Institut für Mathematische Stochastik, Universität Freiburg, Freiburg, Germany, 1999; also available online from www.stochastik.uni-freiburg.de/homepages/neininger/.
[53] R. Neininger, *On a multivariate contraction method for random recursive structures with applications to quicksort*, Random Structures Algorithms, 19 (2001), pp. 498–524.
[54] R. Neininger, *On binary search tree recursions with monomials as toll functions*, J. Comput. Appl. Math., 142 (2002), pp. 185–196.
[55] R. Neininger, *Wiener index of random trees*, Combin. Probab. Comput., to appear.
[56] R. Neininger and L. Rüschendorf, *A General Contraction Theorem and Asymptotic Normality in Combinatorial Structures*, preprint, Institut für Mathematische Stochastik, Universität Freiburg, Freiburg, Germany, 2001; also available online from http://www.stochastik.uni-freiburg.de/homepages/neininger.
[57] A. Panholzer and H. Prodinger, *Binary search tree recursions with harmonic toll functions*, J. Comput. Appl. Math., to appear.
[58] B. Pittel, *Normal convergence problem? Two moments and a recurrence may be the clues*, Ann. Appl. Probab., 9 (1999), pp. 1260–1302.

[59] H. Prodinger, *How to select a loser*, Discrete Math., 120 (1993), pp. 149–159.
[60] S. T. Rachev, *Probability Metrics and the Stability of Stochastic Models*, John Wiley, New York, 1991.
[61] S. T. Rachev and L. Rüschendorf, *A new ideal metric with applications to multivariate stable limit theorems*, Probab. Theory Related Fields, 94 (1992), pp. 163–187.
[62] S. T. Rachev and L. Rüschendorf, *Probability metrics and recursive algorithms*, Adv. in Appl. Probab., 27 (1995), pp. 770–799.
[63] S. T. Rachev and L. Rüschendorf, *Mass Transportation Problems, Vol.* 1: *Theory*, Springer-Verlag, New York, 1998.
[64] M. Régnier, *A limiting distribution for quicksort*, RAIRO Inform. Théor. Appl., 23 (1989), pp. 335–343.
[65] J. M. Robson, *An improved algorithm for traversing binary trees without auxiliary stack*, Inform. Process. Lett., 2 (1973), pp. 12–14.
[66] U. Rösler, *A limit theorem for "Quicksort,"* RAIRO Inform. Théor. Appl., 25 (1991), pp. 85–100.
[67] U. Rösler, *A fixed point theorem for distributions*, Stochastic Process. Appl., 42 (1992), pp. 195–214.
[68] U. Rösler, *The analysis of stochastic divide and conquer algorithms*, Algorithmica, 29 (2001), pp. 238–261.
[69] U. Rösler and L. Rüschendorf, *The contraction method for recursive algorithms*, Algorithmica, 29 (2001), pp. 3–33.
[70] W. Schachinger, *Limiting distribution of the costs of partial match retrievals in multidimensional tries*, Random Structures Algorithms, 17 (2000), pp. 428–459.
[71] R. Sedgewick, *Quicksort*, Gurland Publishing, New York, 1980.
[72] S.-H. Shiau and C.-B. Yang, *A fast sorting algorithm and its generalization on broadcast communications*, in Computing and Combinatorics (Sydney, 2000), Lecture Notes in Comput. Sci. 1858, Springer-Verlag, New York, 2000, pp. 252–261.
[73] R. T. Smythe and H. M. Mahmoud, *A survey of recursive trees*, Theory Probab. Math. Statist., 51 (1996), pp. 1–27.
[74] N. Trinajstić, *Chemical Graph Theory*, Vol. 2, CRC Press, Boca Raton, FL, 1983.
[75] J. D. Valois, *Introspective sorting and selection revisited*, Software—Practice and Experience, 30 (2000), pp. 617–638.
[76] V. M. Zolotarev, *Ideal metrics in the problem of approximating distributions of sums of independent random variables*, Theory Probab. Appl., 22 (1977), pp. 433–449.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1723–1744

ARE BITVECTORS OPTIMAL?*

H. BUHRMAN†, P. B. MILTERSEN‡, J. RADHAKRISHNAN§, AND S. VENKATESH¶

Abstract. We study the *static membership problem*: Given a set S of at most n keys drawn from a universe U of size m, store it so that queries of the form "Is u in S?" can be answered by making few accesses to the memory. We study schemes for this problem that use space close to the information theoretic lower bound of $\Omega(n \log(\frac{m}{n}))$ bits and yet answer queries by reading a small number of bits of the memory.

We show that, for $\epsilon > 0$, there is a scheme that stores $O(\frac{n}{\epsilon^2} \log m)$ bits and answers membership queries using a randomized algorithm that reads just one bit of memory and errs with probability at most ϵ. We consider schemes that make no error for queries in S but are allowed to err with probability at most ϵ for queries not in S. We show that there exist such schemes that store $O((\frac{n}{\epsilon})^2 \log m)$ bits and answer queries using just one bitprobe. If multiple probes are allowed, then the number of bits stored can be reduced to $O(n^{1+\delta} \log m)$ for any $\delta > 0$. The schemes mentioned above are based on probabilistic constructions of set systems with small intersections.

We show lower bounds that come close to our upper bounds (for a large range of n and ϵ): Schemes that answer queries with just one bitprobe and error probability ϵ must use $\Omega(\frac{n}{\epsilon \log(1/\epsilon)} \log m)$ bits of storage; if the error is restricted to queries not in S, then the scheme must use $\Omega(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m)$ bits of storage.

We also consider deterministic schemes for the static membership problem and show tradeoffs between space and the number of probes.

Key words. data structures, set membership problem, bit probe model, set systems

AMS subject classifications. 68P05, 68P10, 68P20

PII. S0097539702405292

1. Introduction.

In this paper, we study the *static membership* problem: Given a subset S of at most n keys from a universe $U = \{1, 2, \ldots, m\}$, store it so that queries of the form "Is u in S?" can be answered by making few accesses to the memory. This is a fundamental data structure problem with a long history. Yao [37] showed that, if the data structure consists of a table with n cells, where the keys are stored explicitly and the universe from which the set S is chosen is large enough, then the sorted table with binary search is optimal. In order to study data structures where elements of the set S are not stored explicitly, Yao (in the same paper) proposed the *cell probe model*. In this model, the set S is stored as a table of cells, each capable of holding one element of the universe; that is, if the universe has size m, where m is a power of two, then each cell holds $\log m$ bits. Queries are to be answered by probing the table adaptively; that is, each probe can depend on the results of earlier probes and the query element u. The goal is to process membership queries with as few probes as possible and, at the same time, keep the size of the table small.

*Received by the editors November 15, 2001; accepted for publication (in revised form) April 12, 2002; published electronically September 5, 2002.
http://www.siam.org/journals/sicomp/31-6/40529.html

†CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (buhrman@cwi.nl).

‡BRICS and Department of Computer Science, University of Aarhus, 8000 Aarhus C, Denmark (bromille@brics.dk).

§School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India (jaikumar@tcs.tifr.res.in).

¶Institute for Advanced Study, Olden Lane, Princeton, NJ, 08540 (venkat@math.ias.edu). Most of this work was done while the author was at the Tata Institute of Fundamental Research, Mumbai 400005, India; part of this work was done while the author was visiting BRICS at the University of Aarhus.

Fredman, Komlós, and Szemerédi [16] gave a solution for the static membership problem in the cell probe model that used a constant number of probes and a table of size $O(n)$. We shall refer to this scheme as the FKS scheme. Note that, if one is required to store sets of size at most n, then there is an information theoretic lower bound of $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ on the number of bits used. For $n \le m^{1-\Omega(1)}$, this implies that the data structure must store $\Omega(n \log m)$ bits (and must, therefore, use $\Omega(n)$ cells). Thus, up to constant factors, the FKS scheme uses optimal space and number of cell probes. In fact, Fiat et al. [12], Brodnik and Munro [5, 6], and Pagh [25] obtain schemes that use space (in bits) that is within a small additive term of $\lceil \log \sum_{i \le n} \binom{m}{i} \rceil$ and yet answer queries by reading at most a constant number of cells.

An important variation of the cell probe model is the *bitprobe* model, where each cell holds just a single bit rather than an element of the universe. Thus, in this model, the query algorithm is given bitwise access to the data structure. Arguably, the bitprobe complexity of a data structure problem is a fundamental measure; this, in particular, applies to decision problems such as the membership problem, where the final answer to a query *is* a single bit. The bitprobe model is older than the cell probe model. The membership problem was studied in the bitprobe model already by Minsky and Papert in their 1969 book *Perceptrons* [24]. They were interested in *average*-case upper bounds for this problem and did not study worst-case bounds. Although the bitprobe complexity of several other static and dynamic data structure problems has been studied since then [9, 13, 14, 15, 22, 36], the bitprobe complexity of the static membership problem has received very little attention since the work of Minsky and Papert.[1] In this paper, we study *worst*-case bounds for the membership problem. Thus our goal is to *answer queries using the minimum number of bitprobes* and, at the same time, *keep the number of bits stored in the table small*.

1.1. Randomized schemes. We investigate the complexity of the static membership problem when the query processing algorithm tosses coins to decide which bits of the memory to read and is allowed to answer incorrectly with a certain small probability. Though using Las Vegas–style randomization to *construct* data structures is a well-known and established technique (used, for instance, in many hashing-based data structures, such as the FKS scheme), Monte Carlo–style randomization in the *query algorithm* has been used in the field of data structures only very recently [23, 19]. We consider two kinds of randomized schemes: (a) those that make one-sided errors, where the errors are restricted to negative instances alone (that is, these schemes never say "No" when the query element u is in the set S); (b) schemes that are allowed to make two-sided errors (that is, errors are allowed for positive as well as negative instances). It is also possible to consider schemes that make errors on positive instances alone; but for the important case of one-probe schemes, we show that such schemes cannot do better than deterministic schemes.

1.1.1. Randomized schemes with two-sided error. Our main result says that there are randomized schemes that use *just one bitprobe* and yet use space close to the information theoretic lower bound of $\Omega(n \log m)$ bits.

THEOREM 1. *For any $0 < \epsilon \le \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O(\frac{n}{\epsilon^2} \log m)$ bits so that any membership*

[1] The only exception that we are aware of is a remark by Yao and Yao [36] stating without proof that, if one ignores constant factors, the FKS scheme is optimal in the bitprobe model as well: that is, every scheme that uses $O(n \log m)$ bits of storage must use $\Omega(\log m)$ bitprobes (assuming $n \ll m$). For justification and generalization of this remark, see Theorem 6 of this paper.

query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm which probes the memory at just one location determined by its coin tosses and the query element u.

In contrast, deterministic schemes that answer queries using a single bitprobe need m bits of storage (see Theorem 11). By allowing randomization, we can reduce this bound (for constant ϵ) to $O(n \log m)$ bits. This is within a *constant factor* of the space used by a sorted table or a hash table; for $n \leq m^{1-\Omega(1)}$, it is within a constant factor of the *information theoretic* minimum number of bits needed to store the data. *Yet membership queries can be answered with small error probability by looking at a single bit of the data structure.* Note that we allow randomization only in the query algorithm; it is still the case that, for each set S, there is exactly one associated data structure $\phi(S)$. Also, the probability of error is at most ϵ for *all* sets and *all* queries.

Many of the previous results for the membership problem have been based on hashing [16, 37, 36]. We depart from this tradition. The proof of Theorem 1 is based on two-colorings of set systems. The set systems we use are related to those considered by Erdős, Frankl, and Füredi [10] in their study of r-cover-free families of sets. Similar set systems have been used to great advantage in the construction of pseudorandom generators and extractors, starting with the papers of Nisan [26] and Nisan and Wigderson [27] (for recent applications, see [35, 31, 30, 2]); we refer to them as NW-designs.

The properties of NW-designs are, unfortunately, not strong enough for our proof. So we construct an appropriate set system ourselves. In section 3, we describe this set system in more detail and relate it to the existence of a certain kind of strong *expander graphs*. Although we believe that such schemes can have practical uses, our proof relies on an existential argument, which we have not been able to make constructive. Subsequently, Ta-Shma [32], using recent developments in pseudorandomess [33, 34], has obtained an explicit one-probe randomized scheme with two-sided error ϵ for a certain range of n and ϵ. This scheme uses less space than the scheme in Theorem 9.

Is the result of Theorem 1 the best possible? As remarked above, $\Omega(n \log m)$ bits of space are necessary if $n \leq m^{1-\Omega(1)}$. So let us concentrate on the dependence of the size on the error probability ϵ. Unfortunately, our construction does not permit us to have subconstant error probability and still use optimal space. We show that this limitation is unavoidable: if ϵ is made subconstant, then we must use more than $n \log m$ space.

THEOREM 2. *Suppose $\frac{n}{m^{1/3}} \leq \epsilon \leq \frac{1}{4}$. Then any two-sided ϵ-error randomized scheme which answers queries using one bitprobe must use space $\Omega(\frac{n}{\epsilon \log(1/\epsilon)} \log m)$.*

Interestingly, the proof uses upper bounds as well as lower bounds for r-cover-free families [10].

1.1.2. Randomized schemes with one-sided error. As stated above, we do not use NW-designs in our proof of Theorem 1 but instead use a related set system. If we use NW-designs, then we do not get the same savings in space. However, we can now ensure that the error made while processing the query is one-sided.

THEOREM 3. *For any $0 < \epsilon \leq \frac{1}{4}$, there is a scheme for storing subsets S of size at most n of a universe of size m using $O((\frac{n}{\epsilon})^2 \log m)$ bits so that any membership query "Is $u \in S$?" can be answered with error probability at most ϵ by a randomized algorithm which makes a single bitprobe to the data structure. Furthermore, if $u \in S$, the probability of error is 0.*

Note that the dependence on n is now quadratic, unlike in the two-sided scheme, where it was linear. Though this scheme does not operate with optimal space, it still

uses significantly less space than a bitvector. We also show that the scheme we have is essentially optimal: there is necessarily a quadratic dependence on $\frac{n}{\epsilon}$ for any scheme with one-sided error.

THEOREM 4. *Suppose* $\frac{n}{m^{1/3}} \leq \epsilon \leq \frac{1}{4}$. *Consider the static membership problem for sets* S *of size at most* n *from a universe of size* m. *Then any scheme with one-sided error* ϵ *that answers queries using at most one bitprobe must use* $\Omega(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m)$ *bits of storage.*

To prove this theorem, we again use the lower bounds for r-cover-free families.

Remarks.

1. The proof of Theorem 3 is nonconstructive. We also show that there is an explicit one-sided error randomized scheme that uses $O((\frac{n \log m}{\epsilon})^2)$ bits of storage and answers queries using one bitprobe. This result uses the explicit NW-designs (see Theorem 9).
2. One might also consider one-probe one-sided error schemes where no error is made for query elements *not* in the set S. In this case, we show (Theorem 11) that randomness does not help at all: such a scheme must use m bits of storage.

Thus there is no one-sided error, one-probe scheme that uses optimal space. However, the space requirement can be reduced if we allow more probes.

THEOREM 5. *Suppose* $0 < \delta < 1$. *There is a randomized scheme with one-sided error* $n^{-\delta}$ *that solves the static membership problem using* $O(n^{1+\delta} \log m)$ *bits of storage and* $O(\frac{1}{\delta})$ *bitprobes.*

To prove this, we combine a two-sided scheme obtained from Theorem 1 with ideas for one-sided schemes from Theorem 3.

Connection with communication complexity. Theorems 1–4 can also be viewed in the communication complexity setting: Alice gets $u \in \{1, \ldots, m\}$, Bob gets $S \subseteq \{1, \ldots, m\}$ of size at most n, and Alice sends a single message to Bob after which Bob announces whether $u \in S$. Indeed, as was pointed out in [23], this communication game characterizes the data structure problem in the following way: If s is the optimal number of bits in the data structure that can be queried with one bitprobe and a particular bound on the error on positive and negative instances, then $\log s \pm o(\log s)$ is the number of bits sent from Alice to Bob in an optimal protocol for the communication problem with the same error bounds. Thus Theorems 1–4 give bounds for the communication problem which are optimal within a low order term.

Connection with coding theory. The membership problem in the bitprobe model has an interesting coding theoretic interpretation: We are trying to give an encoding $\phi(u)$ of any m-bit string u with at most n 1's so that the length of the encoding is close to the *first order entropy* of u and so that any bit of u can be retrieved by looking at a few bits of $\phi(u)$. Thus we are trying to construct a *locally decodable source code* analogous to the locally decodable *channel* codes of [4, 21].

1.2. Deterministic schemes. As noted previously, the FKS hashing scheme is a data structure for storing sets of size at most n from a universe of size m using $O(n \log m)$ bits so that membership queries can be answered using $O(\log m)$ bitprobes. We show that the FKS scheme makes an optimal number of bitprobes, within a constant factor, for this amount of space. This fact follows from the following general time-space tradeoff.

THEOREM 6. *Suppose a deterministic scheme stores subsets of size* n *from a universe of size* m *using* s *bits of storage and answers membership queries with* t *bitprobes to memory. Then* $\binom{m}{n} \leq \max_{i \leq nt} \binom{2s}{i}$.

COROLLARY 1.1. *Let $\epsilon > 0$, $c \geq 1$ be any constants. There is a constant $\delta > 0$ so that the following holds. Let $n \leq m^{1-\epsilon}$, and let a scheme for storing sets of size at most n of a universe of size m as data structures of at most $cn \log m$ bits be given. Then any deterministic algorithm answering membership queries using this structure must make at least $\delta \log m$ bitprobes in the worst case.*

While the FKS scheme makes an optimal number of probes, the probes made are *adaptive*. In fact, adaptiveness seems to be quite inherent in hashing-based schemes. Somewhat surprisingly, as a corollary to the proof of Theorem 1, we can prove that there is a scheme that uses $O(n \log m)$ bits and answers membership queries with $O(\log m)$ *nonadaptive* bitprobes. Thus adaptive probes do not help much when we consider deterministic schemes that use $O(n \log m)$ space.

More generally, from Theorem 6, it follows that any deterministic scheme that answers queries using t bitprobes must use space at least $ntm^{\Omega(1/t)}$ in the worst case. We show the existence of schemes which almost match the lower bound.

THEOREM 7.

1. *There is a nonadaptive scheme that stores sets of size at most n from a universe of size m using $O(ntm^{\frac{2}{t+1}})$ bits and answers queries using $2t+1$ bitprobes. This scheme is nonexplicit.*
2. *There is an explicit adaptive scheme that stores sets of size at most n from a universe of size m using $O(m^{1/t} n \log m)$ bits and answers queries using $O(\log n + \log\log m) + t$ bitprobes.*

Thus, somewhat surprisingly, if we care only about space up to a polynomial, adaptive schemes are not more powerful than nonadaptive ones.

Finally, we turn our attention to deterministic *two-probe* schemes and ask if they can do better than one-probe schemes, where bitvectors are optimal. We have not been able to answer this question in general. We can show that this is the case if the two bitprobes made are nonadaptive. Thus the second bitprobe is useless for nonadaptive schemes. However, we show that there is a scheme with two adaptive bitprobes that does better than any scheme with two nonadaptive bitprobes for $n = 2$. We do not know whether a second adaptive probe helps for values of n greater than 2.

THEOREM 8.

1. *Any scheme for storing subsets S of size at most n ($n \geq 2$) of a universe of size m such that membership queries can be answered by two nonadaptive bitprobes uses space $s \geq m$ bits.*
2. *Let $m \geq 2$, and let $n = 2$. Then there is a scheme with two adaptive bitprobes and space $s = O(m^{3/4})$.*

1.3. Organization of the paper. We start with the formal definitions in the next section. In section 3, randomized schemes with one-sided error and two-sided error are presented. In section 4, we prove lower bound results for randomized schemes. We end with results on deterministic schemes in section 5.

2. Notation and definitions.

Notation. Unless mentioned explicitly, all logarithms in this paper are to the base 2. We use $[m]$ to denote the set $\{1, 2, \ldots, m\}$. For a set A, $\binom{A}{n}$ denotes the set of all subsets of A of size n, and $\binom{A}{\leq n}$ denotes the set of all its subsets of size at most n.

DEFINITION 2.1 (storing schemes). *An (n, m, s)-*storing scheme *is a method for representing any subset of size at most n of a universe of size m as an s-bit string. Formally, an (n, m, s)-storing scheme is a map ϕ from $\binom{[m]}{\leq n}$ to $\{0,1\}^s$.*

DEFINITION 2.2 (deterministic query schemes). *A deterministic* (m, s, t)-query scheme *is a family* $\{T_u\}_{u\in[m]}$ *of* m *Boolean decision trees of depth at most* t. *Each internal node in a decision tree is marked with an index between* 1 *and* s, *indicating the address of a bit in an* s*-bit data structure. For each internal node, there is one outgoing edge labeled "*0*" and one labeled "*1*." The leaf nodes of every tree are marked "Yes" or "No." Such a tree* T_u *induces a map from* $\{0,1\}^s$ *to* $\{$*Yes, No*$\}$; *this map will also be referred to as* T_u.

DEFINITION 2.3 (deterministic schemes). *An* (n, m, s)*-storing scheme* ϕ *and an* (m, s, t)*-query scheme* $\{T_u\}_{u\in[m]}$ *together form an* (n, m, s, t)-scheme *if, for all* $S \in \binom{[m]}{\leq n}$, *for all* $u \in [m] : T_u(\phi(S)) =$ *Yes if and only if* $u \in S$.

In a nonadaptive scheme, the next probe to be made depends only on the input query q. It does not depend on the results of the previous probes.

DEFINITION 2.4 (nonadaptive query schemes). *A nonadaptive query scheme is a deterministic scheme where, in each decision tree, all nodes on a particular level are marked with the same index between* 1 *and* s *(but nodes on the same level in different trees may be marked differently).*

In a randomized scheme, the storing scheme is deterministic as before. However, the query algorithm is allowed to make random coin tosses to decide the next location to be probed.

DEFINITION 2.5 (randomized schemes). *A randomized* (m, s, t)*-query scheme is a family* $\{\pi_u\}_{u\in[m]}$ *of probability distributions on the set of all Boolean decision trees of depth at most* t. *We answer the query "Is* u *in* S*?" by picking a decision tree according to the distribution* π_u *and return the answer it gives. An* (n, m, s)*-storing scheme and a randomized* (m, s, t)*-query scheme together form an* (n, m, s, t)*-randomized scheme. We say that a randomized scheme has* positive one-sided error ϵ *if, for* $u \notin S$, *the error probability on queries "Is* u *in* S*?" is* 0, *i.e., the answer "No" is always returned, while, if* $u \in S$, *the error probability on the query "Is* u *in* S*?" is at most* ϵ. *Similarly, a randomized scheme has* negative one-sided error ϵ *if, for* $u \in S$, *the error probability on queries "Is* u *in* S*?" is* 0, *i.e., the answer "Yes" is always returned, while, if* $u \notin S$, *the error probability on the query "Is* u *in* S*?" is at most* ϵ. *We say that a randomized scheme has* two-sided error ϵ *if, on query "Is* u *in* S*?," the scheme returns the wrong answer with probability at most* ϵ.

We will be interested in one-probe randomized schemes where, in particular, π_u will be a probability distribution on Boolean decision trees that make at most one probe.

We say that a scheme is *explicit* if there are efficient algorithms that can simulate the storing scheme and the query scheme.

DEFINITION 2.6 (explicit storing schemes). *A family of storing schemes, indexed by* (n, m, s), *is* explicit *if there is a Turing machine, running in time* $s^{O(1)}$, *which, given* $S \subseteq \{0,1\}^m$ *of size* n, *outputs the representation* $\phi(S)$.

DEFINITION 2.7 (explicit query schemes). *A family of (randomized) query schemes, indexed by* (m, s, t), *is* explicit *if there is a (probabilistic) Turing machine, running in time* $(t + \log m)^{O(1)}$, *which, on input* u *and with oracle access to* $\phi(S)$, *executes the correct sequence of probes according to the query scheme and accepts or rejects accordingly.*

DEFINITION 2.8 (explicit schemes). *A family of schemes is explicit if the associated storing and query schemes are explicit.*

3. Upper bounds for randomized schemes. In this section, we show that there exist randomized one-probe schemes that use small space. We first describe the

randomized scheme with one-sided error; the scheme with two-sided error can then be seen as a generalization. Randomized multiprobe schemes with one-sided error will be obtained by combining one-probe schemes with one-sided error and one-probe schemes with two-sided error.

All of our schemes will be based on set systems with small intersections. In particular, we will use a set system of the form $\{\Gamma_u\}_{u\in[m]}$, where $\Gamma_u \subseteq [s]$. The query algorithm, on receiving the query "Is u in S?," will probe a location in Γ_u uniformly and answer "Yes" if and only if it finds a 1 there.

We will describe our schemes using a bipartite graph with vertex sets U and V: U is the universe from which the set S to be stored is drawn, and V is the set of locations in the memory. We connect $u \in U$ to $v \in V$ if, on query "Is u in S?," the cell v in the memory is probed by the algorithm (for some outcome of coin tosses). Thus, when the query element is u, the algorithm probes the memory locations in the neighborhood $\Gamma(u)$ of u with uniform distribution.

3.1. Randomized scheme with one-sided error.

Proof of Theorem 3. Our randomized scheme is based on a bipartite graph with vertex sets U and V, where $U = [m]$ and $|V| = O((\frac{n}{\epsilon})^2 \log m)$. Any instance of the data structure corresponds to a coloring of V using colors from the set $\{0, 1\}$. Hence, if the set $S \subseteq U$ is to be stored correctly, then all locations in $\bigcup_{u\in S} \Gamma(u)$ must be colored 1. This is because the algorithm is not allowed to say "No" when the query element u is in S. Furthermore, since the error probability is at most ϵ, for all $u' \notin S$, at most $\epsilon|\Gamma(u')|$ locations in $\Gamma(u')$ can be colored 1. Thus we get the following condition on the bipartite graph:

$$\forall S \in \binom{U}{\leq n}, \forall u' \in U - S, \left|\Gamma(u') - \bigcup_{u\in S} \Gamma(u)\right| \geq (1-\epsilon)|\Gamma(u')|. \tag{1}$$

We are thus required to pick neighborhoods for vertices $u \in U$ so that the resulting bipartite graph satisfies (1). An NW-design allows us to do this.

Definition 3.1. *A family of sets F is an (m, ℓ, a)-design if F has m sets each of size ℓ and two different sets in F have at most a elements in common.*

Lemma 3.2 (Erdős, Frankl, and Füredi [10, Theorem 2.1]). *If $m \leq \binom{s}{a}/\binom{\ell}{a}^2$, then there is an $(m, \ell, a-1)$-design all of whose elements are subsets of $[s]$.*

We will use an (m, ℓ, a)-design with $a = \lceil \log m \rceil$ and $\ell = \lceil na/\epsilon \rceil$. If $s = \lceil 2e^2\ell^2/a \rceil$ (which is $O((\frac{n}{\epsilon})^2 \log m)$), then

$$\frac{\binom{s}{a}}{\binom{\ell}{a}^2} \;\geq\; \frac{(\frac{s}{a})^a}{(\frac{e\ell}{a})^{2a}} \;\geq\; \frac{(\frac{2e^2\ell^2}{a^2})^a}{(\frac{e\ell}{a})^{2a}} \;\geq\; 2^a \;\geq\; m,$$

and, by the above lemma, there is an (m, ℓ, a)-design (in fact, an $(m, \ell, a-1)$-design), all of whose elements are subsets of $[s]$. Let $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$ be such a design.

Storing scheme. Suppose the set to be stored is $\{u_1, \ldots, u_k\}$, $k \leq n$. Store a bitstring T of size s with 1's in all locations in $M = \Gamma_{u_1} \cup \Gamma_{u_2} \cup \cdots \cup \Gamma_{u_k}$ and 0 elsewhere.

Query scheme. On query "Is u in S?," do the following:

Step 1. Pick a random location i uniformly from Γ_u.

Step 2. If $T(i)$ is 1, say "Yes," otherwise say "No."

Correctness. If $u \in S$, then every location in Γ_u has a 1, and hence we say "Yes." On the other hand, if $u \notin S$, then $|M \cap \Gamma_u| \leq na$. Hence the probability that

the algorithm says "Yes" when we choose a random location in Γ_u is at most $\frac{na}{|\Gamma_u|} = \frac{na}{\ell} \leq \epsilon$. □

The following theorem gives a slightly weaker bound than the one stated in Theorem 3, but it has the advantage that the scheme is explicit. It is based on the explicit version of Lemma 3.2 (see [10, Example 3.2] or [27]).

THEOREM 9. *For any $\epsilon > 0$ and any n, m, there is an (n, m, s)-storing scheme with $s = O((\frac{n \log m}{\epsilon})^2)$ and an associated randomized one-probe query scheme with a negative one-sided error at most ϵ. This scheme is explicit.*

Proof. Let $\mathbf{F}$ be a finite field of size q, where q is the smallest power of two which is at least $(n \log m)/\epsilon$. If $d = \lceil \log m \rceil - 1$, then the number of univariate polynomials of degree at most d is $q^{d+1} \geq m$. We will associate with the element $u \in [m]$ a unique polynomial

$$p_u(X) \stackrel{\text{def}}{=} \sum_{i=1}^{\lceil \log m \rceil} u_i X^{i-1},$$

where u_i is the ith bit in the binary representation of u. Now we store S as a $q \times q$ bitmap, indexed by $\mathbf{F} \times \mathbf{F}$, with bit (x, y) on if and only if $p_u(x) = y$ for some $u \in S$. The size of the data structure is as claimed. To answer the query "Is u in S?," we pick $x \in \mathbf{F}$ at random and say "Yes" if and only if bit $(x, p_u(x))$ of the bitmap is 1. Clearly, if $u \in S$, we always say "Yes." If $u \notin S$, then, for all $u' \in S$, the graphs of p_u and $p_{u'}$ have at most d points in common. Thus at most nd locations of the form $(x, p_u(x))$ of the bitmap will contain a 1. Note that $nd/q \leq \epsilon$. □

Remark. By choosing parameters more carefully in the above proof, we can reduce the space requirement in the above theorem to $O(\frac{n \log m}{\epsilon \log((n \log m)/\epsilon)})^2$.

3.2. Randomized scheme with two-sided error. We now present a scheme that uses space $O(\frac{n}{\epsilon^2} \log m)$ and answers queries using a single-bit probe, making an error with probability at most ϵ. The space needed depends linearly on n, and, when ϵ is a constant, it is within a constant factor of a sorted table or a hash table.

Proof of Theorem 1. We may assume that m is large, say, $m \geq 100$. We first describe the main ideas of our randomized scheme using the bipartite graph $G = (U, V, E)$, where $U = [m]$ and $V = [s]$. On query "Is u in S?," the query algorithm probes a random location in $\Gamma(u)$ and says "Yes" if and only if the location probed contains 1. Now suppose that we need to store the set $S \subseteq U$ ($|S| \leq n$). Then we need a coloring $\chi_S : V \to \{0, 1\}$, such that, for all $u \in S$, at least $(1 - \epsilon)|\Gamma(u)|$ elements of $\Gamma(u)$ are colored 1, and, for all $u' \notin S$, at least $(1 - \epsilon)|\Gamma(u')|$ elements of $\Gamma(u')$ are colored 0. Thus we need to find neighborhoods for the vertices so that the system of sets $\{\Gamma(u)\}_{u \in U}$ admits such a coloring χ_S for all $S \in \binom{U}{\leq n}$. This motivates the following definition.

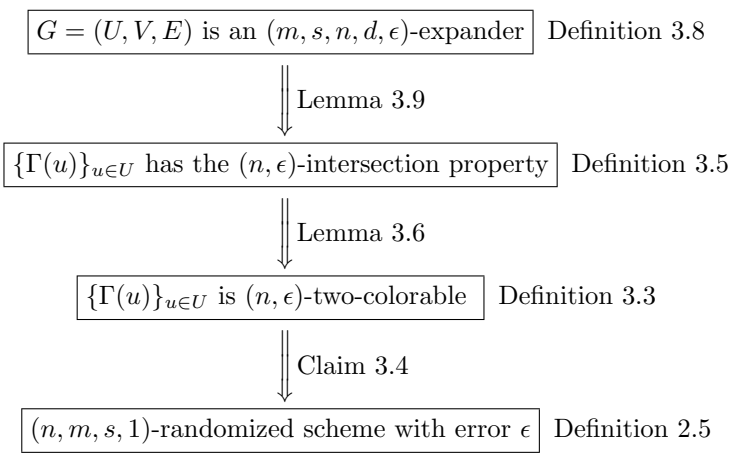
DEFINITION 3.3. *Let $\mathcal{C}_1, \mathcal{C}_0 \subseteq 2^V$. We say that $(\mathcal{C}_1, \mathcal{C}_0)$ is ϵ-two-colorable if there exists $\chi : V \to \{0, 1\}$, such that*

1. *for all $T \in \mathcal{C}_1$, $|\chi^{-1}(0) \cap T| \leq \epsilon |T|$, and*
2. *for all $T \in \mathcal{C}_0$, $|\chi^{-1}(1) \cap T| \leq \epsilon |T|$.*

We say that $\mathcal{C} \subseteq 2^V$ is (n, ϵ)-two-colorable if $(\mathcal{C}_1, \mathcal{C}_0 = \mathcal{C} - \mathcal{C}_1)$ is ϵ-two-colorable for all $\mathcal{C}_1 \in \binom{\mathcal{C}}{\leq n}$.

In this terminology, our goal can be stated as follows: find a bipartite graph $G = (U, V, E)$, with $U = [m]$ and $V = [s]$, such that $\{\Gamma(u)\}_{u \in U}$ is an (n, ϵ)-two-colorable collection of m distinct nonempty sets. We will show that such a graph exists with $|V| = O(\frac{n}{\epsilon^2} \log m)$. For this, we first identify a sufficient condition, which

we call *the* (n,ϵ)*-intersection property*, for a collection to be (n,ϵ)-two-colorable. We then observe that, if G has a certain *expansion property*, then $\{\Gamma(u)\}_{u\in U}$ has the (n,ϵ)-intersection property. Finally, using a probabilistic argument, we show that graphs with the required expansion property exist. To outline the main steps of our proof, we show the following implications and the existence of an (m,s,n,d,ϵ)-expander with $s = O(\frac{n}{\epsilon^2}\log m)$.



Colorable families and randomized schemes.

LEMMA 3.4. *Suppose the bipartite graph $G = (U,V,E)$, with $U = [m]$ and $V = [s]$, is such that $\{\Gamma(u)\}_{u\in U}$ is an (n,ϵ)-two-colorable collection of m nonempty sets. Then there is a randomized scheme for the membership problem that uses s bits to store sets $S \in \binom{U}{\le n}$ and answers membership queries using one bitprobe, and with error probability at most ϵ.*

Proof.

The storing scheme. To store the set $S \in \binom{U}{\le n}$, consider the collections $\mathcal{C}_1 = \{\Gamma(u)\}_{u\in S}$ and $\mathcal{C}_0 = \{\Gamma(u)\}_{u\notin S}$. Since $\{\Gamma(u)\}_{u\in U}$ is an (n,ϵ)-two-colorable collection, $(\mathcal{C}_1,\mathcal{C}_0)$ is ϵ-two-colorable. Let $\chi : V \to \{0,1\}$ be a coloring satisfying the two conditions of Definition 3.3. We store χ as a table of $s = |V|$ bits.

The query scheme. Given a query "Is u in S?," pick i uniformly at random from $\Gamma(u)$, and return "Yes" if $\chi(i) = 1$, and return "No" if $\chi(i) = 0$.

Correctness. If $u \in S$, then $\Gamma(u) \in \mathcal{C}_1$, and $|\chi^{-1}(0) \cap \Gamma(u)| \le \epsilon|\Gamma(u)|$. Thus $\Pr[\chi(i) = 0] \le \epsilon$. Similarly, if $u \notin S$, $\Gamma(u) \in \mathcal{C}_0$ and $\Pr[\chi(i) = 1] \le \epsilon$. □

Intersection property and coloring. Suppose that we wish to ϵ-two-color $(\mathcal{C}_1,\mathcal{C}_0)$. The first thing to try would be to color all of the elements in $\bigcup_{S\in\mathcal{C}_1} S$ with 1 and the rest with 0. Unfortunately, some sets in $\mathcal{C}_0$ might get more than an ϵ fraction of their elements colored 1, and our coloring might not be proper. So we need to first identify those sets in $\mathcal{C}_0$ that might be badly colored in this method: these are precisely the sets that have large intersection with the union of the sets in $\mathcal{C}_1$. Let $\mathcal{C}_0'$ be the collection of these sets. We will now modify our coloring method to pay special attention to sets in $\mathcal{C}_0'$. For now, ignore the sets in $\mathcal{C}_0 - \mathcal{C}_0'$, for they are at no risk of being badly colored while ensuring that the sets in $\mathcal{C}_1$ are properly colored. So we are left with the problem of ϵ-two-coloring $(\mathcal{C}_1,\mathcal{C}_0')$. If $|\mathcal{C}_0'| < |\mathcal{C}_1|$, this is a smaller problem, and we can use induction for this. This motivates the following definition of the ϵ-intersection property. That this definition is sufficient for ϵ-two-colorability

is the main observation of this section, which we state formally in Lemma 3.6 below.

DEFINITION 3.5. *Suppose $\mathcal{C}_1$ and $\mathcal{C}_0$ are collections of sets. We say that $(\mathcal{C}_1, \mathcal{C}_0)$ has the ϵ-intersection property if*

$$\forall\, \mathcal{C}_1' \subseteq \binom{\mathcal{C}_1}{\leq n} (\mathcal{C}_1' \neq \emptyset),\ \left|\left\{S \in \mathcal{C}_0 : \left|T \cap \left(\bigcup_{T' \in \mathcal{C}_1'} T'\right)\right| > \epsilon|T|\right\}\right| < |\mathcal{C}_1'| \text{ and} \tag{2}$$

$$\forall\, \mathcal{C}_0' \subseteq \binom{\mathcal{C}_0}{\leq n} (\mathcal{C}_0' \neq \emptyset),\ \left|\left\{T \in \mathcal{C}_1 : \left|T \cap \left(\bigcup_{T' \in \mathcal{C}_0'} T'\right)\right| > \epsilon|T|\right\}\right| < |\mathcal{C}_0'|. \tag{3}$$

We say that a collection of sets $\mathcal{C}$ has the (n, ϵ)-intersection property if $(\mathcal{C}_1, \mathcal{C}_0 = \mathcal{C} - \mathcal{C}_1)$ has the ϵ-intersection property for all $\mathcal{C}_1 \in \binom{\mathcal{C}}{\leq n}$.

LEMMA 3.6. *Suppose $\mathcal{C}_1, \mathcal{C}_0 \subseteq 2^V$. If $|\mathcal{C}_1| \leq n$ and $(\mathcal{C}_1, \mathcal{C}_0)$ has the (n, ϵ)-intersection property, then $(\mathcal{C}_1, \mathcal{C}_0)$ is ϵ-two-colorable.*

COROLLARY 3.7. *If $\mathcal{C}$ has the (n, ϵ)-intersection property, then $\mathcal{C}$ is (n, ϵ)-two-colorable.*

Proof of Lemma 3.6. We use induction on $|\mathcal{C}_1| + |\mathcal{C}_0|$. The base case, when either $\mathcal{C}_1$ or $\mathcal{C}_0$ is empty, is obvious. For the induction step, consider a pair $(\mathcal{C}_1, \mathcal{C}_0)$ of nonempty collections, with $|\mathcal{C}_1| \leq n$, that has the (n, ϵ)-intersection property. We may assume that $|\mathcal{C}_1| \leq |\mathcal{C}_0|$; otherwise, interchange the roles of 0 and 1. Let

$$\mathcal{C}_0' \stackrel{\text{def}}{=} \left\{T \in \mathcal{C}_0 : \left|T \cap \left(\bigcup_{T' \in \mathcal{C}_1} T'\right)\right| > \epsilon|T|\right\}.$$

Since $(\mathcal{C}_1, \mathcal{C}_0)$ has the (n, ϵ)-intersection property and $1 \leq |\mathcal{C}_1| \leq n$, $|\mathcal{C}_0'| < |\mathcal{C}_1| \leq |\mathcal{C}_0|$. By induction, there exists a two-coloring $\chi : V \to \{0, 1\}$ such that

$$\forall\, T \in \mathcal{C}_1,\ |\chi^{-1}(0) \cap T| \leq \epsilon|T| \text{ and} \tag{4}$$

$$\forall\, T \in \mathcal{C}_0',\ |\chi^{-1}(1) \cap T| \leq \epsilon|T|. \tag{5}$$

We may assume that, if $\chi(v) = 1$, then $v \in \bigcup_{T \in \mathcal{C}_1} T$ (otherwise, change $\chi(v)$ to 0—this cannot hurt (4) and can only help (5)), that is,

$$\chi^{-1}(1) \subseteq \bigcup_{T \in \mathcal{C}_1} T. \tag{6}$$

We claim that χ is also an ϵ-two-coloring of $(\mathcal{C}_1, \mathcal{C}_0)$. For, if $T \in \mathcal{C}_1$, this follows from (4), if $T \in \mathcal{C}_0'$, then it follows from (5), and, if $T \in \mathcal{C}_0 - \mathcal{C}_0'$, then

$$|\chi^{-1}(1) \cap T| \leq \left|T \cap \bigcup_{T' \in \mathcal{C}_1} T'\right| \leq \epsilon|T|,$$

where the first inequality follows from (6) and the second follows from the definition of $\mathcal{C}_0'$. ∎

Expanders and the intersection property. We now relate the (n, ϵ)-intersection property of $\{\Gamma(u)\}_{u \in U}$ to a certain expansion property of $G = (U, V, E)$.

DEFINITION 3.8. *We say that $G = (U, V, E)$ is an (m, s, n, d, ϵ)-expander if $U = [m]$, $V = [s]$, each vertex in U has degree d, and the following condition holds:*

$$\forall S \in \tbinom{U}{\leq 2n},\ |\Gamma(S)| \geq \left(1 - \frac{\epsilon}{2}\right)|S|d. \tag{7}$$

LEMMA 3.9. *Suppose $\epsilon < 1$ and $G = (U, V, E)$ is an (m, s, n, d, ϵ)-expander with $d \geq 1$. Then $\{\Gamma(u)\}_{u \in U}$ is a collection of m nonempty sets and has the (n, ϵ)-intersection property.*

Proof. Since $\epsilon < 1$, we have from (7) that $|\Gamma(\{u, u'\})| > d$ for distinct $u, u' \in U$. Thus $\Gamma(u) \neq \Gamma(u')$, and there are m nonempty sets in $\{\Gamma(u)\}_{u \in U}$. Suppose $\{\Gamma(u)\}_{u \in U}$ does not have the (n, ϵ)-intersection property; then there exists a set $S = \{u_1, \ldots, u_k, v_1, \ldots, v_k\} \subseteq U$ of $2k$ (for some $k \in [n]$) distinct elements such that, for $j = 1, \ldots, k$,

$$\left| \Gamma(v_j) \cap \bigcup_{i=1}^{k} \Gamma(u_i) \right| > \epsilon d.$$

However, then $|\Gamma(S)| < 2kd - k\epsilon d \leq (1 - \frac{\epsilon}{2}) \cdot 2kd = (1 - \frac{\epsilon}{2})|S|d$, violating (7). ∎

Existence of expanders. Finally, we show that the required expander graph exists.

LEMMA 3.10. *For $\epsilon > 0$, $m \geq 8$, and $n \leq m/2$, there is an $(m, \lceil \frac{200n \log m}{\epsilon^2} \rceil, n, \lfloor \frac{\log m}{\epsilon} \rfloor, \epsilon)$-expander.*

Proof. We show the existence of the expander graph $G = (U, V, E)$, where $U = [m]$ and $V = [s]$, using a standard probabilistic argument.[2] For each vertex u in U, we independently choose its set of neighbors in V, $\Gamma(u)$ by picking without replacement $d = \lfloor \frac{\log m}{\epsilon} \rfloor$ elements from V at random. We wish to show that, with nonzero probability, the resulting graph is an expander (satisfying (7)). For $T \in \binom{U}{\leq 2n}$, let

$$\mathcal{E}(T) \stackrel{\text{def}}{=} |\Gamma(T)| < \left(1 - \frac{\epsilon}{2}\right) |T| d.$$

We wish to show that, with nonzero probability, we can (simultaneously) avoid $\mathcal{E}(T)$ for all $T \in \binom{U}{\leq 2n}$.

CLAIM 3.11. $\Pr[\mathcal{E}(T)] \leq \left(\frac{2}{m}\right)^{2|T|}$.

Proof. Let $t \stackrel{\text{def}}{=} |T|$. We prove the claim under the assumption that the neighbors of $u \in U$ are chosen by sampling with replacement. This will imply the claim even for sampling without replacement: to pick a random set of size d, first pick d elements with replacement, resulting in d' distinct elements (say), and then add $d - d'$ new elements randomly. Suppose $T = \{u_1, u_2, \ldots, u_t\}$. Let $N \stackrel{\text{def}}{=} td$. With the choice of elements for $\Gamma(u_1), \ldots, \Gamma(u_t)$, where $\Gamma(u_j) = \{e_{(j-1)d+1}, \ldots, e_{jd}\}$, we associate N random variables $X_1, \ldots, X_N$, such that

$$X_i \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } e_i \in \{e_1, \ldots, e_{i-1}\}, \\ 0 & \text{otherwise.} \end{cases}$$

[2] The argument we use is different from what is usually used for showing the existence of expander graphs. Consider the random graph G obtained by choosing d random neighbors from V (with replacement) for each vertex in U. We wish to avoid the event $\exists S \subseteq U\, (|S| \leq 2n) : |\Gamma(S)| \leq (1 - \frac{\epsilon}{2})d|S|$. One usually [3, p. 331] bounds the probability of this event by

$$\sum_{i=1}^{2n} \binom{m}{i} \binom{|V|}{(1-\frac{\epsilon}{2})di} \left(\frac{(1-\frac{\epsilon}{2})di}{|V|}\right)^{di}.$$

For our choice of parameters, $d = \lfloor \frac{\log m}{\epsilon} \rfloor$ and $|V| = \lceil \frac{200n \log m}{\epsilon^2} \rceil$, this quantity is not less than 1 (for example, consider the term with $i = n$), although, in Lemma 3.10, we show that G has the required expansion property with high probability.

Thus

$$\mathcal{E}(T) \equiv \sum_{i=1}^{N} X_i > \frac{\epsilon}{2}N. \tag{8}$$

Now, for all $i \in [N]$ and $\sigma \in \{0,1\}^{i-1}$,

$$\Pr[X_i = 1 \mid X_1 X_2 \cdots X_{i-1} = \sigma] \leq \frac{(i-1)}{s} \leq \frac{2nd-1}{s} \leq \frac{\epsilon}{100}.$$

Let $p \stackrel{\text{def}}{=} \frac{\epsilon}{100}$, and define N independent random variables $Y_1, \ldots, Y_N$ such that

$$Y_i = \begin{cases} 1 & \text{with probability } \frac{\epsilon}{100}, \\ 0 & \text{otherwise.} \end{cases}$$

Then, for all k,

$$\Pr\left[\sum_{i=1}^{N} X_i \geq k\right] \leq \Pr\left[\sum_{i=1}^{N} Y_i \geq k\right]. \tag{9}$$

We will use the following form of Chernoff's bound (see, for example, Alon and Spencer [1, Theorem A.12]):

$$\Pr\left[\sum_{i=1}^{N} Y_i \geq (p+\delta)N\right] \leq \left(\frac{ep}{p+\delta}\right)^{(p+\delta)N}. \tag{10}$$

Thus

$$\begin{aligned} \Pr\left[\sum_{i=1}^{N} Y_i \geq \frac{\epsilon}{2}N\right] &\leq \left(\frac{e\epsilon/100}{\epsilon/2}\right)^{\epsilon N/2} \\ &\leq \left(\frac{1}{4}\right)^{\epsilon t d} \\ &\leq \left(\frac{1}{4}\right)^{\epsilon t(\frac{\log m}{\epsilon}-1)} \\ &\leq \left(\frac{2}{m}\right)^{2t}. \end{aligned}$$

Claim 3.11 now follows from (8) and (9). □

Since $\Pr[\mathcal{E}(\emptyset)] = 0$, Claim 3.11 implies that

$$\Pr\left[\bigvee_{T \in \binom{U}{\leq 2n}} \mathcal{E}(T)\right] \leq \sum_{t=1}^{2n} \binom{m}{t} \left(\frac{2}{m}\right)^{2t} \leq \sum_{t=1}^{2n} \left(\frac{4}{m}\right)^{t} < 1,$$

where we use our assumption, $m \geq 8$, to justify the last inequality. Thus, with nonzero probability, G is an expander. □

3.3. Multiprobe randomized scheme with one-sided error. We now show that the space requirement for schemes with one-sided error can be reduced if more bitprobes are allowed.

Proof of Theorem 5. Our scheme will have two tables. The first table, T_1, will come directly from Theorem 1 (with an appropriate choice of ϵ); the second table will be based on ideas used in Theorem 3. Our multiprobe query algorithm will, correspondingly, have two phases. In the first phase, it will probe T_1 several times (to reduce the error). In the second phase, it will probe T_2 just once. The scheme is nonadaptive: the locations to be probed are completely determined by the query element and the random string but do not depend on the actual values read.

We will view the first half of our scheme as a randomized multiprobe scheme with small two-sided error.

LEMMA 3.12. *There is a randomized* $(n, m, O(\frac{n}{\epsilon^2}\log m), t)$*-scheme that makes an error of at most* $(2e\epsilon)^{t/2}$.

Proof. We use the randomized $(n, m, s, 1)$-scheme of Theorem 1 but run the query algorithm t times (with independent coin tosses) and say "Yes" if and only if at least $t/2$ of the t runs give the answer "Yes." Using Chernoff's bound (see (10) above), we conclude that

$$\Pr[\mathsf{Error}] \;\le\; (2e\epsilon)^{t/2}. \qquad \square$$

We will use the above lemma with $\epsilon = n^{-\delta/2}/(2e)$ and $t = \lceil 4/\delta \rceil$. This will give us an $(n, m, O(n^{1+\delta}\log m), \lceil 4/\delta \rceil)$-scheme Π with error probability at most n^{-2}. We use Π to construct our multiprobe scheme $\hat{\Pi}$. The first table, T_1, of our scheme is exactly what Π uses; it has $s = O(n^{1+\delta}\log m)$ bits. The second table, T_2, has $s' = \lceil 2n^{1+\delta}\log m \rceil$ bits. To see how the contents of T_2 are decided, let us first describe how the query algorithm uses T_2.

The query algorithm. Suppose the query algorithm of Π uses random strings of length ℓ. In our query algorithm, for each $u \in [m]$, we have a sequence $\sigma_u = \langle r_1, r_2, \ldots, r_{s'} \rangle$ of strings $r_i \in \{0,1\}^\ell$. On receiving the query "Is u in S?," the algorithm first chooses i uniformly at random from $[s']$ and then uses the query algorithm of scheme Π with the ith element of σ_u (that is, r_i above) as the random string and T_1 as its table. If the answer returned by Π is "Yes," we say "Yes." If the answer is "No," we move on to table T_2, probe the location i there, and say "Yes" if and only if we read a 1. Thus, to completely specify the query algorithm, we must fix sequences σ_u for each $u \in [m]$. We will show later how suitable sequences σ_u can be found. First, let us determine the contents of table T_2.

The table T_2. Once the query algorithm is specified, there is a natural choice for the contents of T_2. Let $\mathsf{Error}_\Pi(S, u, r)$ denote the event "after storing the set S in its table, the protocol Π gives a wrong answer for the query 'Is u in S?' when it uses the random string r." For $S \in \binom{[m]}{\le n}$, let

$$R(S,u) \stackrel{\text{def}}{=} \{i : \mathsf{Error}_\Pi(S, u, \sigma_u(i))\} \text{ and}$$
$$R(S) \stackrel{\text{def}}{=} \bigcup_{u \in S} R(S,u).$$

Since we allow no error for query elements $u \in S$, $T_2(i)$ must be 1 for all $i \in R(S)$ when we store the set S in our tables. Let the remaining bits of T_2 be 0. This clearly ensures that $\Pr_i[\mathsf{Error}_{\hat{\Pi}}(S, u, i)] = 0$ for all $S \in \binom{[m]}{\le n}$ and $u \in S$. It remains only to

ensure that $\Pr_i[\mathsf{Error}_{\hat{\Pi}}(S,u,i)] \le n^{-\delta}$ when $u \notin S$. For this, we need to choose σ_u carefully.

Choosing σ_u. For $u \notin S$, we have $\mathsf{Error}_{\hat{\Pi}}(S,u,i) \equiv \mathsf{Error}_{\Pi}(S,u,\sigma_u(i)) \vee i \in R(S)$. Thus

$$\Pr_i[\mathsf{Error}_{\hat{\Pi}}(S,u,i)] \le \Pr_i[\mathsf{Error}_{\Pi}(S,u,\sigma_u(i))] + \Pr_i[i \in R(S)]. \tag{11}$$

To bound the first term on the right, we show

$$\forall u, S,\ |R(S,u)| \ \le\ n \log m; \tag{12}$$

to bound the second, we show

$$\forall S,\ |R(S)| \ \le\ n \log m. \tag{13}$$

Since $n \log m \le \frac{s'}{2n^\delta}$, using these bounds in (11), we obtain $\Pr_i[\mathsf{Error}_{\hat{\Pi}}(S,u,i)] \le n^{-\delta}$.

Thus we need to choose σ_u such that (12) and (13) hold. Let each σ_u be a sequence of s' randomly (uniformly and independently) chosen strings from $\{0,1\}^\ell$. Now

$$\Pr_{r \in \{0,1\}^\ell}[\mathsf{Error}_{\Pi}(S,u,r)] \ \le\ \frac{1}{n^2}.$$

Since σ_u is obtained by picking $s' = \lceil 2n^{1+\delta} \log m \rceil$ strings from $\{0,1\}^\ell$ independently, we get, using Chernoff's bound (see (10) above), that

$$\Pr_{\{\sigma_u\}}[|R(S,u)| > n \log m] \ \le\ \left(\frac{e/n^2}{1/(4n^\delta)}\right)^{n \log m} \ \le\ \left(\frac{4e}{n}\right)^{n \log m}.$$

(Note that $(n \log m)/s' \ge 1/(4n^\delta)$.) Thus

$$\Pr_{\{\sigma_u\}}\left[\exists u \in [m], S \in \binom{[m]}{\le n} |R(S,u)| > n \log m\right] \le m^{n+1} \left(\frac{4e}{n}\right)^{n \log m} < \frac{1}{2}.$$

Thus (12) holds with probability more than $\frac{1}{2}$.

To ensure (13), we first observe that, for each $i \in [s']$,

$$\Pr[i \in R(S)] \ \le\ \sum_{u \in S} \Pr[i \in R(S,u)] \ \le\ n \times \frac{1}{n^2} = \frac{1}{n}.$$

Furthermore, the events "$i \in R(S)$" are independent for different $i \in [s']$. Thus, using Chernoff's bound (see (10) above), we obtain

$$\Pr_{\{\sigma_u\}}[|R(S)| > n \log m] \ \le\ \left(\frac{e/n}{1/(4n^\delta)}\right)^{n \log m} \ \le\ \left(\frac{4e}{n^{1-\delta}}\right)^{n \log m},$$

and

$$\Pr_{\{\sigma_u\}}\left[\exists S \in \binom{[m]}{\le n} |R(S)| > n \log m\right] \ \le\ m^n \left(\frac{4e}{n}\right)^{n \log m} < \frac{1}{2}.$$

Thus (13) holds with probability more than $\frac{1}{2}$.

Since (12) and (13) both hold with probability more than $\frac{1}{2}$, they hold simultaneously for some choice $\{\sigma_u\}_{u \in [m]}$. We fix one such choice in our query algorithm. ∎

4. Lower bounds for randomized schemes. Consider a scheme that uses space s and just one bitprobe. In general, on receiving a query, the algorithm does one of three things based on the outcome of its coin tosses and the query element.

1. It decides to answer "Yes," regardless of what is stored in the table (which it may or may not read).
2. It decides to answer "No," regardless of what is stored in the table.
3. It computes, based on the coin tosses and the query element, an index $i \in [s]$ and
   (a) answers "Yes" if and only if the ith bit of the table is 1 or
   (b) answers "Yes" if and only if the ith bit of the table is 0.

It will be convenient if our query algorithm has the following *standard form*: it always reads some bit of the table and answers "Yes" if and only if it reads a 1. A scheme with a general query algorithm can be modified easily so that the new query algorithm is in standard form. This modification will roughly double the space required but will keep the error probability the same. Suppose the original algorithm used the table $T : [s] \to \{0,1\}$. The new algorithm will use a table $T' : [s+1] \times \{0,1\} \to \{0,1\}$, whose contents are defined by

$$T'(i,b) \stackrel{\text{def}}{=} \begin{cases} T(i) & \text{if } i \in [s] \text{ and } b = 1, \\ \neg T(i) & \text{if } i \in [s] \text{ and } b = 0, \\ b & \text{if } i = s+1. \end{cases}$$

The query algorithm is then modified as follows: in case 1 above, when the old algorithm always said "Yes," the new algorithm reads the bit $T'(s+1,1)$; in case 2, the new algorithm reads $T'(s+1,0)$; in case 3(a), it reads $T'(i,1)$; and, in case 3(b), it reads $T'(i,0)$. In all cases, the answer is "Yes" if and only if the bit read is 1.

Our lower bounds for randomized one-probe schemes are based on bounds for r-cover-free families.

DEFINITION 4.1. *A family of sets $\mathcal{F}$ is r-cover-free if, for $T_0, T_1, \ldots, T_r \in \mathcal{F}$ such that $T_0 \notin \{T_1, T_2, \ldots, T_r\}$, $T_0 \not\subseteq T_1 \cup \cdots \cup T_r$.*

THEOREM 10 (Füredi [17]). *If $\mathcal{F}$ is an r-cover-free family of sets and $r \le |\mathcal{F}|^{1/3}$, then*

$$\left| \bigcup_{T \in \mathcal{F}} T \right| \ge \frac{r^2}{4 \log r + O(1)} \log |\mathcal{F}|.$$

(Similar bounds have been shown in [8, 29, 7]*.)*

4.1. Randomized schemes with one-sided error.

Proof of Theorem 4. Suppose there is a randomized $(n, m, s, 1)$-scheme with negative one-sided error ϵ. As discussed earlier, this implies that there is a randomized $(n, m, 2s+2, 1)$-scheme with negative one-sided error ϵ, whose query algorithm is in standard form. Consider the bipartite graph $G = (U, V, E)$, where $U = [m]$, $V = [2s+2]$, and $(u, v) \in E$ if and only if location v is probed (with nonzero probability) on query "Is u in S?" In particular, on query "Is u in S?," the algorithm picks an element $v \in \Gamma(u)$ at random, according to some distribution D_u, and answers "Yes" if and only if there is a 1 in location v of the table.

Let $r \stackrel{\text{def}}{=} \lceil \frac{n}{\epsilon} \rceil - 1$; note that $r < \frac{n}{\epsilon}$.

CLAIM 4.2. *$\{\Gamma(u)\}_{u \in U}$ is an r-cover-free family.*

Proof. Suppose the claim is false. Then there exist distinct $u, u_1, u_2, \ldots, u_r \in U$ such that $\Gamma(u) \subseteq \bigcup_{i=1}^{r} \Gamma(u_i)$. Let S be a random subset of $\{u_1, u_2, \ldots, u_r\}$ of size n.

Then, for each $i \in \Gamma(u)$, $\Pr[i \in \Gamma(S)] \geq \frac{n}{r}$. For $T \subseteq V$, let $D_u(T) \stackrel{\text{def}}{=} \sum_{i \in T} D_u(i)$. By linearity of expectation,

$$\mathrm{E}[D_u(\Gamma(S))] \;=\; \sum_{i \in \Gamma(u)} D_u(i) \Pr[i \in \Gamma(S)] \;\geq \frac{n}{r} \sum_{i \in \Gamma(u)} D_u(i) \;=\; \frac{n}{r} \;>\; \epsilon.$$

Fix a choice for S with $D_u(\Gamma(S)) > \epsilon$. When S is stored, all locations in $\Gamma(S)$ must contain a 1 (because the error is negative one-sided). Then, on query "Is u in S?," the query algorithm answers "Yes" with probability more than ϵ, but the error allowed is at most ϵ. □

Claim 4.2 and Theorem 10 imply that, if $r \leq m^{1/3}$, then

$$|V| \;\geq\; \frac{r^2}{4 \log r + O(1)} \log m.$$

Thus $s = \Omega(\frac{n^2}{\epsilon^2 \log(n/\epsilon)} \log m)$. □

We next consider positive one-sided error and observe that bitvectors are optimal in this case.

THEOREM 11. *Let $\epsilon < 1$ and $m \geq 1$. Any randomized $(1, m, s, 1)$-scheme with positive one-sided error ϵ must have $s \geq m$.*

Proof. Since $\epsilon < 1$, for each $u \in [m]$, there must be a coin toss sequence r_u for which the query algorithm says "Yes" when the set $\{u\}$ is stored and the query "Is u in S?" is posed. In this case, the algorithm must probe some location of the table, for otherwise it would say "Yes" with nonzero probability even when the empty set is stored. Let $\ell_u \in [s]$ be the location probed, and let $b_u \in \{0, 1\}$ be the bit read. We claim that $\ell_u \neq \ell_{u'}$ for $u \neq u'$. For, suppose $u \neq u'$ and $\ell_u = \ell_{u'}$. We have two cases.

$b_u = b_{u'}$. Store $S = \{u\}$. On query "Is u' in S?" and coin toss sequence $r_{u'}$, the algorithm will answer "Yes," which is not allowed.

$b_i \neq b_j$. Store the empty set. Either on query "Is u in S?" with coin toss sequence r_u or on query "Is u' in S?" with coin toss sequence $r_{u'}$, the answer will be "Yes." However, when the empty set is stored, the answer should be "No" with probability 1 for all queries.

Thus $\ell_u \neq \ell_{u'}$ when $u \neq u'$, implying $s \geq m$. □

4.2. Randomized schemes with two-sided error. To prove the lower bound for randomized schemes with two-sided error, we need to use *upper bounds* on r-cover-free families together with the lower bound.

Proof of Theorem 2. Fix a randomized $(n, m, s, 1)$-scheme that answers queries with probability of error at most ϵ. We assume that the query algorithm is in standard form, and, as before, we model it using the bipartite graph (U, V, E), where $U = [m]$ and $V = [s]$: on query "Is u in S?," the algorithm probes a random location in $[s]$ according to a distribution D_u ($D_u(i) \neq 0$ if and only if $i \in \Gamma(u)$) and answers "Yes" if and only if the location contains a 1.

For our lower bound, we will need an r-cover-free family $\mathcal{F} \subseteq \binom{[m]}{\leq n}$, where $r = \lfloor \frac{1}{\epsilon} \rfloor$. We first present the argument assuming such a family; later we will obtain our lower bound by choosing a suitably large $\mathcal{F}$. For $S \in \mathcal{F}$, let $T_S \subseteq [s]$ be the set of locations of the table that contain a 1 when S is stored. Let $\ell = \lfloor \frac{1}{2\epsilon} \rfloor$. Since $\epsilon \leq \frac{1}{4}$, we have $\ell \geq 1$.

CLAIM 4.3. *$\{T_S : S \in \mathcal{F}\}$ is ℓ-cover-free.*

Proof. Suppose $T_{S_0} \subseteq T_{S_1} \cup T_{S_2} \cup \cdots \cup T_{S_\ell}$ for some $S_0, S_1, \ldots, S_\ell \in \mathcal{F}$, such that $S_0 \notin \{S_1, S_2, \ldots, S_\ell\}$. We will derive a contradiction.

Since $\mathcal{F}$ is r-cover-free and $r \geq \ell$, we have $S_0 \not\subseteq \bigcup_{i=1}^{\ell} S_i$; let $u \in S_0 - \sum_{i=1}^{\ell} S_i$. Since $u \in S_0$, $D_u(T_{S_0}) \geq 1 - \epsilon$. Thus $D_u(\bigcup_{i=1}^{\ell} T_{S_i}) \geq 1 - \epsilon$ and $D_u(T_{S_i}) \geq \frac{1-\epsilon}{\ell}$ for some $i \in [n]$. Fix one such i. Now, when the scheme stores the set S_i and receives the query "Is u in S_i?," it says "Yes" with probability at least $\frac{1-\epsilon}{\ell} \geq 2\epsilon(1-\epsilon) > \epsilon$. However, this is not possible. □

Using the above claim and Theorem 10, we obtain that, if $\ell \leq |\mathcal{F}|^{1/3}$, then

$$s \geq \frac{\ell^2}{4\log \ell + O(1)} \log |\mathcal{F}|. \tag{14}$$

To prove our lower bound, we need to find an r-cover-free family of large size. If $\epsilon \geq \frac{1}{n}$, we use Lemma 3.2 to obtain $\mathcal{F} \subseteq \binom{[m]}{n}$ of size at least

$$\frac{\binom{m}{\lceil \epsilon n \rceil}}{\binom{n}{\lceil \epsilon n \rceil}^2} \geq \left(\frac{m\epsilon}{e^2 n}\right)^{\epsilon n},$$

where the pairwise intersection of sets is of size at most $\lceil \epsilon n \rceil - 1$. Such a family is $\lfloor \frac{1}{\epsilon} \rfloor$-cover-free, for otherwise some pair of sets would intersect on at least $\lceil \frac{n}{\lfloor 1/\epsilon \rfloor} \rceil \geq \lceil \epsilon n \rceil$ elements. Then (14) gives us (using our assumption $n \leq m^{1/3}$)

$$s \geq \frac{(1/\epsilon)^2}{4\log(1/\epsilon) + O(1)} \log \left(\frac{m\epsilon}{e^2 n}\right)^{\epsilon n} = \Omega\left(\frac{n}{\epsilon \log(1/\epsilon)} \log m\right).$$

If $\epsilon < \frac{1}{n}$, we use the r-cover-free family $\binom{[m]}{1}$ and use (14) to obtain

$$s \geq \frac{(1/\epsilon)^2}{4\log(1/\epsilon) + O(1)} \log m = \Omega\left(\frac{n}{\epsilon \log(1/\epsilon)} \log m\right). \qquad \square$$

5. Deterministic schemes. We now show a time-space tradeoff result for deterministic schemes.

Proof of Theorem 6. Recall that the bitstring used to store the set $S \in \binom{[m]}{n}$ is called $\phi(S)$. Let

$$T_S \stackrel{\text{def}}{=} \{\langle \ell, \phi(S)(\ell) \rangle : \text{location } \ell \text{ of } \phi(S) \text{ is probed on query "Is } u \text{ in } S\text{?" for some } u \in S\}.$$

We now observe that the sets T_S have to be incomparable for different S. For, if $T_{S_1} \supseteq T_{S_2}$ for $S_1 \neq S_2$, store the set S_1, and ask the query "Is u in S_1?" for an element $u \in S_2 - S_1$. The scheme will err on this query, which is a contradiction. Now each T_S is a subset of size at most nt of the set $[s] \times \{0,1\}$. It follows, from the Lubell, Yamamoto, and Meshalkin (LYM) inequality (see, for example, Alon and Spencer [1, p. 183]), that $\binom{m}{n} \leq \max_{i \leq nt} \binom{2s}{i}$. □

We now study deterministic schemes when the number of probes allowed is small.

Proof of Theorem 7*, part* 1. We will obtain our deterministic scheme from a randomized $(m, n, O(tnm^{2/(t+1)}), 1)$-scheme with error probability *less* than $\frac{1}{2}$. The randomized scheme we use will be in the standard form: on query "Is u in S?," the query algorithm will pick a location randomly from a set $\Gamma(u)$ and say "Yes" if and only if a 1 is stored there. The size of $\Gamma(u)$ will be exactly $2t+1$. To obtain the deterministic scheme, we read all locations in $\Gamma(u)$ and say "Yes" if a majority (at least $t+1$) of them contains 1.

To construct the randomized scheme, we use the method of Theorem 1. Using calculations similar to those in the proof of Lemma 3.10, one can show that there is a graph $G = (U, V, E)$, where $U = [m]$ and $V = [s]$, such that $s = O(tnm^{2/(t+1)})$ and $|\Gamma_u| = 2t + 1$ for $u \in U$, such that $\{\Gamma(u)\}_{u \in U}$ has the $(n, \frac{t}{2t+1})$-intersection property. By Lemmas 3.6 and 3.4, it follows that there is an $(m, n, O(tnm^{2/(t+1)}), 1)$-scheme with error probability less than $\frac{1}{2}$. □

Proof of Theorem 7*, part* 2. Our adaptive scheme uses a combination of the FKS scheme and the bitvector scheme.

Storing scheme. Given a set T, do the following.

Step 1. Find a prime $p < n^2 \log m$ such that, if $x \neq y$, $x, y \in T$, then $x \bmod p \neq y \bmod p$. The fact that such a prime exists has been shown by Fredman, Komlós, and Szemerédi [16]. Store p using $O(\log n + \log\log m)$ bits.

Step 2. Now the set $T \bmod p$ consists of n elements, each less than $n^2 \log m$. Store this set using the FKS data structure. This requires space $O(n(\log n + \log\log m))$.

Step 3. For every $x \in T$, do the following: divide the string x into t blocks $B_1, \ldots, B_t$, each of size $\log m/t$. For each such block B_i, construct a look-up table of size $2^{\log m/t}$ with a 1 in the index given by B_i. Space required is $nt2^{\log m/t}$ bits.

Space used by the storing scheme is $O(n(\log n + \log\log m) + nt2^{\log m/t})$.

Query scheme. Given a query u, do the following.

Step 1. Read the prime p. This requires $O(\log n + \log\log m)$ bitprobes.

Step 2. Find $u \bmod p$. Now check if there is an element y in T such that $u \bmod p = y \bmod p$ using the FKS structure. This requires $O(\log n + \log\log m)$ bitprobes.

Step 3. If there is no such y, say "No."

Step 4. If there is such a y, retrieve a pointer to it using $O(\log n)$ bitprobes. Then divide u into t blocks, and check if $x = y$ block by block. This requires t bitprobes.

The time used by the query scheme is $O(\log n + \log\log m) + t$. □

Finally, we consider deterministic schemes that use two bitprobes. We show that two nonadaptive probes do not help even for $n = 2$. We also show that adaptiveness helps for $n = 2$.

Proof of Theorem 8*, part* 1. There are 16 different functions mapping $\{0,1\}^2$ to $\{0,1\}$. We will divide them into 3 classes.

1. Degenerate functions. These are the functions depending on at most one variable. There are 6 such functions, namely, 0, 1, u, y, $\bar{x}$, $\bar{y}$.
2. Inflexible functions. These are the functions f, so that there is a value of $f(x, y)$ which determines the value of u as well as the value of y. For instance, $x \wedge y = 1 \Rightarrow x = y = 1$. There are 8 such functions, namely, $x \wedge y$, $\bar{x} \wedge y$, $x \wedge \bar{y}$, $\bar{x} \wedge \bar{y}$, $x \vee y$, $\bar{x} \vee y$, $x \vee \bar{y}$, $\bar{x} \vee \bar{y}$.
3. Flexible functions. These are functions which are neither degenerate nor inflexible. There are 2 such functions, namely, $x \oplus y$ and $\bar{x} \oplus y$.

Suppose the theorem fails. Fix m at the smallest value for which this happens. Note that the theorem is trivially true for $m = 1$. Let U be a universe of size m. The corresponding scheme associates with each $z \in U$ two locations u_z and v_z in $\{1, \ldots, s\}$ and a Boolean function $f_z : \{0,1\}^2 \to \{0,1\}$. Clearly, if, for some z, f_z is constant, the scheme is incorrect. Now assume that, for some z, f_z is degenerate. Assume that

it depends on its first variable. We can get a scheme for a universe of size $m-1$ by fixing the value of u_z, yielding a structure of size $s-1$. This is a contradiction. Thus we can assume that all functions f_z are inflexible or flexible. Now assume, to the contrary, that $s < m$. Let the multigraph $G = (V, E)$ be given by $V = \{1, \ldots, s\}$ and $E = \{e_z = (u_z, v_z) : z \in U\}$. Let an edge e_z be denoted flexible if the corresponding function f_z is flexible, and let it be denoted inflexible otherwise. As $|E| = m$, $|V| = s$, and $s < m$, G contains a cycle C. (As G is a multigraph, it may be the case that C consists of exactly two identical edges.)

Now there are two cases.

1. *The cycle C contains only flexible edges.* Let Z be the elements corresponding to the edges in C. Fix any setting τ of the bits in the data structure. Each edge e_z on the cycle corresponds to an element $z \in Z$, and we can see if $z \in S$ by adding, modulo 2, the setting $\tau(u_z)$ of the bit u_z and the setting $\tau(v_z)$ of the bit v_z or the setting $\overline{\tau(u_z)}$ of the bit $\overline{u}_z$ and the setting $\tau(v_z)$ of the bit v_z. This quantity summed over all z in Z uniquely determines the parity of the number of elements of $S \cap Z$. But this sum is zero or one depending on the number of z of the second type. As the sum determines the parity of $|Z \cap S|$, either $\emptyset$ or $\{z\}$, where z is any element of Z, has no valid representation.
2. *The cycle C contains an inflexible edge.* Let this edge be e_z. There is a choice of $z \in S$ or $z \notin S$ which makes only one configuration of the values of the bits u_z, v_z possible. Fix this choice. Let us assume that it is $z \in S$. (The case $z \notin S$ is similar.) Now let z_1 be the other edge on C, adjacent to v_z. Having already fixed $v_z = u_{z_1}$, there are two possibilities: having thus fixed u_{z_1} either determines whether $z_1 \in S$ or it does not. If it does, either $\{z\}$ or $\{z, z_1\}$ has no valid representation, and we are done. If it does not, we fix the setting of v_{z_1} in the unique way so that $z_1 \notin S$. Now let z_2 be the other edge on C, adjacent to v_{z_1}. Having already fixed $v_{z_1} = u_{z_2}$, there are two possibilities: fixing u_{z_2} in this manner either determines whether or not $z_2 \in S$ or it does not. If it does, either $\{z\}$ or $\{z, z_1\}$ has no valid representation, and we are done. If it does not, we fix the setting of v_{z_2} in the unique way so that $z_2 \notin S$. Now let z_3 be the other edge on C, adjacent to v_{z_2}, etc. Thus, working our way around the cycle, we finally come to an edge e_{z_l}, adjacent to u_z. Thus we have fixed u_{z_l} as well as $v_{z_l} = u_z$, and we conclude that either $\{z\}$ or $\{z, z_l\}$ has no valid representation.

Thus we have arrived at a contradiction, and we conclude $s \geq m$. □

It is easy to get a three probe nonadaptive scheme for sets of size at most 2 using space $O(m^{1/2})$. Such a scheme can be obtained, for instance, from the explicit one-sided error scheme by setting the parameters appropriately.

Proof of Theorem 8*, part* 2. The structure of the proof is as follows. We will first define a certain combinatorial object. We then show that the existence of this object implies the two-probe scheme claimed in the theorem. Then we show, using the probabilistic method (in particular, the *alteration* technique), that the desired object exists.

Thus let $U = \{1, 2, \ldots, m\}$. Let an *augmented s-partition* of U be a system consisting of

1. a partition of U into classes $U_1, U_2, \ldots, U_s$,
2. a set system $M_1, M_2, \ldots, M_s$ on $\{1, 2, \ldots, s\}$,
3. a set system $N_1, N_2, \ldots, N_s$ on $\{1, 2, \ldots, s\}$,
4. a family of one-to-one maps $f_i : U_i \to M_i$,

5. a family of one-to-one maps $g_i : U_i \to N_i$,

with the following property.

For all $U_i \neq U_j$, $x \in U_i$, and $y \in U_j$, we have *either* $f_i(x) \notin M_j$ and $f_j(y) \notin M_i$ *or* $g_i(x) \notin N_j$ and $g_j(y) \notin N_i$.

CLAIM 5.1. *If an augmented s-partition exists, there is an adaptive two-probe scheme using $3s$ bits solving the membership problem.*

Proof.

Storing scheme. The scheme will make use of three tables T, T_0, and T_1, each of size s. Given a set $S = \{x, y\}$, there are three cases.

1. *u and y are in the same class, U_i.* We let $T[i] = 0$ and $T[j] = 1$ for all $j \neq i$. We let $T_0[f_i(x)] = T_0[f_i(y)] = 1$ and $T_0[j] = 0$ for all $j \notin \{f_i(x), f_i(y)\}$. We let $T_1[j] = 0$ for all j.
2. *u and y are in different classes, $x \in U_i$ and $y \in U_j$. Furthermore, $f_i(x) \notin M_j$ and $f_j(y) \notin M_i$.* We let $T[i] = T[j] = 0$ and $T[k] = 1$ for all $k \notin \{i, j\}$. We let $T_0[f_i(x)] = T_0[f_j(y)] = 1$ and $T_0[j] = 0$ for all $j \notin \{f_i(x), f_j(y)\}$. We let $T_1[j] = 0$ for all j.
3. *x and y are in different classes, $x \in U_i$ and $y \in U_j$. Furthermore, $g_i(x) \notin N_j$ and $g_j(y) \notin N_i$.* We let $T[i] = T[j] = 1$ and $T[k] = 0$ for all $k \notin \{i, j\}$. We let $T_1[g_i(x)] = T_1[g_j(y)] = 1$ and $T_0[j] = 0$ for all $j \notin \{g_i(x), g_j(y)\}$. We let $T_0[j] = 0$ for all j.

Query scheme. Given a query q in class U_i, do the following:

1. Read $T[i]$.
2. If it is 0, read $T_0[f_i(q)]$. If 1 is seen, say "Yes." Otherwise, say "No."
3. If it is 1, read $T_1[g_i(q)]$. If 1 is seen, say "Yes." Otherwise, say "No."

It is easily seen that the augmented partition property ensures that the scheme is correct. □

The theorem now follows from the following claim. □

CLAIM 5.2. *For any m, an augmented $O(m^{3/4})$-partition exists.*

Proof. Let $U' = \{1, 2, \ldots, 2m\}$. We shall construct a "blemished" augmented partition of U' and then alter it so that it has the right property.

We assume, without loss of generality, that $s = 2(2m)^{3/4}$ is an integer and divides $2m$. Thus we can partition U' evenly into $U_1, U_2, \ldots, U_s$, each of size $\frac{1}{2}(2m)^{1/4}$. Now we choose M_i and N_i independently at random from among all subsets of size $\frac{1}{2}(2m)^{1/4}$ of $\{1, 2, \ldots, s\}$. We choose f_i and g_i independently at random from all one-to-one functions.

Now consider fixed $x \in U_i$, $y \in U_j$ for $i \neq j$. We will bound the probability that the property fails for this particular pair. Clearly,

$$\Pr[f_i(x) \in M_j] = \frac{|M_j|}{s} = \frac{\frac{1}{2}(2m)^{1/4}}{2(2m)^{3/4}} = \frac{1}{4(2m)^{1/2}}.$$

By the union bound,

$$\Pr[f_i(x) \in M_j \vee f_j(y) \in M_i] \leq \frac{1}{2(2m)^{1/2}},$$

and, similarly,

$$\Pr[g_i(x) \in N_j \vee g_j(y) \in N_i] \leq \frac{1}{2(2m)^{1/2}}.$$

As the two events are independent, we get

$$\Pr[(f_i(x) \in M_j \vee f_j(y) \in M_i) \wedge (g_i(x) \in N_j \vee g_j(y) \in N_i)] \leq \frac{1}{8m}.$$

Therefore, the expected number of pairs (x, y) for which the desired property does not hold is less than $\binom{2m}{2}\frac{1}{8m} < m$. Hence there is a choice of M_i, N_i, f_i, g_i, such that the number of bad pairs (x, y) is at most m. Now, from each such bad pair, remove one of the elements. We remove at most m elements, yielding a universe of size at least m, as desired. For the remaining elements, the property holds, and we are done. □

The bound of $O(m^{3/4})$ in the above claim has been improved to $O(m^{2/3})$ by Radhakrishnan, Raman, and Rao (see [28]).

Acknowledgments. We are grateful to Venkatesh Raman, S. Srinivasa Rao, and Amnon Ta-Shma for their comments on this work, and to the referees for their help in improving the paper.

REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley-Interscience, New York, 1992.
[2] V. Arvind and J. Köbler, *On resource-bounded measure and pseudorandomness*, in Proceedings of the 17th Foundations of Software Technology and Theoretical Computer Science Conference, Lecture Notes in Comput. Sci. 1346, Springer-Verlag, New York, 1997, pp. 235–249.
[3] B. Bollobás, *Random Graphs*, Academic Press, New York, 1985.
[4] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 21–31.
[5] A. Brodnik and J. I. Munro, *Membership in constant time and minimum space*, in Proceedings of the 2nd Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 855, Springer-Verlag, New York, 1994, pp. 72–81.
[6] A. Brodnik and J. I. Munro, *Membership in constant time and almost-minimum space*, SIAM J. Comput., 28 (1999), pp. 1627–1640.
[7] S. Chaudhuri and J. Radhakrishnan, *Deterministic restrictions in circuit complexity*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 30–36.
[8] A. G. Dyachkov and V. V. Rykov, *Bounds on the length of disjunctive codes*, Problemy Peredachi Informatsii, 18 (1982), pp. 7–13 (in Russian).
[9] P. Elias and R. A. Flower, *The complexity of some simple retrieval problems*, J. ACM, 22 (1975), pp. 367–379.
[10] P. Erdős, P. Frankl, and Z. Füredi, *Families of finite sets in which no set is covered by the union of r others*, Israel J. Math., 51 (1985), pp. 79–89.
[11] A. Fiat and M. Naor, *Implicit $O(1)$ probe search*, SIAM J. Comput., 22 (1993), pp. 1–10.
[12] A. Fiat, M. Naor, J. P. Schmidt, and A. Siegel, *Non-oblivious hashing*, J. ACM, 31 (1992), pp. 764–782.
[13] G. S. Frandsen, P. B. Miltersen, and S. Skyum, *Dynamic word problems*, J. ACM, 44 (1997), pp. 257–271.
[14] M. L. Fredman, *Observations on the complexity of generating quasi-Gray codes*, SIAM J. Comput., 7 (1978), pp. 134–146.
[15] M. L. Fredman, *The complexity of maintaining an array and computing its partial sums*, J. ACM, 29 (1982), pp. 250–260.
[16] M. L. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.
[17] Z. Füredi, *On r-cover-free families*, J. Combin. Theory Ser. A, 73 (1996), pp. 172–173.
[18] C. T. M. Jacobs and P. van Emde Boas, *Two results on tables*, Inform. Process. Lett., 22 (1986), pp. 43–48.
[19] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, *Efficient search for approximate nearest neighbor in high-dimensional spaces*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 614–623.

[20] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 604–613.

[21] J. Katz and L. Trevisan, *On the efficiency of local decoding procedures for error-correcting codes*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 80–86.

[22] P. B. Miltersen, *The bitprobe complexity measure revisited*, in Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzberg, Germany, 1993, pp. 662–671.

[23] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson, *On data structures and asymmetric communication complexity*, J. Comput. System Sci., 57 (1998), pp. 37–49.

[24] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.

[25] R. Pagh, *Low redundancy in static dictionaries with $O(1)$ worst case lookup time*, in Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 1644, Springer-Verlag, Berlin, 1999, pp. 595–604.

[26] N. Nisan, *Pseudorandom generators for space-bounded computation*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 204–212.

[27] N. Nisan and A. Wigderson, *Hardness vs. randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[28] J. Radhakrishnan, V. Raman, and S. S. Rao, *Explicit deterministic construction for membership in the bitprobe model*, in Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2161, Springer-Verlag, New York, 2001, pp. 290–299.

[29] M. Ruszinkó, *On the upper bound of the size of r-cover-free families*, J. Combin. Theory Ser. A, 66 (1984), pp. 302–310.

[30] R. Raz, O. Reingold, and S. Vadhan, *Extracting all the randomness and reducing the error in Trevisan's extractors*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 149–158.

[31] M. Sudan, L. Trevisan, and S. Vadhan, *Pseudorandom generators without the XOR lemma*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 537–546.

[32] A. Ta-Shma, *Explicit One-Probe Storing Schemes Using Universal Extractors*, manuscript, 2001.

[33] A. Ta-Shma, C. Umans, and D. Zuckerman, *Loss-less condensers, unbalanced expanders, and extractors*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 143–152.

[34] A. Ta-Shma and D. Zuckerman, *Extractor codes*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 193–199.

[35] L. Trevisan, *Construction of extractors using pseudo-random generators*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 141–148.

[36] A. C. Yao and F. F. Yao, *Dictionary look-up with one error*, J. Algorithms, 25 (1997), pp. 194–202.

[37] A. C. C. Yao, *Should tables be sorted?*, J. ACM, 28 (1981), pp. 615–628.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1745–1760

ON THE BOUNDARY COMPLEXITY OF THE UNION OF FAT TRIANGLES*

JÁNOS PACH† AND GÁBOR TARDOS‡

Abstract. A triangle is said to be δ-*fat* if its smallest angle is at least $\delta > 0$. A connected component of the complement of the union of a family of triangles is called a *hole*. It is shown that any family of n δ-fat triangles in the plane determines at most $O\left(\frac{n}{\delta}\log\frac{2}{\delta}\right)$ holes. This improves on some earlier bounds of Efrat, Rote, Sharir, and Matoušek, et al. Solving a problem of Agarwal and Bern, we also give a general upper bound for the number of holes determined by n triangles in the plane with given angles. As a corollary, we obtain improved upper bounds for the boundary complexity of the union of fat polygons in the plane, which, in turn, leads to better upper bounds for the running times of some known algorithms for motion planning, for finding a separator line for a set of segments, etc.

Key words. holes, boundary complexity, fat objects

AMS subject classifications. 52C45, 52B55

PII. S0097539700382169

1. Introduction and main results.

Many basic problems in computational geometry related to motion planning [SS83, SS83b, SS89, SS90], range searching [K97, GJ97], computer graphics [AK94], and geographic information systems (GIS) [BK97] lead to questions about the complexity of the boundary of the union of certain geometric objects. When the boundary is simple, these problems can usually be solved more efficiently [GS93]. This was the motivation behind a lot of research during the past fifteen years, establishing upper bounds for the complexity (or, equivalently, for the description size) of the union of various objects.

Perhaps the first results of this kind were the following. Given n simply connected regions in the plane, any two of which share at most two (resp., at most three) boundary points, the boundary of their union consists of at most $6n-12$ (resp., at most $n\alpha(n)$) *simple arcs*, i.e., connected pieces whose interior belongs to the boundary of a single region [KL86] (resp., [EG89]; here $\alpha(n)$ denotes the extremely slowly growing inverse of Ackermann's function). In some sense, this result is the best possible: if two regions are allowed to cross at four boundary points, then the boundary of their union may consist of $\Omega(n^2)$ simple arcs. Indeed, consider n very "skinny" pairwise crossing triangles, no three of which have a point in common.

However, it was discovered by Matoušek et al. [MP94] that, if we restrict how skinny our triangles can be, we can still establish a nearly linear upper bound on the complexity of their union. For any $\delta > 0$, a triangle is said to be δ-*fat* if each of its angles is at least δ. (The reciprocal of the smallest angle of a triangle is often called

*Received by the editors December 18, 2000; accepted for publication (in revised form) April 18, 2002; published electronically September 5, 2002. An extended abstract of this paper appeared in [PT00].

http://www.siam.org/journals/sicomp/31-6/38216.html

†Courant Institute, New York University, 251 Mercer St., New York, NY 10012 (pach@cims.nyu.edu). The work of this author was supported by NSF grant CCR-97-321018, OTKA T-020914, and AKP 2000-78 2.1.

‡Rényi Institute, Hungarian Academy of Sciences, H-1364 Budapest, POB 127, Hungary (tardos@renyi.hu). The work of this author was supported by OTKA grants T030059, T029255, FKFP 0607/1999, and AKP 2000-78 2.1.

its *aspect ratio*.) It turned out that, for any fixed $\delta > 0$, the boundary of the union of n δ-fat triangles in the plane consists of at most $n \log\log n/\delta^3$ simple arcs.

The *boundary complexity* of the union of a family $\mathcal{T}$ of triangles (or simply connected regions) is defined as the number of simple arcs along $\mathrm{Bd}(\cup\mathcal{T})$, the boundary of the union of $\mathcal{T}$. A connected component of the complement of $\cup\mathcal{T}$ is called a *hole*. The heart of the argument in [MP94] was the following statement.

THEOREM 0 (Matoušek et al.). *Any family of n δ-fat triangles in the plane determines $O\left(n/\delta^3\right)$ holes.*

The concept of δ-fatness, as well as the above theorem, has been extended to arbitrary polygons by van Kreveld [K98]. For other extensions and generalizations, see [SH93], [S94], [ES97], [EK98], and [E99].

For *wedges* (i.e., cones) in place of triangles, a somewhat better upper bound was found by Efrat, Rote, and Sharir [ER93]. They proved that the number of holes determined by n wedges in the plane (and the boundary complexity of their union) is $O\left(\frac{n}{\delta^2}\log\frac{2}{\delta}\right)$.

Our first theorem generalizes and strengthens this result.

THEOREM 1. *Any family of n δ-fat triangles in the plane determines $O\left(\frac{n}{\delta}\log\frac{2}{\delta}\right)$ holes. This bound is tight up to the logarithmic factor.*

Theorem 1 can be used to establish a more general upper bound for the number of holes determined by a family of triangles with given angles.

THEOREM 2. *Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a family of $n > 1$ triangles in the plane, and let α_i denote the smallest angle of T_i $(1 \le i \le n)$. Suppose $0 < \alpha_1 \le \alpha_2 \le \cdots \le \alpha_n$, and let $k \le n$ be the largest integer satisfying $\sum_{i=1}^{k} \alpha_i < \pi$.*

Then $\mathcal{T}$ determines $O(nk\log k)$ holes. Furthermore, there exists a family $\mathcal{T}' = \{T_1', \ldots, T_n'\}$, where T_i' and T_i are congruent for all i, and $\mathcal{T}'$ determines $\Omega(nk)$ holes.

Of course, the same result applies to wedges, provided that their angles are separated from π. Moreover, in this case, an almost identical upper bound holds for the *boundary complexity* of the union.

THEOREM 3. *Let $\mathcal{T}$ be a family of n wedges in the plane with angles $0 < \alpha_1 \le \cdots \le \alpha_n < \pi$. Let $k \le n$ be the largest integer satisfying $\sum_{i=1}^{k} \alpha_i < \pi$.*

If $k \ge 2$, then the boundary complexity of $\cup\mathcal{T}$ is $O(nk\log k)$. Furthermore, there exists a family of n wedges with angles $\alpha_1, \ldots, \alpha_n$ which determines $\Omega\left((\pi - \alpha_n)nk\right)$ holes.

Notice that Theorem 3 bounds the boundary complexity instead of the number of holes. The bound in Theorem 2 for the number of holes in families of triangles cannot be extended to boundary complexity as there are families of equilateral triangles (for which $k = 2$) with superlinear boundary complexities (at least $\Omega(n\alpha(n))$; cf. [WS88]).

In some applications, e.g., the overlay of triangulated environments in GIS, we cannot assume that *all* of the participating triangles are fat (have bounded aspect ratios). However, very often *most* of them satisfy this condition [ZS99]. To deal with these situations, Agarwal and Bern [AB99] asked whether Theorem 0 can be generalized as follows. Let $\mathcal{T}$ be a family of n triangles in the plane, whose *average* aspect ratio is bounded by a constant. That is, $\sum_{i=1}^{n} \frac{1}{\alpha_i} = O(n)$, where α_i denotes the smallest angle of the ith triangle. Is it then true that $\mathcal{T}$ determines only $O(n)$ (or nearly a linear number of) holes? Theorems 2 and 3 answer this question in the negative. Indeed, let

$$\alpha_i = \begin{cases} \frac{1}{\sqrt{n}} & \text{if } 1 \le i \le \sqrt{n}, \\ 1 & \text{if } \sqrt{n} < i \le n. \end{cases}$$

Then we have $\sum_{i=1}^{n} \frac{1}{\alpha_i} < 2n$, but, according to the second statement of Theorem 2, the number of holes can be as large as $\Omega(n^{3/2})$. This is true even for wedges (see Theorem 3). The first statement of Theorem 2 shows that this bound is tight, apart from the logarithmic factor.

In the case when some of the wedges have angles very close to π, Theorem 3 is not sufficiently tight. A more careful analysis can account more precisely for the contribution of the convex wedges with angles close to π.

THEOREM 4. *Let $\mathcal{T}$ be a family of n wedges in the plane with angles $0 < \alpha_1 \leq \cdots \leq \alpha_n < 2\pi$. Let $l \in [0, n]$ be the largest integer satisfying $\alpha_l < 3\pi/4$, and let $m \in [l, n]$ be the largest integer satisfying $\alpha_m < \pi$. (We set $l = 0$ if $\alpha_1 \geq 3\pi/4$, and we set $l = m = 0$ if $\alpha_1 \geq \pi$.) Let k be the largest integer with $\sum_{j=1}^{k} \alpha_j < \pi$, and, for any $1 \leq i \leq m$, let $k_i \in [0, i)$ be the largest integer such that $\sum_{j=1}^{k_i} \alpha_j < \pi - \alpha_i$.*

Then the boundary complexity of $\cup\mathcal{T}$ is $O(n + \sum_{i=1}^{l} k_i \log k_i + \sum_{i=l+1}^{m} k_i) = O(n + lk \log k + \sum_{i=l+1}^{m} k_i)$, where the sum is taken over all i with $k_i \neq 0$. Furthermore, there is a family of n wedges with angles $\alpha_1, \ldots, \alpha_n$ which determines $\sum_{i=1}^{m'} k_i + m' + 1$ holes, where $m' = \min\{m, n-1\}$.

The rest of the paper is organized as follows. The proofs of Theorems 1, 2–3, and 4 are presented in sections 2–3, 4, and 5, respectively. The last section contains some combinatorial and algorithmic consequences of the main results.

2. Reduction to rhombs and assignment of holes. By a *polygon* we mean a simply connected (bounded or unbounded) region in the plane, whose boundary consists of a finite number of straight-line segments and possibly two half-lines. A family of polygons is said to be in *general position* if no three lines supporting different sides of the polygons pass through the same point. We say that a *point* is *incident* to a hole H if it lies on the boundary of H.

Given a family $\mathcal{P}$ of polygons in the plane, let $h(\mathcal{P})$ and $H(\mathcal{P})$ denote the number of holes determined by $\mathcal{P}$ and the minimum number of nonoverlapping convex polygons the union of these holes can be partitioned into, respectively. Furthermore, let $c(\mathcal{P})$ stand for the number of concave angles of $\Re^2 \setminus \cup\mathcal{P}$, the union of the holes.

LEMMA 2.1. *For any family $\mathcal{P}$ of polygons in the plane, we have*

$$h(\mathcal{P}) \leq H(\mathcal{P}) \leq h(\mathcal{P}) + c(\mathcal{P}).$$

Proof. The lower bound on $H(\mathcal{P})$ follows from the fact that, to cover each hole, we need at least one convex set. To establish the upper bound, we show that every hole with k concave vertices can be partitioned into $k+1$ convex sets. In the case when $k = 0$, the hole itself is convex. For $k > 0$, it is enough to observe that the total number of concave vertices decreases by cutting the hole into two along the angular bisector at a concave vertex. ∎

LEMMA 2.2. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two families of polygons in the plane such that $\cup\mathcal{P}' \subseteq \cup\mathcal{P}$ and any segment connecting two points of $\Re^2 \setminus \cup\mathcal{P}$ which intersects $\cup\mathcal{P}$ also intersects $\cup\mathcal{P}'$.*

Then $H(\mathcal{P}) \leq H(\mathcal{P}')$.

Proof. For any partition of $\Re^2 \setminus \cup\mathcal{P}'$ into a family $\mathcal{C}$ of convex sets, $\{C \setminus \cup\mathcal{P} \mid C \in \mathcal{C}\}$ forms a partition of $\Re^2 \setminus \cup\mathcal{P}$ into convex sets. Note that we needed the condition in the lemma to guarantee that each member of the latter family is convex. ∎

For the rest of this section, we fix an angle $\delta \leq \pi/3$ and set $k := \lceil 2\pi/\delta \rceil$. Clearly, we have $k \geq 6$. Fix any straight line ℓ_0 in the plane. We say that a *segment* (line,

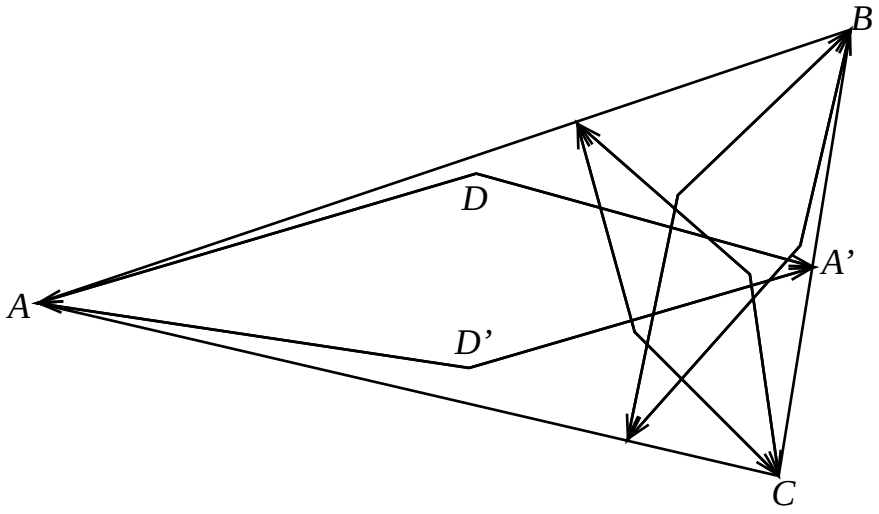


FIG. 1. *Substituting three canonical rhombs for a triangle.*

half-line) is *canonical* if the angle between its supporting line and ℓ_0 is an integer multiple of π/k. A *rhomb* is called *canonical* if (i) all of its sides are canonical, and (ii) two of its angles are equal to π/k.

Next we show that Theorem 1 can be reduced to the following theorem.

THEOREM 2.3. *Any family $\mathcal{R}$ of n canonical rhombs in the plane determines $O(nk\log k)$ holes.*

Proof of Theorem 1 (using Theorem 2.3). Let $\mathcal{T}$ be a family of n δ-fat triangles in the plane. Consider a vertex A of a triangle $ABC \in \mathcal{T}$. By the choice of k, there are at least two canonical half-lines emanating from A, whose initial segments belong to ABC. Therefore, we can pick a point A' on the segment BC and two other points, D and D', in ABC such that $R_A = ADA'D'$ is a canonical rhomb whose angles at A and A' are equal to π/k. Let R_A denote such a rhomb. Similarly, we can define two other canonical rhombs, R_B and R_C, within the triangle ABC. (See Figure 1.)

Let $\mathcal{T}'$ denote the family obtained from $\mathcal{T}$ by replacing every triangle $ABC \in \mathcal{T}$ by the three corresponding canonical rhombs, R_A, R_B, and R_C. By Theorem 2.3, $\mathcal{T}'$ determines $O(nk\log k)$ holes, i.e., $h(\mathcal{T}') = O(nk\log k)$. Now Lemmas 2.1 and 2.2 imply that

$$h(\mathcal{T}) \le H(\mathcal{T}) \le H(\mathcal{T}') \le h(\mathcal{T}') + c(\mathcal{T}').$$

In other words, the number of holes determined by $\mathcal{T}$ may be larger than the number of holes determined by $\mathcal{T}'$, but the difference cannot exceed the total number of concave corners (vertices) in all holes determined by $\mathcal{T}'$. However, every such corner corresponds to a vertex of one of the $3n$ rhombs defined above, so the difference is at most $12n$. □

In the rest of this section and the next section, we establish Theorem 2.3. We may and will assume without loss of generality that the rhombs in $\mathcal{R}$ are in general position. We use the term *edge* only for the sides of the rhombs in $\mathcal{R}$. We call two edges *homothetic* if they are corresponding sides of two homothetic rhombs. We orient every edge e of a rhomb toward its vertex of angle π/k. This vertex is called the *apex* of e. The subsegments of e inherit the orientation of e.

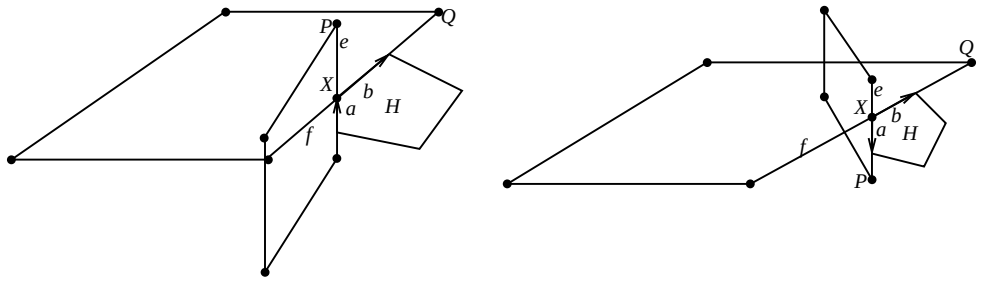


FIG. 2.

Let e be an edge, and let S be a homothecy class of edges not containing e. There may be several holes whose boundaries contain a piece of e next to a piece of some element of S. The first and last such holes along e are said to be *extreme*. Since there are $4n$ edges in at most $4k$ homothecy classes, the number of extreme holes is at most $32kn$. We call the nonextreme holes *intermediate*. Clearly, it is sufficient to bound the number of intermediate holes. As every hole incident to a vertex of a rhomb is extreme, the intermediate holes are convex.

Let H be an intermediate hole. Consider two consecutive segments, a and b, on the boundary of H, belonging to the edges e and f, respectively, and denote their common endpoint by X. Let P and Q denote the apices of e and f, respectively. Suppose that b is oriented away from X. We say that H is a *hole assigned to* e if we have $|X-P| \le |X-Q|$. The distance $|X-P|$ is called the *depth of* H *along* e or, if it leads to no confusion, simply the *depth of* H. We say that H is an *in-hole* or an *out-hole* assigned to e, depending on whether a is oriented toward X or away from X. See Figure 2.

LEMMA 2.4. *Every intermediate hole is assigned to at least one edge.*

Proof. Let H be an intermediate hole. If the segments bounding H are not cyclically oriented, we find two consecutive segments, both oriented away from their common endpoint. In this case, H is an out-hole assigned to one of the edges containing these two segments.

Suppose that the segments forming the boundary of H are cyclically oriented. Let $X_1, X_2, \dots, X_k = X_0$ denote the vertices of H in this cyclic order, and let P_i denote the apex of the edge containing $X_{i-1}X_i$ $(1 \le i \le k)$. Set $P_{k+1} := P_1$ and $X_{k+1} := X_1$. For every $1 \le i \le k$, if H is not an in-hole assigned to the edge containing $X_{i-1}X_i$, then we have

$$|X_i - P_i| > |X_i - P_{i+1}| = |X_i - X_{i+1}| + |X_{i+1} - P_{i+1}|.$$

Summing up these inequalities, we obtain

$$\sum_{i=1}^{k} |X_i - P_i| > \sum_{i=1}^{k} |X_i - P_i| + \mathrm{Per}(H),$$

where $\mathrm{Per}(H)$ stands for the perimeter of H. This contradiction proves that H is an in-hole assigned to an edge supporting one of its sides. □

Next we show that the depths of the in-holes along an edge are $\Omega(1/k)$ apart in a logarithmic scale, and the same is true for the depths of the out-holes. (The depth of

an in-hole can be arbitrarily close to the depth of an out-hole though.) More precisely, we have the following lemma.

LEMMA 2.5. *Let H be an in-hole (out-hole) assigned to an edge e, whose depth is d. Then the depth of no other hole assigned to e is between d and $\left(1-\frac{1}{k}\right)d$ (resp., between d and $\left(1+\frac{1}{k}\right)d$).*

Proof. Let a and b be two consecutive segments of the boundary of H causing H to be assigned to e. Let X be their common endpoint, let f be the edge containing b, and let P and Q be the apices of e and f, respectively. Clearly, a is on e oriented toward X, and b is oriented away from X. The depth of H is $d = |X-P|$. Since H is not an extremal hole, e must cut through the rhomb belonging to the edge f. The length of the piece of XP covered by this rhomb is at least

$$|X-Q|\sin\frac{\pi}{k} \geq |X-P|\sin\frac{\pi}{k} > \frac{d}{k}.$$

Thus the depth of no hole along e can belong to the interval $((1-1/k)d, d)$. The corresponding statement for out-holes can be proved similarly. □

3. Base points. Theorem 2.3 would immediately follow from Lemma 2.5 if we could show that the ratio of the largest and smallest depths of an in-hole (and out-hole) along the same edge is bounded by a polynomial of k. It is not hard to see that this holds for wedges rather than rhombs (and this can be used to give a direct proof of Theorem 3), but the general statement is false. We prove instead that the depths of the intermediate holes (in-holes and out-holes) assigned to a given edge fall into a small number of short intervals.

To formulate our result precisely, we need some preparation. We assign at most *three* so-called *base points* to each edge, according to the following definition.

DEFINITION 3.1 (base points). *Let $e = NP$ be an edge of a rhomb $\Delta = NPSU \in \mathcal{R}$, and let e be oriented toward P. Let Y denote the point of e closest to N that does not belong to the interior of any member of $\mathcal{R}$. Let both P and Y be assigned to e as* base points*. Further, let h denote the oriented half-line (ray) starting at N and passing through U. Consider all edges homothetic to e that either intersect h beyond U or intersect both h and the edge SP. If there is no such edge, then no other base point is assigned to e. Otherwise, let e' denote the edge with this property that intersects h closest to N. If e' intersects h beyond U, then we say that e' is* far from *e and set $Y' := e' \cap h$. If e' intersects the segment PS, then we say that e' is* close to *e and set $Y' := e' \cap PS$. In both cases, Y' is the third base point assigned to e which will be referred to as the* base point forced by *e. Note that this third point is* not *on e. (See Figure 3.)*

The depth of a base point *Y sitting on an edge e is defined as the distance between Y and the apex of e.*

Using the above notation, no point of the open interval YN can be incident to a hole. Thus the depth of a hole assigned to e cannot exceed the depth of the base point Y on e. Consequently, the depth of each hole assigned to e is between the depths of some pair of consecutive base points on e. Although there may be many base points along the same edge, the total number of base points is at most $12n$.

LEMMA 3.2. *If there are two consecutive base points on an edge e with depths d_1 and $d_2 > k^2 d_1$, then there is at most one intermediate hole assigned to e with depth belonging to the interval $(d_1, d_2/k^2)$.*

Proof. Let $e = NP$ be an edge oriented toward P. Choose two consecutive base points on e with depths $d_1 < d_2$, respectively. Let H be an (intermediate) hole

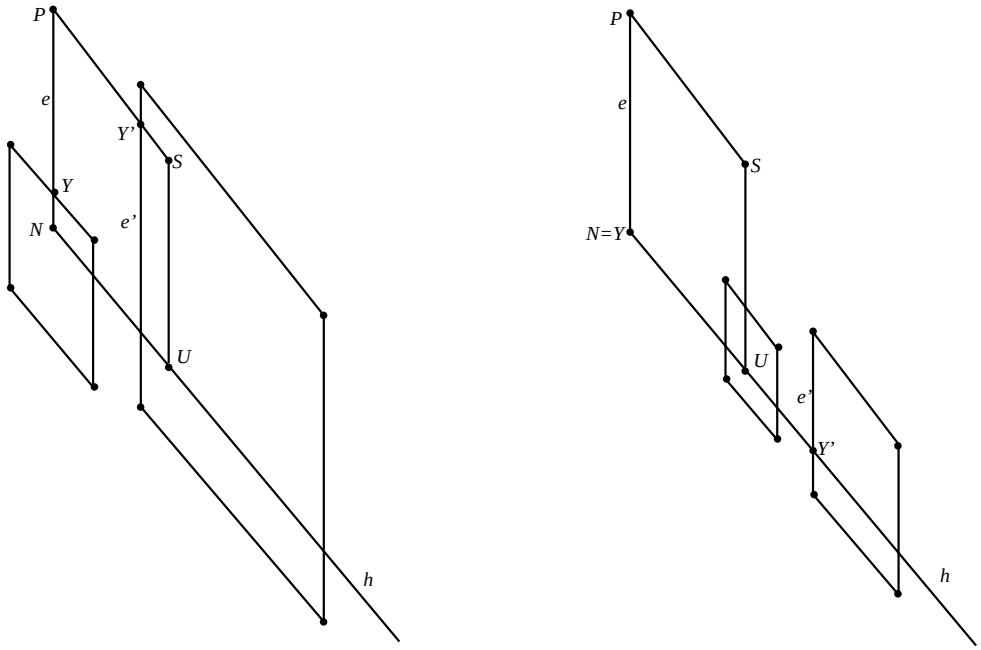


FIG. 3.

assigned to e with depth d satisfying $d > d_1$. We prove that, for every other hole assigned to e, whose depth d' is larger than d, $d' \geq d_2/k^2$ holds.

As before, let a and b be the two consecutive segments on the boundary of H causing H to be assigned to e. Let the segment b belong to the edge f with apex Q. Let X be the common endpoint of a and b. Clearly, a is on e, and b is oriented away from X.

The proof is based on the following claim.

CLAIM. *There is a base point Y on e of depth at least the depth of H such that the angle PYQ is at least $\frac{\pi}{2k}$.*

Proof. We assume without loss of generality that e is vertical and that its upper endpoint is P. As b is oriented away from X, and H is not an extreme hole, we find another edge e_0, homothetic to e, which intersects XQ. Consider the sequence of edges $e_0, e_1, e_2, \ldots, e_t$, where e_i is the edge homothetic to e, containing the base point forced by e_{i-1} $(1 \leq i \leq t)$. The last edge of this sequence, e_t, forces no base point. (See Figure 4.) Let e_i be the edge of the rhomb $\Delta_i = N_i P_i S_i U_i \in \mathcal{R}$ with $e_i = N_i P_i$ oriented toward P_i, and let h_i be the half-line starting at N_i and passing through U_i $(0 \leq i \leq t)$. Notice that h_i intersects e_{i+1} for $i < t$, and denote this point of intersection by X_{i+1}. The intersection of f and e_0 is denoted by X_0. Let Π be the directed polygonal path $QX_0N_0X_1N_1 \ldots X_tN_t$ followed by the half-line h_t. Since all segments of Π (except maybe QX_0) are pointing downward and are at least as steep as f, X must lie above Π.

We distinguish two cases.

Case I. $e = e_{i_0}$ for some $1 \leq i_0 \leq t$. The point $X_{i_0} \in \Pi$ lies below X on the same

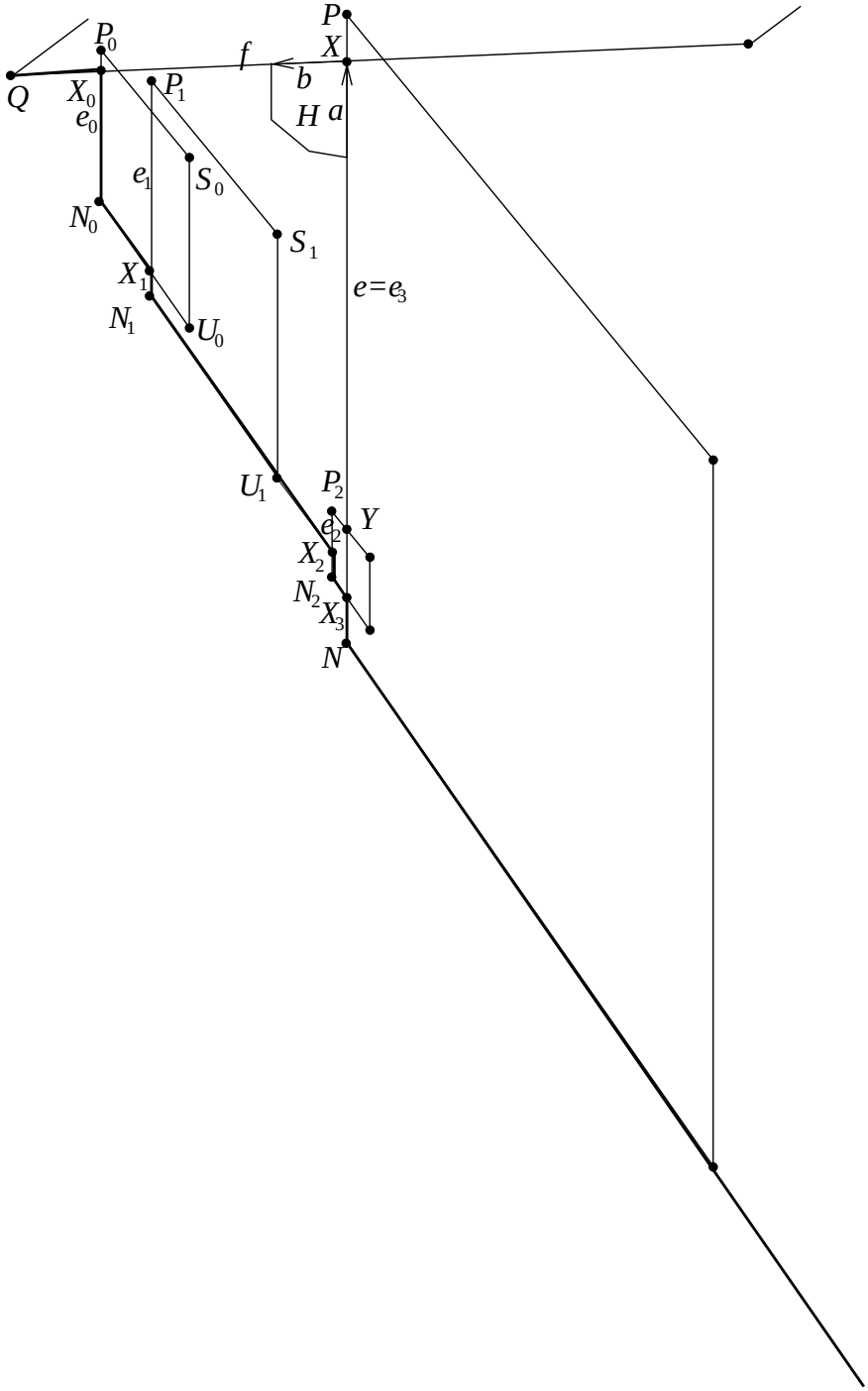


FIG. 4. *The construction of the path* Π *(the bold path).*

vertical edge e. Let Y denote the base point on e forced by e_{i_0-1}. If e is far from e_{i_0-1}, the point Y coincides with X_{i_0}, so it is below X. If e is close to e_{i_0-1}, the open interval $X_{i_0}Y$ belongs to the interior of Δ_{i_0-1}, and the point X incident to H cannot lie on this interval. Therefore, in this case, Y cannot be above X either.

For $0 \le i \le t$, denote by σ_i the strip between the parallel lines containing P_iS_i and N_iU_i. Notice that, if e_i is close to e_{i-1} $(1 \le i \le t)$, then σ_i contains σ_{i-1}.

Let $1 \le j_1 < j_2 < \cdots < j_s$ be the sequence of indices $1 \le j \le i_0$ for which e_j is far from e_{j-1}, and let $j_0 = 0$. For $1 \le i \le s$, the portion of Π between $X_{j_{i-1}}$ and X_{j_i} is contained in σ_{j_i-1}. Thus $X_{j_{i-1}}$ is below the line $P_{j_i-1}U_{j_i-1}$, while X_{j_i} is above the same line. This implies that the angle $X_{j_{i-1}}X_{j_i}P_{j_i}$ is larger than $\pi/(2k)$. If $j_s = i_0$, we have $X_{j_s} = X_{i_0} = Y$. Otherwise, $X_{j_s} \in \sigma_{i_0-1}$, while Y is on the upper boundary of this strip, so the angle $X_{j_s}YP$ is at least π/k. Since we have bounded the slope of each portion of Π, combining these bounds, it follows that the angle PYQ is at least $\pi/(2k)$. This completes the proof of the claim in case I.

Case II. $e \notin \{e_i : 1 \le i \le t\}$. The path Π now cannot cross e. Otherwise, let X' be this point of intersection, and suppose X' lies on the half-line h_i $(i \le t)$. When determining the base point forced by e_i, we must consider the edge e, since the interval $X'P$ contains the point X incident to the hole H and thus cannot be entirely covered by Δ_i. So we have that either $e = e_{i+1}$ or X_{i+1} lies between N_i and X'. Both are contradictory to our assumptions—the latter since X' is not on Π in this case.

So e must lie entirely above Π. Let l denote the line containing e, and let Z be the intersection point $l \cap \Pi$. Suppose that Z belongs to the nonvertical segment of Π starting at the point N_{i_0} for some $i_0 \le t$.

Let j be the largest index between (and including) 1 and i_0 for which e_j is far from e_{j-1}. If there is no such index, set $j = 0$. It follows in exactly the same way as in the previous case that the angle QX_jP_j is at least $\pi/(2k)$. Let Y denote the point of e closest to N that does not belong to the interior of any rhomb in $\mathcal{R}$. Recall that, according to Definition 3.1, Y is a base point, and notice that it does not lie above X.

If l intersects Δ_{i_0}, the point X_j is below the line $P_{i_0}S_{i_0}$, while Y' is on this line or above it. Therefore, in this case, the angle X_jYP is at least π/k. If l does not intersect Δ_{i_0}, consider the portion of Π between X_j to Z, and notice that it lies in the strip between the lines $P_{i_0}S_{i_0}$ and $N_{i_0}U_{i_0}$. Thus X_j is below the line $P_{i_0}U_{i_0}$, but Z (and thus Y) is above the same line. This implies that, in this case, the angle X_jYP is also at least $\pi/(2k)$. Combining this with the same lower bound for the angle QX_jP_j, we obtain that the angle PYQ is at least $\pi/(2k)$. This completes the proof of the claim. □

Now we finish the proof of Lemma 3.2:

The depth of the base point Y is larger than d_1; thus we have $|P-Y| \ge d_2$. Using the law of sines for the triangle QYX, we obtain $|Y - X| \le |Q - X|/\sin\frac{\pi}{2k}$. Since H was assigned to e, we have that $|P - X| \le |Q - X|$. Therefore,

$$\begin{aligned} d_2 &\le |P - Y| = |P - X| + |X - Y| \\ &\le \left(1 + \frac{1}{\sin\frac{\pi}{2k}}\right)|Q - X| \le k|Q - X|. \end{aligned}$$

Note that the rhomb in $\mathcal{R}$ belonging to the edge f covers an interval of e, whose length is at least $|Q - X|\sin(\pi/k) \ge d_2/k^2$. As X is an endpoint of this interval, any hole assigned to e, whose depth d' is larger than the depth $d = |P - X|$ of H, must satisfy the inequality $d' > d_2/k^2$; hence the lemma is true. □

Proof of Theorem 2.3. According to Lemma 2.5, the depths of the intermediate holes assigned to an edge are separated from each other by $\Omega(1/k)$ in a logarithmic scale. By Lemma 3.2, if an edge has $c+1$ base points, then the depths of all holes assigned to this edge, except for at most c of them, fit in c intervals, each of length $2\log k$ in logarithmic scale. Thus this edge has $O(ck\log k)$ holes assigned to it. Thus, by Lemma 2.4, the total number of intermediate holes is $O(nk\log k)$. Since the number of extreme holes is $O(nk)$, Theorem 2.3 follows. □

4. Generalizations—proofs of Theorems 2 and 3.

Proof of Theorem 2. Obviously, the triangles $T_{k+1},\dots,T_n$ are $\pi/(k+1)$-fat. By Theorem 1, they determine $O(nk\log k)$ holes. By adding the first k triangles, we increase the number of intersection points and, therefore, the number of holes by $O(nk)$.

As for the construction, let $l=\lfloor k/2\rfloor$, and arrange $T_1',\dots,T_l'$ so that any two intersect in a single point v, and all of them are contained in a right angle wedge W with apex v. This is possible because we have $\sum_{i=1}^{l}\alpha_i<\pi/2$. Let ε be the distance of v from the nearest other vertex of $T_1',\dots,T_l'$. We place T_i' for $i>l$ so that

- T_i' meets both rays bounding W but does not contain v;
- $T_i'\cap W$ is contained in the ball of radius ε around v;
- the sets $T_i'\cap W$ are pairwise disjoint for $i>l$.

All of these conditions can be satisfied by placing the triangles one by one so that we put a vertex of the next triangle, corresponding to an acute angle, sufficiently close to v. The obtained configuration has $(l-1)(n-l+1)$ holes inside W. This quantity is $\Omega(nk)$, as required, unless $l=1$. In the latter case, we arrange the triangles T_i' $(i>1)$ so that all of them have a point in common outside T_1', and their intersections with T_1' are distinct single points. These triangles determine at least $n-1$ holes. □

We continue with Theorem 3. (Notice that it is a simple special case of Theorem 4.)

Proof of Theorem 3. First we prove the $O(nk\log k)$ bound for the *number of holes* determined by the convex wedges in $\mathcal{T}$. As we have indicated before, the direct proof of this bound along the lines described in section 2 is simpler than the proof of Theorem 1. However, at this point, it is more convenient to deduce it from Theorem 2. Let h be the number of holes determined by the wedges. First, we split each wedge with an obtuse angle into two congruent wedges. Then we replace each wedge by a triangle, intersecting it with a half-plane that contains all intersection points between the boundaries of the original wedges. We make sure that all new angles introduced exceed $\pi/4$. Following this procedure, we obtain a family of at most $2n$ triangles that determine at least $h-n+1$ holes. The value k for this new family (as defined in Theorem 2) is at most 3 larger than the corresponding value for the original family of wedges. Applying Theorem 2 to the triangles, we obtain the desired bound for wedges.

To prove the same upper bound for the *boundary complexity*, notice that $\alpha_i\ge\pi/(k+1)$ for $i>k$. As in the proof of Theorem 2, we can disregard the first k wedges because their contribution to the boundary complexity is at most $4kn$. We proceed as in [MP94]. We partition the remaining wedges into $2k+2$ classes so that all wedges belonging to the ith class contain in their interior a half-line having a $2\pi i/(2k+2)$ positive angle from a reference direction. Now every vertex of the union of all wedges in a given class is either the apex of a wedge or is the last vertex along one of the open half-lines bounding the wedges. Thus the boundary complexity of this union is linear in the number of wedges in the class. Using the combination

lemma of [EG90] (see also Lemma 2.1 in [MP94]) to merge the classes in a binary tree-like fashion, it follows that the boundary complexity of the union of all wedges in all classes is $O(nk\log^2 k)$. To get rid of the extra $\log k$ factor, we consider the family of wedges $\mathcal{T}'$ we obtain at an intermediate step through the combination process. It is the union of some j of the original $2k+2$ families. We can make sure that these are j consecutive families. Applying an affine transformation, if necessary, we can achieve that the angle of every wedge belonging to these families is $\Omega(1/j)$. Since such a transformation does not change the number of holes, we obtain the better bound $O(mj\log j)$ for the number of holes determined by $\mathcal{T}'$, where m is the number of wedges in $\mathcal{T}'$. Using the combination lemma with this better bound, we conclude that the boundary complexity of $\mathcal{T}$ is $O(nk\log k)$.

To verify the last statement of Theorem 3, we use almost the same construction as in the proof of Theorem 2. The only difference is that now we have to start with a wedge W whose angle is smaller than $\pi-\alpha_n$; otherwise, no wedge of angle α_n could intersect it in the required manner. Let $l\in[0,n/2]$ be the largest integer satisfying $\sum_{i=1}^{l}\alpha_i<\pi-\alpha_n$. Clearly, we have $l\geq\min\{\lfloor(1-\alpha_n/\pi)k\rfloor,n/2\}$. Select l wedges in W with angles $\alpha_1,\dots,\alpha_l$ such that the intersection of any two is the apex v of W. Then choose $n-l$ wedges of angles $\alpha_{l+1},\dots,\alpha_n$ such that

- each of them intersects both boundary half-lines of W;
- none of them contains v;
- their intersections with W are pairwise disjoint bounded sets.

The resulting family determines $(l-1)(n-l+1)$ holes in W. This is $\Omega((\pi-\alpha_n)nk)$ unless $l\leq 1$. In the latter case, we use a trivial construction similar to the one described at the end of the proof of Theorem 2: we pick $n-1$ wedges that intersect the remaining wedge in distinct single points. This family determines at least $n-1$ holes. □

5. Wedges of angles close to π—Theorem 4. This section is devoted to the proof of Theorem 4.

First we establish the upper bound $O(B)$ for the *number of holes* in $\mathcal{T}$, where $B=m+lk\log k+\sum_{i=l+1}^{m}k_i$. The same bound on the *boundary complexity* then follows directly from the combination lemma of [EG90]: by Theorem 3, and the boundary complexity of the subfamily consisting of the l smallest wedges in $\mathcal{T}$ is $O(lk\log k)$, and the boundary complexity of the subfamily consisting of the next $m-l$ wedges of $\mathcal{T}$ is $O(m)$, while the boundary complexity of the family of the $n-m$ largest wedges of $\mathcal{T}$ (whose angles are at least π) is clearly $O(m)$, as it determines a single convex hole.

We proceed as in the proof of Theorem 1, but now we have to deal with different angles.

Fix a reference direction, and say that a wedge is *small* if its angle is $\pi/2^s$ for some integer $s\geq 2$ and if the angles between its boundary rays and the reference direction are integer multiples of $\pi/2^s$. A wedge is *large* if its angle is $\pi-2\pi/2^s$ for some integer $s\geq 2$ and if the angles between its boundary rays and the reference direction are integer multiples of $\pi/2^s$.

Let W_i be the wedge in $\mathcal{T}$ of angle α_i, and let P_i be its apex. For $i\leq l$, define W_i' to be the maximal small wedge with apex P_i that is contained in W_i. For $l<i\leq m$, let W_i' be the maximal large wedge with apex P_i that is contained in W_i. Let $\mathcal{T}'=\{W_i'|1\leq i\leq m\}$. By Lemmas 2.1 and 2.2, we have

$$h(\mathcal{T})\leq H(\mathcal{T})\leq H(\mathcal{T}')\leq h(\mathcal{T}')+m.$$

Thus, when passing from $\mathcal{T}$ to $\mathcal{T}'$, the number of holes cannot decrease by more than m.

Let α_i' denote the angle of the wedge W_i' ($i \le m$), and let k' and k_i' be defined for $\mathcal{T}'$ in exactly the same way as k and k_i were defined for $\mathcal{T}$ ($1 \le i \le m$). Notice that $\alpha_i/4 < \alpha_i' \le \alpha_i$, so $k' \le 4k+3$. Furthermore, we have $\pi - \alpha_i' < 4(\pi - \alpha_i)$, which implies that $k_i' \le 16k_i + 15$. Thus it is sufficient to prove the desired upper bound on the number of holes determined by the modified family $\mathcal{T}'$. For notational convenience, from now on, we assume that $\mathcal{T}' = \mathcal{T}$, i.e., that $\mathcal{T}$ consists of l small and $m-l$ large wedges.

We use the terms *wedge*, *apex*, *ray*, and *hole* only for the wedges in $\mathcal{T}$, their apices, their rays, and the holes determined by them. Assume without loss of generality that no three rays have a point in common. We say that two rays are *homothetic* if they are corresponding sides of two wedges that are translates of each other.

A triplet (X, r_1, r_2) is called a *vertex* if X is the intersection point of two distinct rays, r_1 and r_2, and it lies on the boundary of a hole. Our plan is to define several special types of vertices, to bound the number of vertices of each type separately, and finally to bound the number of holes by showing that each hole has a point on its boundary that appears in a vertex of some special type. A vertex (X, r_1, r_2) is said to be *small* (*large*) if both of the wedges supporting r_1 and r_2 are small (resp., large) wedges. The number of small and large vertices can be bounded by the boundary complexity of the family consisting of all small or all large wedges in $\mathcal{T}$, and, by Theorem 3, we have the following claim.

CLAIM 5.1. *There are $O(lk \log k)$ small and $O(m)$ large vertices.*

In what follows, when we consider a vertex (X, r_1, r_2), the wedges supporting r_1 and r_2 will be denoted by W_1 and W_2, respectively. The apices of W_1 and W_2 will be denoted by P_1 and P_2, respectively. If a vertex is neither small nor large, then it is said to be *hybrid*. For a hybrid vertex (X, r_1, r_2), one of W_1 and W_2 is small, and the other is large. In this case, let i and j denote the indices of the corresponding small and large elements of $\mathcal{T}$, respectively, with angles $\alpha_i = \pi/2^s$ and $\alpha_j = \pi - 2\pi/2^t$, respectively, where $1 \le i \le l$, $l+1 \le j \le m$, $s \ge 2$, and $t \ge 2$.

A vertex (X, r_1, r_2) is called *strongly extremal* if it is the first or last vertex along r_1 or r_2. Obviously, we have the following claim.

CLAIM 5.2. *There are at most $4m$ strongly extremal vertices.*

A hybrid vertex (X, r_1, r_2) is said to be *blunt* if $s < t$. Considering the canonical properties of the wedges in $\mathcal{T}$, we see that, if a blunt vertex (X, r_1, r_2) is not strongly extremal, then the boundaries of the wedges meet in three vertices, and X is the middle one (along either boundary). Furthermore, if W_1 is large, we find that r_2 is perpendicular to the angular bisector of W_1 (see Figure 5).

We claim that, along a ray bounding a large vertex, there is at most one blunt vertex that is not strongly extremal. Suppose, for contradiction, that (X, r_1, r_2) and (X, r_1, r_2') are both blunt and not strongly extremal and W_1 is a large wedge with apex P_1. We may also assume that X belongs to the interval $X'P_1$. Here the ray r_2' is parallel to r_2, so the wedges W_1 and W_2 together cover the part of r_2' on one side of X' (see Figure 5). Therefore, (X', r_1, r_2') must be strongly extremal, which is a contradiction. Therefore, the number of blunt vertices that are not strongly extremal is at most $2m$, which implies the following claim.

CLAIM 5.3. *The total number of blunt vertices is $O(m)$.*

A hybrid vertex (X, r_1, r_2) is said to be *sharp* if $s > t + \log k$.

CLAIM 5.4. *The number of sharp vertices is $O(\sum_{j=l+1}^{m} k_j)$.*

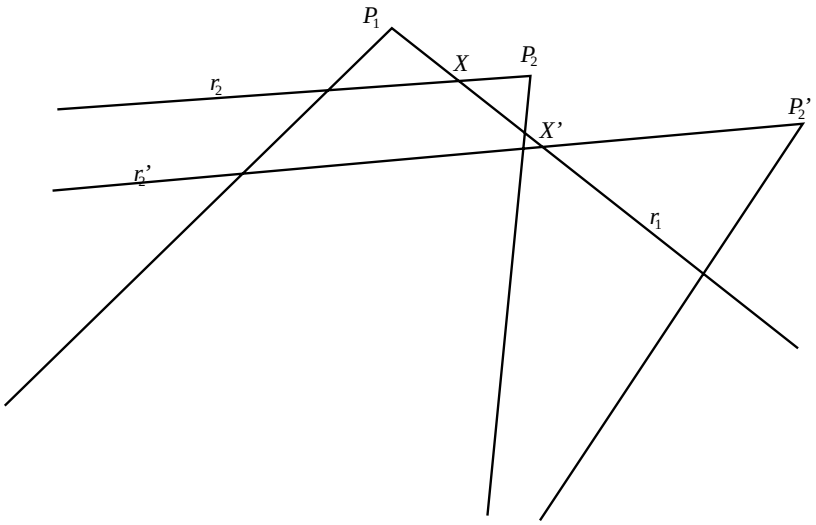


FIG. 5. *Blunt vertices.*

Proof. Let (X, r_1, r_2) be sharp. We have $i \le k_j$, for otherwise none of the angles of the first $k_j + 1$ wedges in $\mathcal{T}$ would exceed $\pi/2^s < \pi/2^{t+\log k}$, and their sum would be less than $\pi - \alpha_j = 2\pi/2^t$, contradicting the definition of k_j. Thus there are at most $O(k_j)$ sharp vertices involving the jth wedge in $\mathcal{T}$ $(j > l)$, and Claim 5.4 follows. ∎

A vertex (X, r_1, r_2) is called *extremal* if it is the first or the last vertex along r_1 among all vertices of the form (X', r_1, r_2') with r_2' being homothetic to r_2.

CLAIM 5.5. *The number of extremal vertices is* $O(B)$.

Proof. It is sufficient to establish this bound for those hybrid extremal vertices which are neither strongly extremal nor blunt nor sharp. Fix such a vertex (X, r_1, r_2). We distinguish two cases.

Case i. W_1 is small, and W_2 is large. For a given small wedge W_1 (which determines s), there are at most $\log k + 1$ possible values for t, without the vertex being sharp or blunt. For a given t, there are at most four homothecy classes possible for W_2 without the vertex being strongly extremal. Thus each ray of a small wedge is involved in at most $8 \log k + 8$ extremal vertices satisfying the condition of case i. Therefore, the total number of extremal vertices of this type is $O(l \log k)$.

Case ii. W_1 is large, and W_2 is small. Fix W_1 to be the jth wedge in $\mathcal{T}$. This determines the value of t. For a given value $s > t$, there are less than 2^{s-t+2} possible homothecy classes for W_2 without the vertex being strongly extremal. Notice that, if z among these classes for all $s > t$ are nonempty, then the sum of $\lceil z/2 \rceil$ of the smallest angles is less than $\pi - \alpha_j = 2\pi/2^t$. This implies that $z = O(k_j)$. The total number of extremal vertices involving r_1 that satisfy the condition in case ii is $O(k_j)$, and the total number of all extremal vertices of this type is $O(\sum_{i=l+1}^{m} k_i)$. ∎

A vertex (X, r_1, r_2) is called *covered* if the interval XP_1 is at least as long as the interval XP_2 and P_1 and W_2 lie on different sides of r_2.

CLAIM 5.6. *The number of covered vertices is* $O(B)$.

Proof. It is enough to consider those hybrid covered vertices that are neither extremal nor sharp nor blunt. Let (X, r_1, r_2) be such a covered vertex. As (X, r_1, r_2) is not extremal, there exists another vertex (X', r_1, r_2') for which X' belongs to the interval XP_1 and the ray r_2' is homothetic to r_2. Here the wedge W_2', whose boundary

ray is r_2', covers all of r_2 except an interval P_2Q. Our goal is to show that W_1 covers a sufficiently large subinterval XY of P_2Q. Considering these intervals XY for different covered vertices involving r_2, we find that they cover any point at most twice. Therefore, any lower bound on the lengths of these intervals yields an upper bound on their number. As in the proof of the previous claim, we distinguish two cases.

Case i. W_1 is small, and W_2 is large. By the relative position of the wedges W_1, W_2, and W_2', we have $|P_2Q| = O(|XY|/2^{s-t})$. This implies that r_2 appears in $O(k_j+1)$ covered vertices that satisfy the condition of case i. Therefore, the total number of covered vertices of this type is $O(m + \sum_{j=l+1}^{m} k_j)$.

Case ii. W_2 is small, and W_1 is large. Here we have to use that (X, r_1, r_2) is not sharp so that $s \le t + \log k$. As in case i, one can show that $|P_2Q| = O(k|XY|)$. Hence r_2 appears in $O(k)$ covered vertices satisfying the condition of case ii, and the total number of covered vertices of this type is $O(lk)$. □

The proof of the upper bound in Theorem 4 can now be completed by showing that each hole determined by $\mathcal{T}$ has a point on its boundary that appears as the leading term of a strongly extremal or a covered vertex. Indeed, if no such strongly extremal vertex exists, then the hole must be a (bounded) convex polygon. Consider the orientation of the edges inherited from the rays oriented toward their apices. If it is not *cyclic*, we find a vertex X with two outgoing edges. Obviously, either (X, r_1, r_2) or (X, r_2, r_1) is covered, where r_1 and r_2 are the two rays containing X. We deal with the cyclically oriented case in exactly the same way as in the proof of Theorem 1: assuming that no vertex around the hole is covered, we obtain several inequalities, whose sum gives a contradiction. This concludes the proof of the upper bound in Theorem 4 on the number of holes as well as on the boundary complexity of $\mathcal{T}$.

It remains to describe a construction for the lower bound. Choose an $\epsilon > 0$ such that $\epsilon < (\pi - \alpha_i - \sum_{j=1}^{k_i} \alpha_j)/n$ for all i, and let ℓ_0 be a fixed horizontal line. By the *direction* of a half-line h from a point P on ℓ_0, we mean the angle between the half-lines h and the part of ℓ_0 to the right of P. We place the wedges W_i with angles α_i $(i = 1, \dots, m')$ one by one, according to the following rules.

1. The apex of W_i is on ℓ_0 sufficiently to the right so that all intersection points of boundaries of wedges already placed are outside W_i and to the left of it.
2. W_i is above ℓ_0.
3. The direction of the right ray of W_i is ϵ larger than the direction of the left ray of W_{k_i}, or it is simply ϵ if $k_i = 0$.

By the choice of k_i and ϵ, all of the above requirements can be satisfied. The value $H = H(\{W_1, \dots, W_{m'}\})$ of the resulting family (see the beginning of section 2 for the definition) is the bound in Theorem 4 for the number of holes. For $i > m'$, we place a wedge of angle α_i such that it does not contain any intersection point between boundaries of W_j for $j \le m'$, but one of its boundary rays is on ℓ_0 and contains the apices of all W_j $(j \le m')$.

6. Concluding remarks and applications. As in [MP94], Theorems 1 and 2 yield the following upper bounds for the boundary complexity of a family of triangles.

COROLLARY 6.1. *The boundary complexity of any family of n δ-fat triangles in the plane is $O(\frac{n}{\delta}(\log\log n \log\frac{2}{\delta} + \log^2\frac{2}{\delta}))$. Moreover, the boundary of the union can be computed in time $O(\frac{n\log n}{\delta}(\log\log n \log\frac{2}{\delta} + \log^2\frac{2}{\delta}))$.*

COROLLARY 6.2. *Let $\mathcal{T} = \{T_1, \dots, T_n\}$ be a family of $n > 1$ triangles in the plane, and let α_i denote the smallest angle of T_i $(1 \le i \le n)$. Suppose $0 < \alpha_1 \le \alpha_2 \le \cdots \le \alpha_n$, and let $k \le n$ be the largest integer such that $\sum_{i=1}^{k} \alpha_i < \pi$.*

Then the boundary complexity of $\mathcal{T}$ cannot exceed $O(nk(\log\log n \log k + \log^2 k))$.

Plugging Theorem 3 into the analysis of the running time of the algorithm described in [ER93], we obtain the following two results.

COROLLARY 6.3. *The union of a family of n δ-fat wedges in the plane can be computed in time $O\left(\frac{n}{\delta}\log^2\frac{2}{\delta} + n\log n\right)$.*

A line ℓ is called a *separator* for a family $\mathcal{S}$ of pairwise disjoint segments in the plane if ℓ avoids all members of $\mathcal{S}$ and there is at least one member of $\mathcal{S}$ on both of its sides.

COROLLARY 6.4. *Given a family $\mathcal{S}$ of n line segments in the plane such that the ratio between the length of the shortest segment in $\mathcal{S}$ and the diameter of $\cup\mathcal{S}$ is at least $\delta > 0$, there is an algorithm which determines whether $\mathcal{S}$ admits a separator, and finds one if it exists, in $O\left(\frac{n}{\delta}\log\frac{n}{\delta}\log^2\frac{1}{\delta}\right)$ time and $O\left(\frac{n}{\delta}\log^2\frac{1}{\delta}\right)$ space.*

Van Kreveld [K98] extended the definition of fatness to a (not necessarily convex) simple polygon. He proved that every δ-fat simple polygon of k vertices can be covered by $O(k)$ δ-fat triangles, and such a covering can be constructed in $O(k\log k)$ time. Therefore, Theorem 1 generalizes to δ-fat simple polygons whose total number of sides is n.

Efrat [E99] introduced another generalization of the notion of fatness to compact connected regions of "constant description complexity" depending on two real parameters. He established an upper bound on the boundary complexity of a system of "fat" objects according to this definition. The dependence of his bounds on the parameters can be improved by using Theorem 1 instead of the results in [MP94].

REFERENCES

[AB99] P. K. AGARWAL AND M. BERN, Open problem raised at International Conference on Discrete and Computational Geometry (Centro Stefano Franscini, ETH Zürich), Monte Verita, Switzerland, 1999.

[AK94] P. K. AGARWAL, M. J. KATZ, AND M. SHARIR, *Computing depth orders and related problems*, in Algorithm Theory—SWAT '94, Lecture Notes in Comput. Sci. 824, Springer-Verlag, Berlin, 1994, pp. 1–12.

[AF92] H. ALT, R. FLEISCHER, M. KAUFMANN, K. MEHLHORN, S. NÄHER, S. SHIRRA, AND C. UHRIG, *Approximate motion planning and the complexity of the boundary of the union of simple geometric figures*, Algorithmica, 8 (1992), pp. 391–406.

[BK97] M. DE BERG, M. KATZ, F. VAN DER STAPPEN, AND J. VLEUGELS, *Realistic input models for geometric algorithms*, in Proceedings of 13th Annual ACM Symposium on Computational Geometry, ACM, New York, 1997, pp. 294–303.

[EG89] H. EDELSBRUNNER, L. GUIBAS, J. HERSHBERGER, J. PACH, R. POLLACK, R. SEIDEL, M. SHARIR, AND J. SNOEYINK, *On arrangements of Jordan arcs with three intersections per pair*, Discrete Comput. Geom., 4 (1989), pp. 523–539.

[EG90] H. EDELSBRUNNER, L. GUIBAS, AND M. SHARIR, *The complexity and the construction of many faces in arrangements of lines and of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.

[E99] A. EFRAT, *The complexity of the union of (α,β)-covered objects*, in Proceedings of the 15th Annual ACM Symposium on Computational Geometry, ACM, New York, 1999, pp. 134–142.

[EK98] A. EFRAT AND M. KATZ, *On the union of κ-curved objects*, Comput. Geom., 14 (1999), pp. 241–254.

[EK97] A. EFRAT, M. KATZ, F. NIELSEN, AND M. SHARIR, *Dynamic data structures for fat objects and their applications*, Comput. Geom., 15 (2000), pp. 215–227.

[ER93] A. EFRAT, G. ROTE, AND M. SHARIR, *On the union of fat wedges and separating a collection of segments by a line*, Comput. Geom., 3 (1993), pp. 277–288.

[ES97] A. EFRAT AND M. SHARIR, *On the complexity of the union of fat objects in the plane*, in Proceedings of the 13th Annual ACM Symposium on Computational Geometry, ACM, New York, 1997, pp. 104–172.

[GS93] L. Guibas and M. Sharir, *Combinatorics and algorithms of arrangements*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer-Verlag, Berlin, 1993, pp. 9–36.

[GJ97] P. Gupta, R. Janardan, and M. Smid, *A technique for adding range restrictions to generalized searching problems*, Inform. Process. Lett., 64 (1997), pp. 263–269.

[K97] M. J. Katz, 3-*D vertical ray shooting and* 2-*D point enclosure, range searching, and arc shooting amidst convex fat objects*, Comput. Geom., 8 (1997), pp. 299–316.

[KL86] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.

[K98] M. van Kreveld, *On fat partitioning, fat covering and the union size of polygons*, Comput. Geom., 9 (1998), pp. 197–210.

[MP94] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, *Fat triangles determine linearly many holes*, SIAM J. Comput., 23 (1994), pp. 154–169.

[PT00] J. Pach and G. Tardos, *On the boundary complexity of the union of fat triangles*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 423–431.

[SS83] J. T. Schwartz and M. Sharir, *On the "piano movers" problem* I, Comm. Pure Appl. Math., 36 (1983), pp. 345–398.

[SS83b] J. T. Schwartz and M. Sharir, *On the "piano movers" problem* II, Adv. in Appl. Math., 4 (1983), pp. 298–351.

[SS89] J. T. Schwartz and M. Sharir, *A survey of motion planning and related geometric algorithms*, in Geometric Reasoning, D. Kapur and J. Mundy, eds., MIT Press, Cambridge, MA, 1989, pp. 157–169.

[SS90] J. T. Schwartz and M. Sharir, *Algorithmic motion planning in robotics*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 391–430.

[S94] F. van der Stappen, *Motion Planning amidst Fat Obstacles*, Ph.D. thesis, Faculteit Wiskunde & Informatica, Universiteit Utrecht, The Netherlands, 1994.

[SH93] F. van der Stappen, D. Halperin, and M. Overmars, *The complexity of the free space for a robot moving amidst fat obstacles*, Comput. Geom., 3 (1993), pp. 353–373.

[WS88] A. Wiernik and M. Sharir, *Planar realization of non-linear Davenport-Schinzel sequences by segments*, Discrete Comput. Geom., 3 (1988), pp. 15–47.

[ZS99] Y. Zhou and S. Suri, *Analysis of a bounding box heuristic for object intersection*, J. ACM, 46 (1999), pp. 833–858.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1761–1782

APPROXIMATE STRING MATCHING: A SIMPLER FASTER ALGORITHM*

RICHARD COLE† AND RAMESH HARIHARAN‡

Abstract. We give two algorithms for finding all approximate matches of a pattern in a text, where the edit distance between the pattern and the matching text substring is at most k. The first algorithm, which is quite simple, runs in time $O(\frac{nk^3}{m}+n+m)$ on all patterns except *k-break periodic* strings (defined later). The second algorithm runs in time $O(\frac{nk^4}{m}+n+m)$ on k-break periodic patterns. The two classes of patterns are easily distinguished in $O(m)$ time.

Key words. algorithms, string matching, edit distance

AMS subject classification. 68W40

PII. S0097539700370527

1. Introduction.

The *approximate string matching* problem is to find all of those positions in a given text which are the left endpoints of substrings whose *edit distance* to a given pattern is at most a given number k. Here, the edit distance between two strings is the minimum number of insertions, deletions, and substitutions needed to convert one string to the other. It is convenient to say that such a substring *matches* the pattern.

This problem is of significant importance, especially in the context of identifying sequences similar to a query sequence in a protein or nucleic acid database. In this case, however, the insertions, deletions, and substitutions need to be appropriately weighted. This variant of the problem is touched on only briefly in this paper for there are other issues to resolve.

Let n be the length of the text and m the length of the pattern. Then an $O(nm)$ algorithm is easy to obtain. This algorithm is a dynamic programming algorithm that finds the edit distance between every prefix of the pattern and every prefix of the text, not counting any cost for characters in the text which are to the left of the pattern. (We will refer to this as the *local edit distance*.) The number of text-pattern prefix pairs is $O(nm)$, and each pair can be processed in constant time, provided the pairs are processed in a certain natural order. The way to think about this order is to consider an array with columns associated with text prefixes of increasing length ordered toward the right and rows associated with pattern prefixes of increasing length ordered downward. Each entry in this array represents the local edit distance of a text-pattern prefix pair. These entries are computed in an order such that all entries in rows $1 \ldots i$ are computed before row $i+1$ is computed, and the entries within a row are processed in order from left to right.

Landau and Vishkin [LV89] obtained an $O(nk)$ algorithm for this problem. This algorithm was based on the above dynamic programming paradigm as well. However,

*Received by the editors April 5, 2000; accepted for publication (in revised form) December 18, 2000; published electronically September 12, 2002. This work was supported by NSF grants CCR-9503309 and CCR-9800085. An abstract of this work appeared in the *Proceedings of the ACM–SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 1998.
http://www.siam.org/journals/sicomp/31-6/37052.html

†Courant Institute, New York University, 251 Mercer Street, New York, NY 10012 (cole@cs.nyu.edu).

‡Department of Computer Science, Indian Institute of Science, Bangalore, 560012, India (ramesh@csa.iisc.ernet.in). This work was done in part while the author was visiting New York University.

their order of computing the array entries was a clever one. They observed that, in each 45-degree top-left to bottom-right diagonal, the entries are nondecreasing downward and that one need compute only k entries in each diagonal, namely, those entries whose value is different from the value of the preceding entry in the same diagonal. They show how these entries can be computed in constant time per entry, using a suffix tree of the pattern and the text.

Some other early work on this problem is described in [LV85, LV86, LV88, GG88, GP90].

The question that then arose was whether an $O(n + m)$ time algorithm was possible, at least for the case when k is small, e.g., $O(m^\epsilon)$, for some ϵ between 0 and 1. The intuition which suggests that this would be possible is that most of the pattern must match exactly when k is small.

An algorithm with an average case performance of $O(\frac{nk \log m}{m})$ time on random strings when $k < \frac{m}{\log m + O(1)}$ was given by Chang and Lawler [CL90]. While linear (sublinear, actually) on the average, the worst case performance of this algorithm was still $\Theta(nk)$. The assumption of the text being random is a strong one as random strings do not match with very high probability, but this algorithm may work well even on somewhat less random strings.

Baeza-Yates and Navarro [BN96] gave an algorithm with a running time of $O(n)$ for the case when $mk = O(\log n)$. In addition, they obtained another algorithm whose performance in the average case is $O(n)$ for medium k/m ratios. They also report finding this algorithm to be faster than previous algorithms experimentally, especially in the case when the pattern has moderate size, the error ratio k/m is not too high, and the alphabet size is not too small.

Recently, the above question was answered positively by Sahinalp and Vishkin [SV97], who obtained an algorithm with the following performance. Their algorithm takes $O(nk^{3+\frac{1}{\log 3}}(\frac{\log^* n}{m})^{\frac{1}{\log 3}} + n + m)$ time when there is *no periodicity* anywhere in the text or the pattern. Here, "no periodicity" means that even very local periodicity, e.g., two repeated characters, is not allowed. When there exists any periodicity in the text, the time taken by their algorithm is $O(nk^{8+\frac{1}{\log 3}}(\frac{\log^* n}{m})^{\frac{1}{\log 3}} + n + m)$. Their algorithm uses the technique of deterministic coin tossing in order to sparsify the set of diagonals which need to be processed in the above array and then processes only these diagonals using the Landau–Vishkin algorithm. This technique and the associated proofs of complexity and correctness, especially when there is periodicity present, are fairly involved.

Our contribution in the paper is twofold.

1. We give a very simple way of sparsifying the set of diagonals which need to be processed in the above matrix. This method is completely different from the Sahinalp–Vishkin algorithm and does not use deterministic coin tossing. All it requires is finding all occurrences of a number of aperiodic pattern substrings[1] of suitable length in the text. This immediately gives us an $O(\frac{nk^3}{m} + n + m)$ time algorithm, except when the pattern and the text are *k-break periodic*. By k-break periodic, we mean that there are $O(k)$ substrings of size k^2, each such that the portions of the text and the pattern between these substrings (or *breaks*, as we call them) are all periodic. We believe that k-break periodic is a rather strict property and k-break periodic strings would be quite rare in practice.

[1] "Aperiodic," here and throughout, refers to the usual notion of periodicity; i.e., the largest proper suffix of the substring which is also a prefix has a length of less than half that of the substring. This notion of aperiodicity is much weaker than that required by the Sahinalp–Vishkin algorithm.

2. We show how to process k-break periodic texts and patterns in $O(\frac{nk^4}{m}+n+m)$ time. While processing such strings in $O(\frac{nk^6}{m}+n+m)$ and even $O(\frac{nk^5}{m}+n+m)$ is quite easy, the $O(\frac{nk^4}{m}+n+m)$ time algorithm is nontrivial. The technical difficulties we face in obtaining this algorithm include the fact that the various periodic stretches between breaks need not have the same period and that periodic stretches in the pattern and the text need not align in a match of the pattern. Of course, there cannot be too many misalignments since only k mismatches are allowed.

Thus this paper gives an algorithm for approximate string matching which is not only faster and simpler than the Sahinalp–Vishkin algorithm but also helps us understand what kinds of text and patterns are hard to handle for this problem and why. We conjecture that the right bound is $O(\frac{nk^3}{m}+n+m)$ even for the k-break periodic case but have been unable to obtain an algorithm with this performance. We also believe that obtaining an algorithm which takes $o(\frac{nk^3}{m}+n+m)$ time will be hard.

The rest of this paper is organized as follows. Section 2 gives some necessary definitions. Section 3 gives an overview of our algorithm, and section 4 describes our sparsification algorithm and how it gives an $O(\frac{nk^3}{m}+n+m)$ time algorithm for the case when either the text or the pattern is not k-break periodic. One of the tools used in this algorithm is a simple modification of the Landau–Vishkin algorithm [LV89] and is described in section 5. (This modification is also used in [SV97].) Section 6 describes how to process the text when the pattern is k-break periodic but the text is not. Section 7 describes the first attempt at handling k-break periodic patterns and texts and obtains an $O(\frac{nk^6}{m}+n+m)$ time algorithm. Section 8 gives our more sophisticated scheme to handle such patterns and texts in $O(\frac{nk^4}{m}+n+m)$ time. Section 9 gives some intuition regarding the difficulties to be overcome in obtaining an $O(\frac{nk^3}{m}+n+m)$ time algorithm. Section 10 briefly discusses the weighted version of the problem.

2. Definitions and preliminaries. We assume that suffix trees for the pattern and the text can be constructed in linear time [CR94, F98].

We will assume that m, the pattern length, is at least $5k^3$. The Landau–Vishkin $O(nk+m)=O(\frac{nk^4}{m}+n+m)$ time algorithm is used for shorter patterns.

We will also assume that the text has length $2m-2k$ and the pattern has length m. If the text is longer, then it is partitioned into pieces of length $2m-2k$, with adjacent pieces overlapping in $m+k-1$ characters. The reason this suffices is that any substring of the text which matches the pattern has length in the range $[m-k, m+k]$. Thus all matches of the pattern are completely contained within some piece.

Periodicity. The *period length* of a string is defined to be the smallest i such that two instances of the string, one shifted i to the right of the other, match wherever they overlap. A string is said to be *aperiodic* if its period length is more than half the string length and *periodic* otherwise. A string is *cyclic* if it can be written as u^i, $i \geq 2$. A periodic string can be written as $u^i v$, where u is acyclic, $i \geq 2$, and v is a prefix of u. The following properties of periodic strings are well known (see [CR94]) and will be used implicitly throughout this paper.

1. The period length of a string can be determined in linear time, and so can its lexicographically least cyclic shift.
2. An acyclic string is not identical to any of its cyclic shifts. Therefore, a string s cannot be written as $u^i v$ and as $u'^i v'$, where $u \neq u'$, u, u' are acyclic, v is a prefix of u, v' is a prefix of u', and $|u|+|u'| \leq |s|$.
3. If u is acyclic, then every cyclic shift of u is acyclic as well.

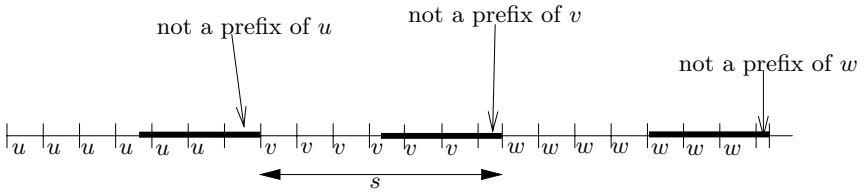


FIG. 1. *A k-break periodic string. Thick regions correspond to aperiodic substrings of length k^2. u, v, w are all at most $k^2/2$ in length.*

4. If string s has period length u but string sa (a is a single character) does not have period length u, then any suffix of sa of length at least $2u$ is aperiodic.

***k*-break periodic strings.** The pattern is said to be *k-break periodic* if it contains at most $2k-1$ disjoint aperiodic substrings of length k^2. The text is said to be *k-break periodic* if it contains at most $10k-2$ disjoint aperiodic substrings of length k^2.

LEMMA 2.1. *It can be determined in $O(m)$ time if the text and the pattern are k-break periodic. Further, if the pattern is not k-break periodic, then $2k$ disjoint aperiodic substrings of the pattern of length k^2 each can be found in $O(m)$ time. Similarly, if the text is not k-break periodic, then $10k-1$ disjoint aperiodic substrings of the text of length k^2 each can be found in $O(m)$ time.*

Proof. We consider only the pattern here. The text is processed similarly.

We process the pattern from left to right, performing various rounds. In each round, a new aperiodic length k^2 substring disjoint from all previously found substrings is determined. The stretches between any two consecutive substrings determined above will have period length at most $k^2/2$. Finally, if the collection of aperiodic strings constructed above has size less than $2k$, then the pattern is k-break periodic. The time taken by all rounds together will be $O(m)$.

A round is performed as follows. The portion of the pattern to the right of the last aperiodic length k^2 substring determined earlier is considered in this round. (If this is the first round, then the pattern is considered starting from its leftmost character.) The shortest prefix s of this portion (see Figure 1) of the pattern with the following properties is determined: $|s| \geq k^2$, and the length k^2 suffix of s is aperiodic. This computation will take $O(|s|)$ time and is described in the next paragraph. The length k^2 suffix of s is added to the collection of disjoint aperiodic strings being constructed. The total time taken over all rounds is clearly $O(m)$.

It remains to show how s is determined in $O(|s|)$ time. Let s' denote the length k^2 prefix of the above portion of the pattern. First, the period length δ of s' is determined in $O(k^2)$ time (see property 1 above). If $\delta > k^2/2$, then $s = s'$. Otherwise, the leftmost pattern character $p[i]$ to the right of s' with the property that $p[i] \neq p[i-\delta]$ is determined; s is the extension of s' up to and including $p[i]$. The time taken above is clearly $O(|s|)$. By property 4, the length k^2 suffix of s is aperiodic, as required. ∎

Canonical periods. Consider a periodic string $u^i v$, where v is a prefix of u, $i \geq 2$, and u is not cyclic. Note that all cyclic shifts of u are distinct since u is not cyclic. The *canonical period* of $u^i v$ is the string y which is the lexicographically smallest circular shift of u. Note that $u^i v$ can be written in a unique way as $xy^j z$, where x is a suffix of y and z is a prefix of y. Also note that, given u, y can be determined in $O(n)$ time (see property 1 above) and that y is acyclic as well (see property 3 above).

The following lemma about edit distance between periodic strings will be instrumental in the design of our algorithm.

LEMMA 2.2. *Consider two strings u^i and xw^jy of the same length, where x is a suffix of w, y is a prefix of w, and u, w are canonical periods of their respective strings. If $|u^i| = |xw^jy| \geq k^3 + k^2$ and $|u|, |w| \leq \frac{k^2}{2}$, then the edit distance between these two strings is at least $k+1$ unless $u = w$.*

Proof. Suppose $u \neq w$. Note that $i \geq 2(k+1)$. There are two cases.

First, suppose $|u| = |w|$; u and w cannot be cyclic shifts of each other as they are both canonical periods. It follows that each occurrence of u must incur at least one mismatch. The lemma follows in this case.

Second, suppose $|u| \neq |w|$. Partition u^i into disjoint substrings of length $|u|+|w|$. There must be at least $k+1$ such substrings. In addition, there must be at least one insertion/deletion/substitution in each such substring by property 2 above. The lemma follows. □

3. Overview. The algorithm first determines if the pattern is k-break periodic. More specifically, it determines whether there is a collection of $2k$ disjoint aperiodic length k^2 substrings in the pattern. Two cases are considered next, depending upon whether or not such a collection of substrings exists.

The sparse case. If such a collection exists, then section 4 describes a *sparsification* procedure that determines $O(\frac{m}{k^2})$ windows in the text, each of size k, which are the only locations where pattern matches can possibly begin. The matches starting in these windows are then found in $O(m)$ time by a simple modification of the Landau–Vishkin algorithm described in section 5.

The k-break periodic case. On the other hand, if no such collection exists, then the pattern is k-break periodic. In this case, in section 6, we show that all matches of the pattern in the text must occur in a portion of the text which is k-break periodic. We also show how this portion can be found in $O(m)$ time. In section 7, we show how to find all occurrences of k-break periodic patterns in k-break periodic texts in $O(k^6)$ time. This is improved to $O(k^4)$ time in section 8. This leads to an overall complexity of $O(\frac{nk^4}{m} + n + m)$ for this case.

4. Sparsification. In this section, we assume that the pattern has $2k$ disjoint aperiodic length k^2 substrings and that these substrings have been found. We call these substrings *breaks*. We show how to determine $O(\frac{m}{k^2})$ text windows, each of size k, in which potential matches of the pattern can begin. This will take $O(m)$ time.

First, we find all exact occurrences of each of these $2k$ breaks in the text. Note that these breaks have equal length. The time taken for this procedure is $O(m)$, using a standard multiple pattern matching algorithm [AC75].

Next, we partition the text into disjoint pieces of size k^2. Consider a particular piece $t[i \ldots j]$. We partition it into disjoint windows of size k each. We will show how to determine at most 12 windows such that any pattern match beginning in this piece must begin in one of these windows.

Note that at least k of the breaks must match exactly in any match of the pattern. Consider one particular break x. As x is aperiodic, any two occurrences of x in the text are at least a distance of $\frac{k^2+1}{2}$ apart. It is not hard to show that any pattern match beginning in $t[i \ldots j]$ with x matching exactly must begin in one of six size k windows in $t[i \ldots j]$ (see Figure 2). We can represent this fact by putting a mark for x on each of these windows. For a match to begin in a particular window, it must

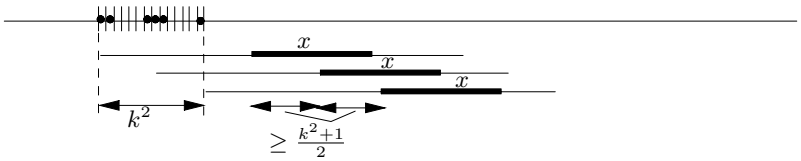


FIG. 2. *Windows which are marked for x. Filled circles indicate marks. Each instance of the pattern shown has x matching exactly.*

receive a mark from each of at least k breaks. Since each break marks at most six windows, there are at most $12k$ marks in all and therefore at most 12 windows in which matches can begin.

The verification of matches beginning in these windows is described next. It shows how all pattern matches beginning in a particular text window of length l can be found in time $O(k(k+l))$. Thus all matches beginning in a particular text window of length k can be found in $O(k^2)$ time. Then the time taken over all $O(\frac{m}{k^2})$ windows will be $O(m)$.

5. The Landau–Vishkin algorithm and sparse match verification. First, we give an overview of the Landau–Vishkin algorithm. Subsequently, we show how to find all pattern matches beginning in a particular text window of length l in time $O(k(k+l))$. We assume that the suffix tree of the pattern and the text combined has been constructed and processed for least common ancestor queries [SV88] so that the longest common prefix of any two suffixes in the text/pattern can be determined in $O(1)$ time.

5.1. The Landau–Vishkin algorithm. We review the Landau–Vishkin algorithm in this section. The classical approach to solving approximate string matching is to model it as a shortest paths problem on a graph defined on the entries of the following matrix.

Consider a matrix $A[0 \ldots m, 0 \ldots 2m-2k]$. $A[i,j]$ will be the value of the best match of $p[1 \ldots i]$ with any suffix of $t[1 \ldots j]$ for $1 \leq i \leq m, 1 \leq j \leq 2m-2k$. The 0th row and column are dummies put in for technical reasons which will become clear shortly.

The dependency graph. To determine the entries of A, we define a *dependency graph* G with weighted edges as follows. For each $i \geq 1$ and $j \geq 1$, there is a directed edge from $A[i,j]$ to each of $A[i,j-1], A[i-1,j], A[i-1,j-1]$, with weights $1, 1, y$, respectively, where y is 0 if $t[j] = p[i]$ and 1 otherwise. In addition, there is an edge from $A[i,0]$ to $A[i-1,0]$ with weight 1 and another from $A[0,j]$ to $A[0,j-1]$ with weight 0 for each $i \geq 1$ and each $j \geq 1$.

It is easy to see that the value of $A[i,j]$ is the weight of the shortest path from $A[i,j]$ to $A[0,0]$.

The algorithm. This algorithm takes $O(k)$ time for each diagonal in A. Consider a diagonal $A[0+*, j+*]$ (here , $*$ takes values from 1 to m). For each $l = 1 \ldots k$, it computes the bottommost vertex on this diagonal whose shortest path has weight l. This is done for each l in sequence, each point taking constant time to compute.

Suppose the above has been done for a particular value of l for all diagonals. Consider $l+1$ now and the diagonal $A[0+*, j+*]$. The bottommost vertex with shortest path $l+1$ on this diagonal is computed in constant time as follows. Let $A[0+a, j-1+a], A[0+b, j+b], A[0+c, j+1+c]$ be the bottommost vertices on their respective diagonals whose shortest paths have weight l (see Figure 3).

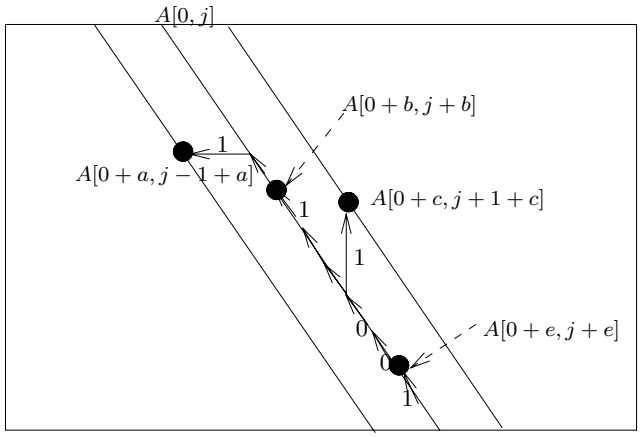


FIG. 3. *Finding the bottommost point on* $A[0+*, j+*]$ *with shortest path value* $l+1$.

Consider the three points $A[0+a, j+a], A[0+b+1, j+b+1], A[0+c+1, j+c+1]$, and take the bottommost of these three points; call this bottommost point $A[0+d, j+d]$. Next, find the bottommost point $A[0+e, j+e]$ such that all the edges on the path from $A[0+e, j+e]$ to $A[0+d, j+d]$ have weight 0. $A[0+e, j+e]$ is the required point. The longest 0 weight path along a diagonal starting at any particular point on the diagonal can be found in constant time using a longest common prefix computation.

5.2. Sparse match verification. We show how to find all pattern matches beginning in a particular text window $t[i \ldots i+l-1]$ of length l in time $O(k(k+l))$. In fact, our description will show how to find pattern matches ending in a particular text window $t[i \ldots i+l-1]$ of length l in time $O(k(k+l))$. Pattern matches beginning in the above window can be found using an analogous procedure. (Imagine reversing the pattern and the text.)

Note that in order to determine pattern matches ending in the above text window, it suffices to determine the bottommost points whose shortest paths have weight k on each of the diagonals $A[0+*, i-m+*] \ldots A[0+*, i-m+l-1+*]$. Let B denote the band formed by these diagonals. Let B' denote the band formed by the diagonals $A[1+*, i-m-k+*] \ldots A[1+*, i-m+l-1+k+*]$. The algorithm is based on the following simple observation.

Observation. If the shortest path of a point in B has weight at most k, then this shortest path stays entirely within B'. This is true because horizontal and vertical edges have weight 1 in G. The shortest path of a point in $B'-B$ might leave B'. However, to compute shortest paths for points in B, it is not necessary to compute shortest paths of points in $B'-B$ correctly; rather, it suffices to compute shortest paths using only edges in B'.

There are $O(k+l)$ diagonals in B'. Running the Landau–Vishkin procedure takes $O(k)$ time per diagonal, giving $O(k(k+l))$ time overall.

Remark. Suppose we are given two strings s_1, s_2, which are substrings of the pattern/text. Then note that the above procedure can easily be generalized to find the edit distances of s_2 with each of the $\Theta(k)$ longest suffixes of s_1 in time $O(k^2)$. Each such distance is determined correctly only if it is at most k. If it exceeds k, then the fact that it exceeds k is determined as well.

Further, note that the above procedure can also be generalized to find, for each of the $\Theta(k)$ longest suffixes of s_2, the edit distances with each of the $\Theta(k)$ longest suffixes of s_1 in time $O(k^2)$. As in the previous paragraph, each such distance is determined correctly if it is at most k; otherwise, the fact that it exceeds k is determined.

6. Text processing for k-break periodic patterns. We assume that the pattern has at most $2k-1$ breaks, i.e., disjoint substrings of length k^2, such that the stretches in between these breaks are periodic with period at most $k^2/2$. We show how to obtain a substring z of the text such that z is k-break periodic (i.e., has at most $10k-2$ breaks) and all potential matches of the pattern lie completely within z. This is done in $O(m)$ time.

Let x be the shortest text substring with its right end coinciding with the middle of the text and having $2(2k-1)+k+1 = 5k-1$ disjoint aperiodic substrings of length k^2. If no such x exists, then x is just the first half of the text. Let y be the shortest substring beginning in the middle of the text and having $5k-1$ disjoint aperiodic substrings of length k^2. If no such y exists, then y is just the second half of the text. We claim that all pattern matches must lie within $z = xy$.

Suppose a match of the pattern has its left end to the left of x. Recall that the text has length $2m-2k$. Then this pattern occurrence must touch or overlap the boundary of x and y, and, therefore, it must overlap the whole of x. (Otherwise, more than k insertions/deletions would be required.) However, x has $5k-1$ disjoint aperiodic substrings of length k^2, and at most $2(2k-1)$ of them can overlap breaks in the pattern; the remaining $k+1$ (or more) aperiodic text substrings of length k^2 must incur at least one mismatch each (because an aperiodic substring of length k^2 when aligned with a periodic stretch with period length at most $k^2/2$ must incur at least one mismatch; see also Figure 1). Therefore, the pattern cannot match in the above configuration, which is a contradiction. Similarly, it can be shown that the pattern cannot match with its right end to the right of y.

Determining x, y. This is done in $O(m)$ time using an algorithm similar to the algorithm in Lemma 2.1.

7. Finding matches of k-break periodic patterns. In this section, we assume that both the text and the pattern are k-break periodic. Recall that there are at most $2k-1$ ($10k-2$, respectively) disjoint aperiodic length k^2 substrings in the pattern (text, respectively) such that the stretches between them are periodic with period length at most $\frac{k^2}{2}$. Recall that these substrings are called breaks.

7.1. The $O(k^6)$ algorithm. First, we classify all potential matches into two categories. The first category contains potential matches in which some break in the pattern or some endpoint in the pattern is within distance $2(k^3+k^2)+k^2$ from the beginning or end of some break or endpoint in the text. The remaining potential matches are in the second category.

7.1.1. The first category. Note that matches in the first category must begin in one of $O(k^2)$ windows, each of size $O(k^3)$. All matches in these windows can be found using the algorithm in section 5 in $O(k^6)$ time. It remains to find matches in the second category.

7.1.2. The second category. Note that all potential matches in the second category also begin in one of $O(k^2)$ windows. Within each window, the order in which the various text and pattern intervals appear from left to right remains the same. The problem is that these windows could be long. Consider one such window.

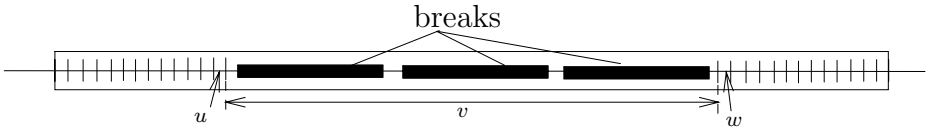


FIG. 4. *An interval.*

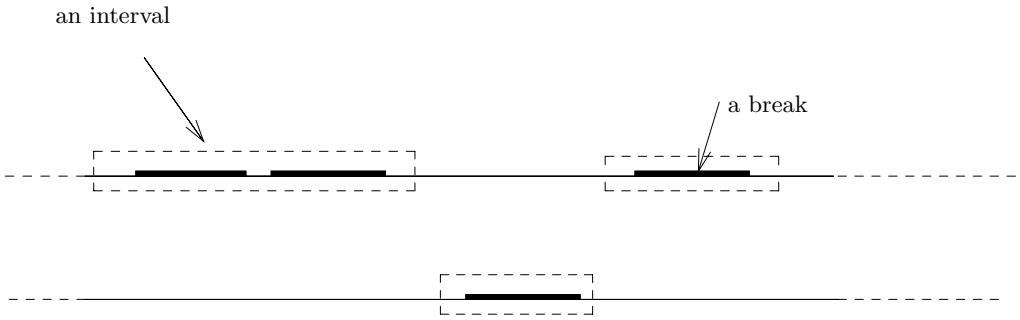


FIG. 5. *Placement of a portion of the pattern in the second category.*

Definitions. We need to form *intervals* in the text and the pattern before proceeding. First, we form groups of breaks in the pattern and the text. A group is a maximal sequence of breaks such that the periodic stretch between neighboring breaks has length less than $2(k^3 + k^2) + k^2$. An *interval* is a substring which includes all breaks in a group and extends on either side by a further distance described below. Let u denote the canonical period of the stretch to the left, and let w denote the canonical period of the stretch to the right. On the left side, the interval extends to the least distance between $k^3 + k^2$ and $k^3 + k^2 + \frac{k^2}{2}$ so as to have an integral number of occurrences of u (see Figure 4). On the right side, the interval extends to the least distance between $k^3 + k^2$ and $k^3 + k^2 + \frac{k^2}{2}$ so as to have an integral number of occurrences of w. Note that there are at least $2(k + 1)$ occurrences of the canonical period on either side. Thus each interval can be written in canonical form as $u^i v w^j$, where $|u^i|, |w^j| \geq k^3 + k^2$, $i, j \geq 2(k + 1)$, and u, w are not cyclic. We call u the *left canonical period* (*lcp*) of this interval and w the *right canonical period* (*rcp*).

Note that there are potentially two exceptions to the rules above, namely, the first and the last intervals in the pattern/text. The leftmost interval may be terminated by the left end of the pattern/text and therefore may not satisfy $|u^i| \geq k^3 + k^2$. Similarly, the rightmost interval may be terminated by the right end of the pattern/text and therefore may not satisfy $|w^j| \geq k^3 + k^2$. Prematurely terminated intervals are called *incomplete*; others are called *complete*.

The case when the pattern has only one interval which is incomplete on both the left and the right needs to be treated as a special case. We will address this case later. Until then, assume that each interval in the pattern is complete either on the right or on the left.

Properties of second category matches. Note that in all matches in this category, an interval or endpoint in the pattern (text, respectively) cannot overlap or touch an interval in the text (pattern, respectively). (See Figure 5.) This is because the period length of any periodic stretch is at most $\frac{k^2}{2}$, and, if such an overlap occurs,

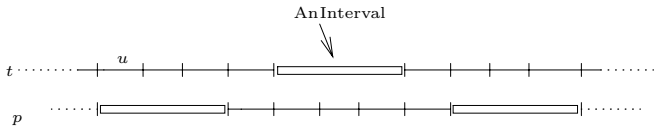


FIG. 6. *Locked intervals.*

then some break or endpoint in the text would be distance at most $2(k^3+k^2)+k^2$ from some break or endpoint in the pattern. Further, in all matches in this category, the endpoints of an interval will be *locked*, i.e., in alignment with the canonical periods in the overlapping periodic stretch (see Figure 6), as is proved in the following lemmas. This property enables us to process this category efficiently.

LEMMA 7.1. *The pattern has a second category match only if the lcps and rcps of all pattern intervals (except possibly the lcp of the first interval, in case it is incomplete on the left, and the rcp of the last interval, in case it is incomplete on the right) and of all text intervals overlapping the pattern are identical (denoted by, say, u). In addition, in any second category match of the pattern, all periodic stretches in the text (pattern, respectively) overlapping intervals in the pattern (text, respectively) must have canonical period u.*

Proof. First, consider the first part of the lemma. Order the intervals involved in this match (i.e., all pattern intervals and all text intervals overlapping the pattern) from left to right in order of occurrence. We show that the rcp of one interval s_1 in this sequence is identical to the lcp of the next interval s_2 in this sequence. Further, if interval s in this sequence is either complete or in the text, we show that the lcp and rcp of s are identical. The first part of the lemma follows.

Let u denote the rcp of s_1 and w denote the lcp of s_2. Note that $|u|, |w| \leq \frac{k^2}{2}$. Further, the suffix of s_1 which is cyclic in u has length at least $k^3 + k^2$ and likewise for the prefix of s_2 which is cyclic in w.

There are two cases for showing that the rcp of s_1 is identical to the lcp of s_2. First, suppose both intervals are in the pattern. Then they are both overlapped by a periodic stretch in the text with canonical period, say, x. However, by Lemma 2.2, $x = u$ and $x = w$. Therefore, $u = w$, as required. Second, suppose that s_1 is in the text and s_2 is in the pattern. (The case when s_1 is in the pattern and s_2 is in the text is symmetric.) Then s_1 is overlapped in the pattern by a stretch with canonical period w. By Lemma 2.2, $u = w$, as required.

Next, suppose s is in the text or is complete. We show that its lcp and rcp are identical. First, note that s will be complete in either case (i.e., if the pattern overlaps the first/last text interval and this interval is incomplete, then this match is in the first category). Assume that s is a complete interval in the text. (The case when s is a complete interval in the pattern is similar.) Let u be the lcp of s and v the rcp. Since s is complete, the prefix of s which is cyclic in u has length at least $k^3 + k^2$ as does the suffix of s which is cyclic in v. Note that $|u|, |v| \leq \frac{k^2}{2}$. The portion of the pattern overlapping s is periodic with canonical period, say, x. Then, by Lemma 2.2, $x = v = u$, as required.

Now consider the second part of the lemma. Consider a periodic stretch in the text having canonical period, say, w, which overlaps an interval s in the pattern. Assume that s is complete on the right. (A similar proof holds for the case when s is complete on the left; any interval must be complete either on the right or on the left, by our assumption above.) Note that $|u|, |w| \leq \frac{k^2}{2}$. Further, the suffix of s which

is cyclic in u has length at least $k^3 + k^2$. It follows from Lemma 2.2 that $u = w$. A similar proof holds for the case of a periodic stretch with canonical period w in the pattern overlapping an interval s in the text. (As stated earlier in this proof, s is necessarily complete in this case.) □

LEMMA 7.2. *Consider a match of the pattern in the second category. Consider any interval s involved in this match (i.e., all pattern intervals and all text intervals overlapping the pattern). If s is complete on the left and has lcp u, then the portions of the text and pattern between s and the next interval (text or pattern; if there is no such interval, then consider the endpoint) to the left have suffix u and incur no cost for edits. If s is complete on the right and has rcp u, then the portions of the text and pattern between s and the next interval (text or pattern; if there is no such interval, then consider the endpoint) to the right have prefix u and incur no cost for edits.*

Proof. We show that, if s is complete on the left and has lcp u, then the portions of the text and pattern between s and the next interval (text or pattern; if there is no such interval, then consider the endpoint) to the left have suffix u and incur no cost for edits. The other case is symmetric. We assume that s occurs in the pattern. The case when it occurs in the text is similar.

Recall from the definition of intervals that there are at least $2(k+1)$ occurrences of u at the beginning of s. Some instance of u in s amongst the rightmost $k+1$ instances must match the text exactly. There are two cases now.

First, suppose there are no intervals to the left of s involved in the above second category match. Then the portion of the pattern to the left of (and including) the above exact match of u has the form vu^i, where v is a suffix of u. Therefore, the portion of the text which overlaps the above portion of the pattern is also periodic with canonical period u (for an exact match of u occurs within s, and the portion of the text overlapping s is part of a periodic stretch with canonical period u by Lemma 7.1). The lemma follows from the fact that u is a prefix of s.

Next, suppose there is an interval to the left of s involved in the above second category match. Let s' be the rightmost such interval. By Lemma 7.1, the rcp of s' must also be u. As before, some instance of u in s amongst the rightmost $k+1$ instances must match the pattern exactly. Similarly, some instance of u in s' amongst the leftmost $k+1$ instances must match the pattern exactly. The portions of the pattern and the text between the above two exact matches of u are both cyclic in u. The number of edit operations in these portions is at least the difference in their lengths and is at most k. This number only reduces if all of these edits are transferred so that they occur at the right end of these portions. Since there are at most k such edits, all of them will now appear either within s or within the portion of the text overlapping s. It follows that the portion of the text starting from the above matching instance of u in s' and extending up to (but not including) the location aligned with the starting character of s matches the pattern exactly and is cyclic in u, as required. □

Definition. For each interval of length l, define its *locked edit distance* to be the minimum over all g, $h' \leq g \leq h$, of the edit distance between this interval and u^g. Here h is the number such that $|u|(h-1) < l + k \leq |u|h$, and h' is the number such that $|u|(h'-1) < l - k \leq |u|h'$. We compute the locked edit distance for each interval in the text and the pattern. Here, note that this distance is needed only if it is at most k. So we will only compute this distance if it is at most k. This computation takes $O(k^3)$ time as there are $O(k)$ intervals, and, for each interval, the algorithm given in section 5 will take $O(k^2)$ time. (Essentially, the shortest paths from certain

points lying on some of at most $2k+1$ diagonals need to be determined if these paths have cost at most k.)

We need special handling for incomplete intervals. Note that incomplete intervals in the text do not play a role in second category matches (i.e., if the pattern overlaps the first/last text interval and this interval is incomplete, then this match is in the first category). For the up to two incomplete intervals in the pattern, we need to redefine *locked edit distance* as follows.

If the rightmost interval in the pattern is incomplete on the right, its locked edit distance is the minimum edit distance between this interval and some prefix of u^h, where h is the number such that $|u|(h-1) < l + k \leq |u|h$. If the leftmost interval in the pattern is incomplete on the left, its locked edit distance is the minimum edit distance between this interval and some suffix of u^h, where h is the number such that $|u|(h-1) < l + k \leq |u|h$. As before, these locked edit distances need to be computed only if they do not exceed k. This can be done in $O(k^2)$ time using the algorithm in section 5 (see the remarks at the end of that section).

COROLLARY 7.3. *The edit distance between the pattern and the text for a second category match is just the sum of the locked edit distances over all pattern intervals and all text intervals overlapped by the pattern.*

Proof. Consider any second category match. By Lemma 7.1, each interval in the pattern is aligned with a periodic stretch in the text which is cyclic in u. Similarly, each interval in the text overlapped by the pattern is aligned with a periodic stretch in the pattern which is cyclic in u. From Lemma 7.2, it follows that each complete interval in the pattern (text, respectively) is aligned with a string in the text (pattern, respectively) which is cyclic in u. Further, an incomplete interval in the pattern which is complete on the right (left, respectively) is aligned with a string in the text which is periodic with period u and has suffix (prefix, respectively) u. (Note that if the endpoint of the text were closer, preventing a suffix of u, this would be a first category match.) The corollary follows from the definition of locked edit distance. □

Remark. Our aim is to determine all locations in the text where matches of the pattern begin. Consider any second category match. Recall from the last paragraph of the proof of Lemma 7.2 that edits in this match can be transferred to within pattern and text intervals (or to within portions in the text and pattern, respectively, overlapping these intervals) without increasing the edit distance. This transfer will not change the starting point of the match in the text except when the edits transferred are to the left of the leftmost interval (pattern or text, whichever is first). Therefore, edits to the left of the leftmost interval will have to be treated differently while determining the starting points of the various matches. Next, we show how these starting points can be determined in time $O(k^4)$ plus the number of matches.

Algorithm for second category matches. All locked edit distances are computed in $O(k^3)$ time as above. Recall from the beginning of this section that all second category matches begin in one of $O(k^2)$ windows. Consider one such window. The pattern occurs in this window if and only if the sum of the locked edit distances of the relevant intervals is at most k. The precise necessary and sufficient conditions for the pattern to occur starting at a particular text location i in this window are described next.

Note that fixing the window under consideration also fixes the left to right sequence of text and pattern intervals involved in matches in this window. Consider the leftmost interval (pattern or text, whichever is first). There are two cases, depending upon whether this interval is in the pattern or in the text. We consider each case in

turn. In each case, we show that the total time taken to output all matches in this window is $O(k^2)$ plus the number of matches.

Case 1. First, consider the case when this interval is in the pattern. Let x be the prefix of the pattern up to and including the right end of this interval. Recall from Lemma 7.2 that x must be aligned with a text substring with period u and having suffix u, in any match in this window. Let u' be a proper suffix of u such that the substring of the text beginning at location i has the form u' followed by several occurrences of u. Let $leftval$ be defined as the minimum edit distance between x and some string y with the following property: y begins with u', ends with u, and has canonical period u, and $|x| - k \leq |y| \leq |x| + k$. The pattern occurs starting at text location i if and only if $leftval$ plus the locked edit distance of all other pattern and text intervals involved in matches in this window sum to at most k.

It remains to describe how $leftval$ is determined in this case. We compute the edit distances of x and each of the $2k + 1$ longest suffixes of the unique string having canonical period u, suffix u, and length $|x| + k$. This takes $O(k^2)$ time using the algorithm in section 5 (see the remarks at the end of that section). For any relevant location i, $leftval$ is easily determined in constant time from this information.

Case 2. Second, consider the case when the leftmost interval is a text interval. Let x be the substring of the text starting from location i and extending up to and including the right end of this interval. Recall from Lemma 7.2 that x must be aligned with a pattern substring with period u and having suffix u, in any match in this window. Let u' be a proper suffix of u such that the pattern begins with u' followed by several occurrences of u. Let $leftval$ be defined as the minimum edit distance between x and some string y with the following properties: y begins with u', ends with u, and has canonical period u, and $|x| - k \leq |y| \leq |x| + k$. The pattern occurs starting at text location i if and only if $leftval$ plus the locked edit distance of all other pattern and text intervals involved in matches in this window sum to at most k.

$leftval$ is determined as follows for this case. Note that x depends on the value of i and that there are too many values of i which need to be considered. The key to fast computation of $leftval$ in this case is that the i's can be partitioned into $O(k)$ equivalence classes based on their offsets with respect to the canonical period u. Specifically, if the left end of the pattern is shifted by distance k, then the edit distance, if no more than k, is unchanged (so long as the left end of the pattern remains in the window in question). For the left end of I has at least $2(k + 1)$ disjoint occurrences of u; one of them is aligned with a copy of u in the pattern. A shift of the pattern left end by k units to the right can be thought of as removing this copy of u from the text, thereby leaving the edit distance unchanged.

We perform the following computation. Let z denote the suffix of u of length $|u'| + k$ if $|u| \geq |u'| + k$ and the string u otherwise. Let x' be formed by concatenating z with the leftmost interval (which is a text interval in this case). Let y' denote the string with canonical period u, suffix u, and length $|x'| + k$. For each of the k longest suffixes of x', we find the edit distances with each of the $3k$ longest suffixes of y'. This takes $O(k^2)$ time using the algorithm in section 5. (In fact, we are really only interested in suffixes of y' with prefix u'.) For any relevant text location i, $leftval$ is easily determined in constant time from this information.

Let u'' be that proper suffix of u such that the text substring starting at location i has prefix u'' followed by several occurrences of u. From Lemma 7.2, we note that, if $|u| \geq |u'| + k$ and $leftval$ is at most k, then $|u''| \leq |u'| + k$. Consider that suffix x''

of x' having the form u'' followed by the leftmost interval. We claim that, if $leftval$ is at most k, then it equals the minimum edit distance of x'' with a suffix y'' of y' whose size is within k of x'' and which begins with u'. To see this, the following three observations suffice. First, x can be obtained from x'' and y from y'' by inserting strings cyclic in u. Second, the leftmost interval is a suffix of both x, x'' and has at least $2(k+1)$ disjoint occurrences of u at its left end. Third, in any approximate match of x and y of value at most k, all but k of the various disjoint occurrences of u must match exactly; the same is true of any approximate match of x'', y'' of value at most k.

The special case. We consider the case when there is only one interval in the pattern, and it is incomplete in both directions. Thus the whole pattern is a single interval. In a second category match, the entire pattern is overlapped by a single periodic stretch in the text with canonical period, say, w, of length at most $\frac{k^2}{2}$. Clearly, in this situation, it suffices to find matches beginning in any window of length $|w|$; all second category matches in which the pattern is completely aligned with the periodic stretch with canonical period w can be interpreted from this information as in the previous paragraph. All matches beginning in a window of length $|w| \leq \frac{k^2}{2}$ can be found in $O(k^3)$ time using the algorithm in section 5.

8. The $O(k^4)$ algorithm. Recall that the pattern is k-break periodic. However, the periods of the periodic stretches between various pairs of consecutive breaks could be different. Suppose the pattern has at most $2k-1$ and the text has at most $10k-2$ *bad segments* of length at most $4|u|$ each such that the stretches between two adjacent bad segments are cyclic with canonical period u for some string u. Such texts and patterns are called *even more periodic*. First, we will show how to handle patterns and texts which are not even more periodic in $O(k^4)$ time. The even more periodic case is the hardest and is handled in section 8.1.

The following steps are performed to determine whether or not the pattern is even more periodic and to process it in case it is not.

Step 1. Recall that the periodic stretches in the pattern could have distinct periods. We choose a multiset U of disjoint substrings $u_1^2, \ldots, u_{2k+1}^2$ of the pattern as follows. The periodic stretches in the pattern are considered in nonincreasing order of period length. For a particular stretch with canonical period, say, v, all (or as many as necessary to achieve the desired cardinality of $2k+1$) disjoint occurrences of v^2 in it are added to U. This procedure continues until U has exactly $2k+1$ substrings in it. Since the pattern is assumed to have length at least $5k^3$ (see beginning of section 2), $2k+1$ such substrings always exist. (Recall that all periods have length at most $k^2/2$, and the total length of breaks is at most $2k^3$.) Let w denote u_{2k+1}.

There are two cases now. If $|w| \leq k$, then nothing is done in this step. Suppose $|w| > k$. Then we show how to obtain $O(\frac{m}{|w|})$ windows, each of size k, in which pattern matches can begin. In fact, we show something stronger, namely, in any window of size $|w|$ in the text, there are only a constant number of the above windows of size k in which pattern matches can begin.

All occurrences of the u_i^2's in the text are found in linear time using a standard multiple pattern matching algorithm [AC75]. Next, the text is partitioned into disjoint windows of size k each. Note that two occurrences of u_i^2 in the text occur at least a distance $|u_i|$ apart (since u_i is a canonical period and therefore not cyclic; see property 2 in section 2 as well). Pattern matches in which u_i^2 matches exactly must therefore begin in $O(\frac{m}{|u_i|}) = O(\frac{m}{|w|})$ windows; this is represented by putting a mark

for u_i^2 on each such window. Over all strings in U, the total number of marks put is at most $(2k+1) * \frac{2m-2k}{|w|}$ (the size of the text is $2m-2k$), and only windows with at least $k+1$ marks can have pattern matches beginning in them. It follows that pattern matches can begin in only $O(\frac{m}{|w|})$ windows of size k each. In addition, any two windows which are more than a distance k apart and which receive $k+1$ marks each must both receive a mark for some u_i^2. Since any two occurrences of u_i^2 occur at least a distance $|u_i| \geq |w|$ apart, any two windows which are more than a distance k apart and which receive $k+1$ marks each must actually be a distance at least $w-2k$ apart. It follows that, in any window of size $|w|$ in the text, there is only a constant number of the above windows of size k in which pattern matches can begin. The total time taken is $O(m)$.

We now include all occurrences of all u_i's which are not identical to w as breaks in the pattern. The number of breaks in the pattern is still $O(k)$, each break being $O(k^2)$ in length. In addition, all periodic stretches have period lengths of at most $|w|$. In the text, all periodic stretches with period lengths of more than $|w|$ are partitioned into disjoint substrings of length k^2; these substrings are also included as breaks. Note that one substring in each stretch could have a length less than k^2; this substring is just merged with the next break to the right. So the text has several breaks now, each of length between k^2 and $2k^2$. Now, as in section 6, the text is trimmed so that it has only $O(k)$ breaks. The key property used in this trimming is that any break, when aligned with a periodic stretch in the pattern (which now has period length of at most $|w|$), must incur at least one mismatch. Thus both the pattern and the text now have $O(k)$ breaks of length $O(k^2)$ each, with all intervening periodic stretches having period length at most $|w|$.

Step 2. We partition p into disjoint pieces of length $2|w|$. A piece-substring is a substring beginning and ending at piece boundaries. A piece-substring is *homogeneous* if at least three-fourths of the pieces in it have the same canonical period; it is *heterogeneous* otherwise.

Step 2: Case 1. If there exists a heterogeneous piece-substring of length $2|w| * (4k+1)$ in the pattern, then this piece-substring must overlap a break in the text in any match of the pattern. This is because any alignment of this piece-substring with a periodic stretch (which now has period length of at most $|w|$) is guaranteed to give at least $k+1$ mismatches. (At least $k+1$ pieces will have a canonical period different from the canonical period of the periodic stretch.)

A heterogeneous piece-substring, if one exists, can be found in $O(m)$ time. In addition, if such a piece-substring exists, then all matches of the pattern must begin in $O(k)$ windows, each of size $O(k^2+k|w|)$. If $|w| \leq k$, then the total size of these windows is $O(k^3)$, and all matches beginning in these windows can be found in $O(k^4)$ time using the algorithm in section 5. If $|w| > k$, then these windows can be further refined by taking intersections with the windows obtained in Step 1 (recall that pattern matches begin in only $O(1)$ length k windows in any length $|w|$ window) to give $O(k^2)$ windows each of size $O(k)$. Thus the total time taken to find all matches using the algorithm in section 5 is $O(k * k^2 * k) = O(k^4)$ in this case as well.

Step 2: Case 2. Suppose there is no heterogeneous piece-substring of length $2|w|*(4k+1)$ in the pattern. Then three-fourths of the pieces in every piece-substring of length $2|w| * (4k+1)$ have the same canonical period, u, say, $|u| \leq |w|$. Any periodic stretch which has a canonical period different from u has a length less than $2|w|(k+1) + 4|w| = 2|w|(k+3)$; otherwise, there would be at least $k+1$ complete pieces occurring contiguously within this periodic stretch, each having a canonical

period different from u; any piece-substring of length $2|w| * (4k+1)$ containing these pieces would then be heterogeneous.

We now make each periodic stretch in the pattern which has a canonical period different from u and a length at least $2|w|$ a break. (This is in addition to the existing breaks.) Periodic stretches in the pattern with canonical periods different from u and length less than $2|w|$ are appended to the next breaks to the right. Thus the pattern now has $O(k)$ breaks, each of length $O(|w|k+k^2)$, and all intervening periodic stretches have period u.

In the text, we redefine the breaks as follows. All existing breaks continue to be breaks. Recall that each of these has length between k^2 and $2k^2$. Call these breaks class 1 breaks. All periodic stretches with canonical period different from u also become breaks now; call these breaks class 2 breaks. Next, both classes of breaks are together reorganized into a new set of breaks so that each resulting break has length at least $4|w|k + 2k^2$ and at most $2(4|w|k + 2k^2)$; this reorganization involves clubbing together existing breaks to form new breaks by including intervening strings and extending at the ends, or alternatively, partitioning a break into smaller breaks, if necessary. The length restrictions on the resulting breaks imply that the above reorganization allows for each class 1 break to be contained completely inside some resulting break; further, if a class 2 break is broken down and distributed over several resulting breaks, then each substring into which it is broken down has length at least $2|w|$. Then each resulting break contains one of the following (below, the first two cases relate to those resulting breaks which include a class 1 break, and the third relates to those resulting breaks which are derived from class 2 breaks):

1. A length k^2 aperiodic substring (these were the original breaks).
2. A substring with period length more than $|w|$ and having at least two consecutive occurrences of the canonical period (see the new breaks defined just before Step 2; also recall that $|w| \leq \frac{k^2}{2}$). Clearly, this canonical period will be different from u.
3. A substring of length $2|w|$ with period length at most $|w|$ and canonical period different from u, $|u| \leq |w|$.

Now, as in section 6, the text is trimmed so that the total number of breaks in each half of the text is $O(k)$. The key property used in this trimming is that any text break, when aligned with a periodic stretch with canonical period u in the pattern, must incur at least one mismatch. This holds because of the properties listed above. Thus both the pattern and the text now have $O(k)$ breaks of length $O(k^2 + |w|k)$ each, with all intervening periodic stretches having canonical period u.

Now consider those substrings of the pattern of length $2|u|$ which do not have canonical period u. There are two subcases now.

First, suppose there are at least $2k$ such disjoint substrings. Then at least k of these substrings must match exactly in any match of the pattern. For such a substring to match exactly, it must be aligned with a text substring which is not a periodic stretch of u's. Recall that the text has $O(k)$ breaks and that all intervening periodic stretches have canonical period u. It follows that there are $O(k)$ windows in which possible matches of the pattern can begin, each window having length $O(|w|k + k^2)$. If $|w| \leq k$, then all these matches can be found in $O(k * k^2 * k) = O(k^4)$ time using the algorithm in section 5. And, if $|w| > k$, then each of the above windows of size $O(|w|k + k^2)$ can be further refined by taking intersections with the windows obtained in Step 1 to get $O(k^2)$ windows of size $O(k)$ each; the $O(k^4)$ time bound follows in this case as well.

The second subcase arises when there are fewer than $2k$ disjoint substrings of length $2|u|$ with canonical period different from u in the pattern. As in section 6, the text can now be trimmed so that it has at most $10k - 2$ of these. Clearly, all stretches in the text and in the pattern between the above substrings are periodic with canonical period u. (This follows from the fact that if two substrings, both having length $2|u|$ and canonical period u, overlap in $|u|$ locations, then their union also has canonical period u by definition.) Next, by extending each substring of length $2|u|$ with canonical period different from u on either side, the intervening stretches can be made cyclic in u (earlier they were just periodic with canonical period u but not necessarily cyclic); the length of each such substring can go up to $4|u|$ in the process. Our text and pattern are now both even more periodic (defined at the beginning of this section).

8.1. The $O(k^4)$ algorithm for the even more periodic case. To get a faster algorithm, we have to define intervals which have stronger properties than those defined in section 7. We define an *interval* in the pattern (text, respectively) to be a set of disjoint substrings of the pattern (text, respectively). Roughly speaking, intervals are formed by extending *bad segments* (substrings of length between $2|u|$ and $4|u|$ which do not have canonical period u) at either end while skipping over other intervals. Intervals will always have the property that they end in at least one, possibly more, occurrences of the period u at each end. The *span* of an interval is the substring between and including the leftmost and the rightmost characters in the interval. In contrast to the intervals defined in section 7, spans of intervals defined here could be nested one inside the other.

Recall the definition of locking from Figure 6. We say that an interval in the pattern (text, respectively) *locks* in a particular alignment if the portion of the text (pattern, respectively), if any, with which this interval is aligned is a cyclic repetition of u.

Our strategy will be to identify intervals in the pattern and the text with total length $O(k|u|)$, with each interval having length at least $2|u|$. These intervals will have the following property: in any match of the pattern, either some pattern interval overlaps some text interval, or all of the pattern and text intervals are locked.

All matches in the first category clearly occur in at most $O(k^2)$ windows, each of length $\max\{k, |u|\}$. If $|u| \leq k$, then the total length of all of these windows is $O(k^3)$, and all matches in these windows can be found in $O(k^4)$ time using the algorithm in section 5. If $|u| > k$, then recall that $|u| \leq |w|$, that potential matches of the pattern have been determined in Step 1, and that there is only a constant number of windows of length k within any length $|w|$ window in which these matches can begin. It follows that all matches must again begin in $O(k^2)$ windows, each of length $O(k)$; these matches can again be found in $O(k^4)$ time.

Matches in the second category will also occur in $O(k^2)$ windows but of larger size. Whether or not the pattern matches in one such window will depend upon the *locked edit distance* of some of the intervals defined. These matches will be easy to find. In particular, if the pattern matches at a particular position in this window, then it will match at all positions which are shifts of multiples of $|u|$ from this position in this window.

8.2. Defining intervals. We show how the pattern is processed. The text is processed similarly.

We define intervals to contain all sufficiently small strings that are not repetitions of string u. More specifically, an interval I will be a string with a 2^i-fold repetition of

string u at either end for a suitable i. The best match of I with a string u^k, $|u^k| \geq |I|$, for suitable k will be in locked alignment. Intervals are chosen to minimize i in a sense made precise below. Further, the intervals are chosen so that, in any match in which the intervals in the text and pattern do not overlap, the intervals are all in locked alignment.

We define intervals as follows in $O(\log k)$ rounds. In each round, a set of partially formed intervals inherited from the previous round is processed. These intervals will be disjoint from each other. Some of the intervals being processed in the current round will be fully formed at the end of this round; these will not be processed in subsequent rounds. The remaining intervals will be processed further in the subsequent rounds.

The first round begins with a minimal collection of disjoint intervals, called *initial* intervals, where each initial interval is just a bad segment (defined at the beginning of section 8.1). Recall that the portions of the string between the initial intervals are cyclic in u. The following procedure is performed in each round i, $i \geq 1$.

2^i-extending interval I. For each partially formed interval I being processed in the current round i, a 2^i-*extension* is determined as below. Starting from the left end of I, walk to the left, skipping over any substrings in fully formed intervals, until either another partially formed interval is reached or 2^i instances of u have been encountered. The same procedure is repeated at the right end. The substrings walked over in this process (ignore the substrings skipped over) along with the substrings in I together constitute the 2^i-extension of I.

An interval I processed in round i is said to be *successful* in this round if, after extension, it does not overlap or touch another extended interval on both the left and on the right.

Finally, we form new intervals by taking a *union* of the various extended intervals. Each new interval comprises maximal collections of extended intervals above such that consecutive extended intervals in each collection overlap or touch each other. Thus, if two extended intervals overlap or touch, then they become part of the same interval now. Each new interval comprises exactly those pattern positions which belong to one of the extended intervals in the corresponding maximal collection of extended intervals. Of these new intervals, some will be fully formed, as described in the next paragraph. Those which are not fully formed will be carried over to the next round.

Condition for full-formedness. Each interval will have an i-*nested cost* to be defined below. Those intervals I whose span has locked edit distance (with respect to u) at most 2^i plus the i-nested cost of I will be fully formed at the end of this round; the remaining intervals will be processed again in the next round.

Definitions. The i-*current cost* of an interval I which is processed in round i is the locked edit distance of the span of I with respect to u if it is fully formed by the end of round i and 2^i plus its i-nested cost if it is not yet fully formed at the end of round i. The *final cost* of an interval is its current cost at the end of the last round or its locked edit distance (with respect to u) if it is fully formed. The i-nested cost of I is the sum of the final costs of the fully formed intervals which were skipped over while forming I and the $(i-1)$-current costs of those partially formed intervals which are nested within I and were unsuccessful in round i. As the base case, we define the 0-current cost of an initial interval to be 1. Lemma 8.1 describes the motivation for the above definitions.

LEMMA 8.1. *For all $i \geq 0$, the i-current cost of an interval I processed in round i is a lower bound on the cost of aligning the span s of I with a periodic stretch of u's.*

Proof. Consider a least cost match of s in a periodic stretch of u's. If $i = 0$, then

the lemma follows from the fact that initial intervals have canonical periods different from u and therefore incur at least one mismatch. So assume that $i > 0$.

Note that s has 2^i occurrences of u at either end, possibly interspersed with intervals fully formed before round i. Some or all of these occurrences of u in s could be out of alignment with u's in text. If all of these occurrences of u at the left end or at the right end are out of alignment, then the cost of aligning s is at least 2^i plus, inductively, the i-nested cost. On the other hand, if at least one occurrence of u on either side aligns, then we claim that all occurrences of u further from the extremes of s from these two occurrences align as well. This is because the portions of s outside these two occurrences of u consist only of u's and other fully formed intervals, and fully formed intervals, by induction, cost at least (and, of course, at most) their locked edit distance. Therefore, the cost of the best match of s is the same as its locked edit distance with respect to u. The claim now follows from the fact that the i-current cost of I is the smaller of this distance and 2^i plus the i-nested cost. □

Termination conditions for the rounds. The ith round is the last round if the sum of the i-current costs of those intervals which are obtained in round i and are not nested inside other intervals and the sum of the final costs of those intervals that are fully formed earlier and not nested inside other intervals (we call both of these kinds of intervals together *final* intervals) exceeds k, or if all intervals are fully formed. When the sum of the above costs is more than k, all matches of the pattern must have some interval in the text overlapping or touching some interval in the pattern. Clearly, the number of rounds is $O(\log k)$. The cost of processing a round, i.e., extending and computing the costs, is $O(k^3)$. (Each of up to $O(k)$ intervals requires a locked edit distance calculation, and each calculation is performed in $O(k^2)$ time using the algorithm described in section 5.) This can be reduced to $O(k^2)$ time by performing the edit distance calculations more carefully, keeping in mind that the collective error that can be tolerated over all edit distance calculations is k. However, the bound of $O(k^3)$ per round suffices to achieve our final bound of $O(k^4)$.

Remark on the text. A similar formation of intervals is done in the text, except that interval formation continues until either each interval is fully formed or $\log k + 1$ rounds are done, whichever is sooner.

Special cases. The above interval formation algorithm needs to be suitably modified to account for the endpoints of the text and the pattern. We will very briefly sketch the special handling of intervals which encounter premature termination at either the left or the right end. Consider an interval which is prematurely terminated on the left. Intervals prematurely terminated on the right are handled similarly. In future rounds, this interval will be extended only to the right until it is fully formed. Recall that full-formedness is related to the locked edit distance of the span of the interval (with respect to u). The locked edit distance for such intervals is defined as in section 7 (i.e., the span of this interval need not be aligned with a cyclic string of u's but with a string whose canonical period is u and which has suffix u).

Interval lengths. We need the following lemma before describing the remainder of the algorithm.

LEMMA 8.2. *The length of the span s of an interval I obtained in round i is at most $8|u|*i$-current cost of I.*

Proof. Consider the various initial intervals J in s. For each such initial interval J, consider the interval $\text{int}_j(J)$ which is the unique interval processed in round j whose span contains J. There may not be such an interval, of course. J is said to be *alive* in round j if it is the leftmost (rightmost, respectively) initial interval in $\text{int}_j(J)$ at the

beginning of round j and $\text{int}_j(J)$ hasn't yet reached the left endpoint (right endpoint, respectively). Let the last round in which J is alive be denoted by $last(J)$. The *contribution* of J to s is defined to be the sum of the lengths of all of the strings involved in extending the intervals $\text{int}_1(J), \ldots, \text{int}_{last(J)}(J)$ plus the length of J itself. Clearly, the length of s is at most the sum of the lengths of the contributions of the various initial intervals in s.

The contribution of J is at most $2 * (\sum_{l=1}^{last(J)} 2^l) * |u| + |J| \leq 2(2^{last(J)+1} - 2)|u| + 4|u| \leq 2 * 2^{last(J)+1} * |u|$ if $last(J) \neq 0$ and $|J| \leq 4|u|$ otherwise. The $(last(J) - 1)$-current cost of $\text{int}_{last(J)-1}(J)$ is at least $2^{last(J)-1}$ plus its $(last(J) - 1)$-nested cost if $last(J) \geq 1$. We call the quantity $2^{last(J)-1}$ the *capacity* of J (unless $last(J) = 0$, in which case the capacity is defined to be 1). It is easy to see that the capacities of the various initial intervals in s sum to at most the i-current cost of I. The lemma follows. □

The algorithm. First, intervals are formed as above. Next, two minimal sets of final text intervals, one on either side of the middle of the text, each with total final cost exceeding k, are chosen. (If either of these two sets does not have final cost exceeding k, then all of the text intervals in the corresponding half are taken.) By Lemma 8.2, the total lengths of the spans of these final text intervals and the final pattern intervals will be $O(k|u|)$. Ignore the remaining text intervals for the moment. Each match in the span of one of these text intervals that overlaps or touches the span of one of the final pattern intervals is found. In addition, matches in which one of the endpoints of the pattern is aligned with one of these text intervals is found. These matches occur in $O(k^2)$ windows, each of size $O(\max\{k, |u|\})$. If $|u| > k$, then each of the above windows can be further refined by taking intersections with the windows obtained in Step 1 to get $O(k^2)$ windows of size $O(k)$ each. All such matches can then be found in $O(k^4)$ time using the algorithm in section 5.

Next, we consider the remaining matches of the pattern. Note that the spans of the final intervals in the pattern cannot overlap with the spans of the above final intervals chosen in the text. In addition, the text can be trimmed so as to contain only the above final intervals, by an argument similar to the one used in section 6. The reason for this is that, by Lemma 8.1, the above final intervals in the left half of the text will incur more than k mismatches if all of them are overlapped by the pattern (note that they must all be aligned with periodic stretches having canonical period u in the matches being considered) and likewise for the above final intervals in the right half. It follows that the spans of the final intervals in the pattern cannot overlap with the spans of any of the final intervals in the text in any of the remaining matches.

The remaining matches occur in $O(k^2)$ windows as well. Consider one such window. Consider the final costs of the final intervals in the pattern and the final costs of those final intervals in the text which overlap the pattern. If any one of these text intervals is partially formed, then the pattern cannot match the text because the final cost of this text interval is more than k, and Lemma 8.1 implies that the span of this interval must incur more than k edit operations when aligned with a periodic stretch of u's. Similarly, if any of the pattern intervals is partially formed, then, again, the sum of the final costs of these pattern intervals exceeds k, and the pattern cannot match. So suppose that all of these text and pattern intervals are fully formed. Then the final cost of each such interval is its locked edit distance. If the sum of these final costs is at most k, then the pattern matches at intervals of $|u|$ in this window with all of these final intervals locked, and otherwise it does not match anywhere in this window. The precise locations where matches occur can be determined as in the

algorithm for second category matches described toward the end of section 7.1.2.

9. Is $O(\frac{nk^3}{m}+n+m)$ running time possible? The following text and pattern appear to form a hard case for our problem. They are defined in terms of an acyclic string u. Apart from $\Theta(k)$ substrings, each of length equal to $|u| = \Theta(k)$, the text and the pattern are periodic with period u. Suppose these *bad* substrings appear at intervals of $\Theta(k^2)$ in the text and at intervals of $\Theta(k)$ in a length $\Theta(k^2)$ prefix of the pattern. There are $\Theta(k^2)$ windows, each of size k, in which one of these pattern substrings overlaps some text substring. Exactly one bad pattern substring overlaps a bad text substring in any of the pattern placements in these windows. Our current $O(k^4)$ algorithm will take $\Theta(k^2)$ time to determine matches, if any, in each window, giving an $\Theta(k^4)$ time algorithm for this case. However, it is conceivable that an algorithm which takes $O(k)$ amortized time per window can be obtained by observing that the average edit distance between pairs of text-pattern substrings must be $O(1)$; otherwise, there can be few matches. The difficulty we face is that the occurrences of u among the bad substrings of the pattern need not align with occurrences of u in the text.

10. The weighted case. In the weighted case, deletions of different characters and the various substitutions may have differing costs, but, by way of normalization, all will be required to have cost at least 1.

The approximate matches with cost $\leq k$ can be found using essentially the same algorithm; the only change needed is to the Landau–Vishkin algorithm to take into account the differing costs. The details are left to the reader.

A important application in the weighted case is to match a pattern against a database of strings. We would like to apply the above algorithm. For efficiency, one approach would be to have a precomputed suffix tree for the database of strings. This suffix tree would then need to be incremented to incorporate the pattern string so as to enable the above algorithm to be used. Following the match calculation, the modification to the suffix tree would need to be undone. It would also be useful to support both insertions and deletions to the database. We leave this topic as an open problem.

REFERENCES

[AC75] A. V. Aho and M. J. Corasick, *Efficient string matching: An aid to bibliographic search*, Comm. ACM, 18 (1975), pp. 333–340.

[BN96] R. Baeza-Yates and G. Navarro, *A faster algorithm for approximate string matching*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1075, Springer-Verlag, Berlin, 1996, pp. 1–23.

[CL90] W. I. Chang and E. Lawler, *Approximate string matching in sublinear expected time*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1990, pp. 116–125.

[CR94] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, New York, 1994, pp. 27–31.

[F98] M. Farach, *Optimal suffix tree construction with large alphabets*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 137–143.

[GG88] Z. Galil and R. Giancarlo, *Data structures and algorithm for approximate string matching*, J. Complexity, 4 (1988), pp. 33–72.

[GP90] Z. Galil and K. Park, *An improved algorithm for approximate string matching*, SIAM J. Comput., 19 (1990), pp. 989–999.

[LV85] G. Landau and U. Vishkin, *Efficient string matching in the presence of errors*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 126–136.

[LV86] G. Landau and U. Vishkin, *Introducing efficient parallelism into approximate string matching and a new serial algorithm*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 220–230.

[LV88] G. Landau and U. Vishkin, *Fast string matching with k differences*, J. Comput. System Sci., 37 (1988), pp. 63–78.

[LV89] G. Landau and U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 158–169.

[SV97] S. Sahinalp and U. Vishkin, *Approximate Pattern Matching Using Locally Consistent Parsing*, manuscript. Abstract appeared in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 320–328.

[SV88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1783–1793

A MATTER OF DEGREE: IMPROVED APPROXIMATION ALGORITHMS FOR DEGREE-BOUNDED MINIMUM SPANNING TREES*

J. KÖNEMANN† AND R. RAVI†

Abstract. In this paper, we present a new bicriteria approximation algorithm for the *degree-bounded minimum spanning tree* problem. In this problem, we are given an undirected graph, a nonnegative cost function on the edges, and a positive integer B^*, and the goal is to find a minimum-cost spanning tree T with maximum degree at most B^*. In an n-node graph, our algorithm finds a spanning tree with maximum degree $O(B^* + \log n)$ and cost $O(\mathtt{opt}_{B^*})$, where $\mathtt{opt}_{B^*}$ is the minimum cost of any spanning tree whose maximum degree is at most B^*. Our algorithm uses ideas from Lagrangean duality. We show how a set of optimum Lagrangean multipliers yields bounds on both the degree and the cost of the computed solution.

Key words. approximation algorithms, network algorithms, bicriteria approximation, spanning trees, degree-bounded spanning trees, Lagrangean relaxation

AMS subject classifications. 68W25, 05C05, 05C85, 68R10, 90C29

PII. S009753970036917X

1. Introduction.

1.1. Motivation and formulation.

In the design of computer networks a fundamental problem is that of transmitting a single information packet from a given source-host to a set of recipient-hosts. This problem is widely known as the *broadcast* or *multicast* problem, depending on whether we want to transmit the packet to all other hosts or to a specific subset of recipients. Both problems have been widely studied [3, 6, 15]. In particular, it is believed that the multicast problem will play an increasingly important role in data networks.

A naive solution to the multicast problem would be to implement it as a series of unicasts. In other words, the source sends a single packet to every recipient host. The routing is done independently for each of the unicasts. However, the cost of this approach in terms of bandwidth consumption becomes unacceptable if the number of hosts in the multicast group grows.

Graph theoretic ideas have turned out to be essential in the design of efficient network routing protocols. A physical network can be modeled as a complete graph where each host is associated with a node, and an edge represents the *virtual link* between the corresponding hosts. Usually, edges of that graph are annotated by the *transmission delay* of the corresponding virtual link. A standard solution to broadcasting and multicasting problems then is to send packets along the edges of a minimum spanning tree (MST) rooted at the source node [15]. Every internal node duplicates the incoming message and sends it to its children.

However, a spanning tree might have a high *fan-out* at certain internal nodes. Switches in point-to-point networks may vary in their ability to support multicasting. That is, some routers may not support multicasting at all and others may support only a limited number of copies they can make of an incoming packet [17]. Bauer

*Received by the editors March 16, 2000; accepted for publication (in revised form) May 1, 2002; published electronically September 12, 2002. This research was supported in part by NSF grant 96-25279.
http://www.siam.org/journals/sicomp/31-6/36917.html
†GSIA, Carnegie Mellon University, Pittsburgh, PA 15213 (jochen@cmu.edu, ravi@cmu.edu).

and Varma [1] show that it is natural to model switch capabilities by node degrees in graphs.

The preceding discussion suggests that a solution to the multicasting problem should minimize the total transmission delay *and* the maximum degree of a vertex in the computed solution. Traditional approaches for this kind of *bicriteria* problem often compute the minimum-cost solution under a linear combination of the two cost measures [12, 14]. However, in the case of very disparate objectives, these techniques usually do not produce useful solutions.

In this paper, we address a natural budgeted version of the *degree-bounded minimum spanning tree* (BMST) *problem*. Here, we are given an undirected graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, and a positive integer $B \geq 2$. We would like to find a spanning tree T of maximum vertex degree at most B and minimum cost. This formulation was first introduced in [14] and can be modeled by the following integer linear program:

$$\text{(IP)} \qquad \texttt{opt}_B = \min \quad \sum_{e \in E} c_e x_e$$

$$\text{(1.1)} \qquad \text{s.t.} \quad x(\delta(v)) \leq B \quad \forall v \in V,$$

$$x \in \text{SP}_G\,,$$

$$x \text{ integer}.$$

Here, $\delta(v)$ denotes the set of all edges of E that are incident to v and SP_G is the spanning tree polyhedron, that is, the convex hull of edge-incidence vectors of spanning trees of G. We note that complete descriptions of SP_G are known in the literature [2, 4].

1.2. Previous work and our result. Ravi et al. [14] showed how to compute a spanning tree T of maximum degree $O(B \log(\frac{n}{B}))$ and total cost at most $O(\log n)\,\texttt{opt}_B$. They generalize their ideas to Steiner trees, generalized Steiner forests, and the node-weighted case.

Another result that is related to our work is given in a paper by Khuller, Raghavachari, and Young [9]. The authors show how to compute a spanning tree of n points in the plane that has degree at most 3 (4) and cost at most 1.5 (1.25)—that of a minimum-cost spanning tree (without any degree constraints).

While the approximation factor of $O(\log n)$ on the cost of the solution cannot be improved substantially (via reductions from the set covering problem [10]) in the node-weighted case, improvements for the edge-weighted case were left open in [14]. Our main result is such an improvement and is stated in the following theorem. We denote the degree of a node v in tree T by $\delta_T(v)$. Let the maximum node degree in a tree T be denoted by $\Delta(T)$.

THEOREM 1.1. *There is a polynomial-time approximation algorithm that, given a graph $G = (V, E)$, a nonnegative cost function $c : E \to \mathbb{R}^+$, a constant $B^* \geq 2$, and a parameter $\omega > 0$, computes a spanning tree T s.t.*

1. $\Delta(T) \leq (1 + \omega)bB^* + \log_b n$ *for any arbitrary constant* $b > 1$, *and*
2. $c(T) < (1 + 1/\omega)\,\texttt{opt}_{B^*}$.

For instance, choosing $\omega = 1/2$ and $b = 2$ would yield a tree with degree at most $3B^* + \log_2 n$ and cost at most $3\,\texttt{opt}_{B^*}$.

1.3. Technique: Lagrangean duality. Our algorithm builds on the Lagrangean dual of (IP) resulting from *dualizing* the degree constraints. We denote its value

by $\texttt{opt}_{LD(B)}$.

$$\text{(LD(B))} \qquad \texttt{opt}_{LD(B)} = \max_{\lambda \geq 0} \min_{T \in \mathrm{SP}_G} \left\{ c(T) + \sum_{v \in V} \lambda_v (\delta_T(v) - B) \right\}.$$

For any fixed $\lambda \geq 0$, an optimum integral solution to (IP) is a feasible candidate for attaining the inner minimum above. Since the maximum degree of such a solution is at most B and $\lambda \geq 0$, it follows that $\texttt{opt}_{LD(B)}$ is a lower bound on $\texttt{opt}_B$.

PROPOSITION 1.2 (see [13]). $\texttt{opt}_{LD(B)} \leq \texttt{opt}_B$.

The interesting fact is that $\texttt{opt}_{LD(B)}$ can be computed in polynomial time [13]. The result is a vector λ^B of optimum Lagrangean multipliers on the nodes and a set of optimum trees $\mathcal{O}^{\mathcal{B}}$, all of which achieve the inner minimum for this set of multipliers. In other words, every tree $T^B \in \mathcal{O}^B$ minimizes the following objective:

$$c(T^B) + \sum_{v \in V} \lambda_v^B (\delta_{T^B}(v) - B).$$

Given λ^B, the task of finding a tree T that minimizes the above objective function is called the Lagrangean subproblem of (LD(B)).

Using $c^{\lambda^B}(uv) = c(uv) + \lambda_u^B + \lambda_v^B$, the last expression can be restated as

$$c^{\lambda^B}(T^B) - B \sum_{v \in V} \lambda_v^B. \tag{1.2}$$

Notice that for a given λ^B and B, the second term is a constant. Hence, any MST of G under cost c^{λ^B}, denoted by $\texttt{MST}(G, c^{\lambda^B})$, is a solution for T.

An important feature of our algorithm is to relax the degree constraints slightly from B to $(1+\omega)B$ for some fixed $\omega > 0$ and show that there is a tree $T \in \mathcal{O}^{(1+\omega)B}$ that satisfies the conditions of Theorem 1.1.

This paper is organized as follows: in section 2 we review results on the related *minimum degree spanning tree problem*. In particular, we present the fundamental ideas from [5, 7]. In section 3, we state our algorithm. Finally, we give the analysis of our procedure in section 4.

2. Minimum degree spanning trees. Related to the BMST problem is the problem of minimizing the maximum degree of a spanning tree (MDST) in some graph G. This problem is $\mathcal{NP}$-hard since the *Hamiltonian path* problem is a special case. In fact, it is $\mathcal{NP}$-complete to decide for any $k \geq 2$ whether G contains a spanning tree of maximum degree k [8].

Fürer and Raghavachari presented an approximation algorithm with an additive performance guarantee of one [7]: i.e., they describe a polynomial-time algorithm that finds a spanning tree T of G s.t. $\Delta(T) \leq \Delta^* + 1$, where Δ^* denotes the minimum possible maximum degree over all spanning trees. In the same paper the authors also give a local search algorithm for the MDST problem. This approach yields a tree with maximum degree at most $b\Delta^* + \log_b n$ for any constant $b > 1$. Later, Fischer noted that this procedure can be adapted to find a minimum-cost spanning tree of approximately minimum maximum degree in an edge-weighted graph [5].

The local search algorithms from [5, 7] play an important role in this paper. In this section we show a minor strengthening of these results that is crucial to the analysis of our algorithm.

2.1. A local improvement algorithm. In this section, we explain the basic ideas from the local search algorithm for the MDST problem. We state the algorithm since we use it later. The procedure starts with a spanning tree T and tries to improve it by *swapping* nontree edges against tree edges.

DEFINITION 2.1. *Given a tree T and a nontree edge uv, let $\mathcal{C}(uv)$ be the unique cycle in $T \cup \{uv\}$ and let $wz \in \mathcal{C}(e)$. We call the swap $\langle uv, wz\rangle$ an* improvement for *w if*

$$\max\{\delta_T(u), \delta_T(v)\} + 1 < \delta_T(w).$$

If an edge swap $\langle uv, wz\rangle$ is an improvement step for either w or z, then the maximum degree of the nodes u, v, w, and z decreases as a result of the swap; we call such a swap simply an improvement.

The algorithm in [7] performs improvement steps as long as possible. In fact, it is not hard to see that, starting with an arbitrary tree, the number of possible improvements is finite. We end up with a *locally optimal tree*.

DEFINITION 2.2. *A tree T is called a* locally optimal tree *(LOT) if no vertex degree can be decreased by applying an improvement swap.*

Computing a LOT might be too ambitious a goal, however. In fact, it is not known how to do this in polynomial time. However, the analysis in [7] shows that it is enough to compute a *pseudo-optimal* tree.

DEFINITION 2.3. *A tree T of maximum degree $\Delta(T)$ is called a* pseudo-optimal tree *(POT) if for all vertices v with $\Delta(T) - \lceil \log_b n \rceil \leq \delta_T(v) \leq \Delta(T)$, no improvement step for v is applicable. Here b is an arbitrary constant bigger than one.*

Fischer's adaptation [5] of the algorithm from [7] computes a minimum-cost spanning tree of approximately minimum maximum degree. To obtain his algorithm we have to make two small changes to the procedure from [7]. First, instead of starting with an arbitrary spanning tree, we start with a minimum-cost spanning tree. Second, an improvement step must be *cost neutral*. That is, for an improvement step $\langle uv, wz\rangle$ to be applicable, we must have $c_{uv} = c_{wz}$. Algorithm 1 states the procedure.

ALGORITHM 1. The algorithm `PLocal` computes a POT.

```
1: Given graph G = (V, E) and cost function c : E → R+
2: T ← MST(G, c)
3: while T is not pseudo optimal do
4:    Identify cost neutral improvement ⟨uv, wz⟩
5:    T ← T − wz + uv
6: end while
```

2.2. Analysis and running time. In what follows, we highlight and strengthen the major ideas of the analysis from [5, 7]. The strengthening is due to Éva Tardos [16] and leads to a shorter and simpler proof of Lemma 4.5 than the one that appeared in the extended abstract [11].

The fundamental underlying proof idea for the unweighted problem is based on an averaging argument that we present here. Let a set $W \subseteq V$ be s.t. for a given graph $G = (V, E)$, the graph $G[V - W]$ has t connected components. A spanning tree of G has to connect these t components *and* the nodes from W. We need exactly $t + |W| - 1$ edges going between the nodes of W and the t connected components to accomplish this. Each of these edges must be incident to a node from W. Hence averaging yields a lower bound of $(t + |W| - 1)/|W|$ on the maximum degree Δ^* of T.

PROPOSITION 2.4 (see [7]). *Let W be a set of size w whose removal splits G into t components. Then $\Delta^* \geq \left\lceil \frac{w+t-1}{w} \right\rceil$.*

We now turn to the weighted case, i.e., the *minimum degree minimum-cost spanning tree* problem. The above-mentioned strengthening of the results from [5] is based on the following definitions.

DEFINITION 2.5. *Given an undirected graph $G = (V, E)$ and a nonnegative cost function c on the edges, let $\mathcal{O}^c$ be defined as*

$$\mathcal{O}^c = \{T : T \text{ is an MST under cost } c\}.$$

In the following we will be talking about convex combinations of spanning trees. Hence we introduce some further simplifying notation.

DEFINITION 2.6. *Let $T_c^\alpha = \sum_{T \in \mathcal{O}^c} \alpha_T T$ be a convex combination of minimum-cost spanning trees of G with respect to cost function c, i.e., $\alpha_T \geq 0$ for all T and $\sum_{T \in \mathcal{O}^c} \alpha_T = 1$. We denote the* fractional degree *of vertex v in T_c^α by*

$$\delta_c^\alpha(v) = \sum_{T \in \mathcal{O}^c} \alpha_T \delta_T(v).$$

Finally we define the minimum maximum degree of convex combinations of spanning trees.

DEFINITION 2.7. *Given $G = (V, E)$ and a nonnegative cost function c on the edges, let Δ_c^* denote the minimum maximum degree of any convex combination of minimum-cost spanning trees, i.e.,*

$$\Delta_c^* = \min_{\text{convex comb. } \alpha} \max_{v \in V} \delta_c^\alpha(v).$$

The following easy proposition will be used in the later analysis.

PROPOSITION 2.8 (see [7]). *For any constant $b > 1$ and a tree T, let S_d be the set of nodes that have degree at least d in T. Then, there is a*

$$d \in \{\Delta_T - \lceil \log_b n \rceil + 1, \ldots, \Delta_T\}$$

s.t. $|S_{d-1}| \leq b|S_d|$.

The main theorem is the following.

THEOREM 2.9 (see [5, 7]). *If T is a pseudo-optimal MST, then $\Delta_T < b\Delta_c^* + \lceil \log_b n \rceil$ for any constant $b > 1$. Moreover, a pseudo-optimal MST can be computed in polynomial time.*

Proof. Given a constant $b > 1$, choose d as in Proposition 2.8. That is, we have $|S_{d-1}| \leq b|S_d|$. Recall that S_d contains the nodes of degree at least d in the tree T.

Removing S_d from T leaves us with a forest F. Let $\widehat{G}$ be obtained from G by contracting each connected component of F. It is now easy to see that every minimum-cost spanning tree of G contains a minimum-cost spanning tree of $\widehat{G}$ (e.g., every edge added by Kruskal's algorithm for finding a minimum-cost spanning tree for G is feasible for a minimum-cost spanning tree of $\widehat{G}$ if it were not contracted in the formation of $\widehat{G}$).

Let $(u, v) \in E - T$ be an edge that connects two components of F s.t. $u, v \notin S_{d-1}$, i.e., both u and v have degree at most $d - 2$. We claim that such an edge cannot be included in any MST of $\widehat{G}$. To see that, let $P_{u,v}^T$ be the edges of the unique u, v-path in T and let $\widehat{P_{u,v}^T}$ be the subset of the edges of $P_{u,v}^T$ that are in $\widehat{G}$.

It follows from the pseudo optimality of T that the cost of edge (u, v) must be higher than the cost of each edge from $\widehat{P_{u,v}^T}$. Otherwise, (u, v) can be swapped in place

of another edge of the same or higher cost in $\widehat{P^T_{u,v}}$, and all such edges are incident to at least one node in S_{d-1}, leading to an improvement. This means (u, v) cannot be a part of any MST of $\widehat{G}$. Equivalently, a minimum-cost spanning tree of G must use edges incident to S_{d-1} to connect the components of F and the nodes of S_d.

By the definition of S_d, we know that F has at least

$$|S_d|d - 2(|S_d| - 1) = |S_d|(d-2) + 2$$

trees. This follows from an easy counting argument after observing that every node in S_d has degree at least d in T and there are at most $|S_d| - 1$ edges going between nodes of S_d.

This means that we need at least

$$|S_d|(d-2) + 2 + |S_d| - 1 = |S_d|(d-1) + 1$$

edges to connect up the components of F and the nodes of S_d in *any spanning tree*. By the preceding argument, each of these edges has to be incident to at least one node of degree at least $d-1$ in an MST. Hence the average degree of a node of S_{d-1} in any MST is

$$\frac{|S_d|(d-1)+1}{|S_{d-1}|}.$$

Moreover, the average degree of a node in S_{d-1} in any *convex combination* of MSTs is also at least the above ratio. Since Δ^*_c denotes the minimum possible maximum degree of any fractional MST, it follows, using our choice of index d from Proposition 2.8, that

$$\Delta^*_c > \frac{d-1}{b}.$$

Using the range of d, we obtain $\Delta(T) < b\Delta^*_c + \lceil \log_b n \rceil$. The results in [5, 7] show a lower bound on the degree of any MST. The extension to fractional MSTs is the mentioned strengthening [16] of the previous ideas.

For the running time we follow [7]. Note that each improvement step can be implemented in polynomial time. We need to bound the number of iterations. The proof uses a potential function argument. Define the potential of a vertex v as

$$\Phi(v) = 3^{\delta_T(v)},$$

where T is the current tree. The total potential is the sum over all vertex potentials, that is,

$$\Phi(T) = \sum_{v \in V} \Phi(v).$$

Now, an improvement step in Algorithm 1 improves the degree of a vertex $v \in S_d$ with $\delta_T(v) = d$ and $d \geq \Delta(T) - \lceil \log_b n \rceil + 1$. Hence, the reduction in the potential is going to be at least

$$(3^d + 2 \cdot 3^{d-2}) - 3 \cdot 3^{d-1} = 2 \cdot 3^{d-2}.$$

Using the range of d, we can lower bound the right-hand side of the last equality by

$$3^{\Delta(T) - \log_b n - 1} = \Omega\left(\frac{3^{\Delta(T)}}{n}\right).$$

The potential $\Phi(T)$ of the tree T is at most $n3^{\Delta(T)}$. This implies that the overall decrease of the potential due to the improvement step is

$$\Omega\left(\frac{\Phi(T)}{n^2}\right).$$

In other words, we reduce the potential by at least a polynomial factor in each iteration. In $O(n^2)$ iterations the reduction is by a constant factor. Hence, the algorithm needs $O(n^3)$ improvement steps in total. ∎

3. The BMST algorithm. In this section, we describe our algorithm for the BMST problem. It uses the Lagrangean formulation (LD(B)) from the introduction and Algorithm 1.

The input to our algorithm consists of a graph G, a nonnegative cost function c, a degree bound B^*, and a positive constant ω. Let $B = (1+\omega)B^*$.

ALGORITHM 2. Our algorithm for the BMST problem.

1: Given graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a degree bound $B^* \geq 2$, and a parameter $\omega > 0$
2: $B \leftarrow (1+\omega)B^*$
3: $\lambda^B \leftarrow \texttt{Solve}(LD(B))$
4: $\overline{T} \leftarrow \texttt{PLocal}(G, c^{\lambda^B})$

Since the optimum Lagrange multipliers and pseudo-optimal MSTs can be computed in polynomial time [7, 13], Algorithm 2 runs in polynomial time.

Recall that c^{λ^B} denotes the original cost function c *augmented* by the Lagrangean multipliers λ^B, i.e., $c^{\lambda^B}_{uv} = c_{uv} + \lambda_u + \lambda_v$. We use $\mathcal{O}^B$ to denote the set of all minimum-cost spanning trees of G for cost function c^{λ^B}.

4. Analysis. In this section we prove Theorem 1.1. First, we show that the cost $c(\overline{T})$ of the tree output by Algorithm 2, $\overline{T}$, is small. Then, we prove that $\overline{T}$ has low maximum degree. Our proofs rely on techniques from Lagrangean duality.

4.1. The cost of $\overline{T}$. Recall that $\texttt{opt}_{LD(B)} \leq \texttt{opt}_B$ from Proposition 1.2. Unfortunately, $\texttt{opt}_{LD(B)} = \texttt{opt}_B$ is not true in general. There might be a *duality gap*. However, it can be shown that $\texttt{opt}_{LD(B)}$ equals the optimum objective function value of the relaxation of (IP). The proof is insightful, and hence we present it here.

THEOREM 4.1 (see [13]). $\texttt{opt}_{LD(B)} = \min\{c(T) : T \in SP_G \ \forall v \in V : \delta_T(v) \leq B\}$.

Proof. We can restate (LD(B)) as the following linear program in variables η and λ. Recall that we denote its maximum objective function value by $\texttt{opt}_{LD(B)}$.

$$\begin{aligned} (4.1) \qquad \max \quad & \eta \\ \text{s.t.} \quad & \eta \leq c(T) - \sum_{v \in V} \lambda_v (B - \delta_T(v)) \quad \forall T \in \mathrm{SP}_G\,, \\ & \lambda \geq 0. \end{aligned}$$

The first block of constraints states that η is at most the cost of any spanning tree T of G with respect to the Lagrangean function (1.2). The maximization objective of (4.1) forces η to attain the best possible cost. Writing down the dual of the last

program yields

$$
\begin{aligned}
(4.2) \qquad \min \quad & c\left(\sum_{T\in \mathrm{SP}_G} \alpha_T T\right) \\
\text{s.t.} \quad & \sum_{T\in \mathrm{SP}_G} \alpha_T = 1, \\
& \sum_{T\in \mathrm{SP}_G} \alpha_T \delta_T(v) \le B, \qquad \sum_{T\in \mathrm{SP}_G} \alpha_T = B \quad \forall v \in V, \\
& \alpha \ge 0.
\end{aligned}
$$

Note that $T^\alpha = \sum_{T\in \mathrm{SP}_G} \alpha_T T$ is a convex combination of trees in SP_G. Also, observe that $\sum_{T\in \mathrm{SP}_G} \alpha_T \delta_T(v)$ is precisely the degree $\delta^\alpha(v)$ of this fractional tree at node v. These observations yield the theorem. ∎

The theorem has two nice corollaries that we use. In the following, let λ^B denote the vector of optimum Lagrangean multipliers for (LD(B)). Recall that $\mathcal{O}^B$ is the set of minimum-cost spanning trees under c^{λ^B}.

COROLLARY 4.2. *There exists a convex combination $T^\alpha = \sum_{T\in\mathcal{O}^B} \alpha_T T$ s.t.*

1. $\forall v \in V,\ \delta^\alpha_{c^{\lambda^B}}(v) \le B$ *and*
2. $\lambda^B_v > 0$ *only if* $\delta^\alpha_{c^{\lambda^B}}(v) = B$.

Proof. This follows from complementary slackness applied to the optimum solutions of the dual linear programs (4.1) and (4.2). ∎

The second corollary gives a bound on $\Delta^*_{c^{\lambda^B}}$.

COROLLARY 4.3. $\Delta^*_{c^{\lambda^B}} \le B$.

Proof. By Corollary 4.2, we know that there is a convex combination T^α of trees from $\mathcal{O}^B$ s.t. $\delta^\alpha_{c^{\lambda^B}}(v) \le B$ for all v. Hence

$$\Delta^*_{c^{\lambda^B}} = \min_\alpha \max_{v\in V} \delta^\alpha_{c^{\lambda^B}}(v) \le B. \qquad \blacksquare$$

We now prove that $c(\overline{T})$ is small.

LEMMA 4.4. *For all trees $T \in \mathcal{O}^B$ we have $c(T) < (1 + 1/\omega)\,\mathtt{opt}_{B^*}$.*

Proof. Recall that we defined $B = (1+\omega)B^*$.

The following inequality holds for every $T \in \mathcal{O}^B$:

$$
\begin{aligned}
(4.3) \qquad \sum_{v\in V} \lambda^B_v(\delta_T(v) - B^*) &\le c(T) + \sum_{v\in V} \lambda^B_v(\delta_T(v) - B^*) \\
&\le \mathtt{opt}_{LD(B^*)}.
\end{aligned}
$$

In the first inequality we just added $c(T)$. Note that the right-hand side of the first line is just the Lagrangean objective function (1.2) for B^*. Recall that T is an MST with respect to c^{λ^B}. Moreover, λ^B is a feasible set of multipliers for (LD(B^*)). Hence, the second inequality follows.

We also have

$$
\begin{aligned}
c(T) &= c(T) + \sum_{v\in V} \lambda^B_v(\delta_T(v) - B^*) + \sum_{v\in V} \lambda^B_v(B^* - \delta_T(v)) \\
&\le \mathtt{opt}_{LD(B^*)} + \sum_{v\in V} \lambda^B_v(B^* - \delta_T(v)),
\end{aligned}
$$

where the inequality follows from (4.3). Applying Proposition 1.2 and the fact that $\delta_T(v) \geq 1$ for all $v \in V$ leads to

$$c(T) < \texttt{opt}_{B^*} + B^* \sum_{v \in V} \lambda_v^B.$$

In the remainder of this proof we will derive the inequality $B^* \sum_{v \in V} \lambda_v^B \leq \texttt{opt}_{B^*}/\omega$. This yields the lemma. From Corollary 4.2, we know that there is a convex combination

$$T^\alpha = \sum_{T \in \mathcal{O}^B} \alpha_T T$$

s.t. $\lambda_v^B > 0$ only if $\delta^\alpha_{c^{\lambda^B}}(v) = B$.

We derive a new inequality by summing over all $T \in \mathcal{O}^B$, α_T times the inequality (4.3) for each T. We obtain

$$\sum_{T \in \mathcal{O}^B} \alpha_T \left(\sum_{v \in V} \lambda_v^B (\delta_T(v) - B^*) \right) \leq \texttt{opt}_{LD(B^*)} \sum_{T \in \mathcal{O}^B} \alpha_T. \tag{4.4}$$

The right-hand side is equivalent to $\texttt{opt}_{LD(B^*)}$ because $\sum_{T \in \mathcal{O}^B} \alpha_T = 1$. Reordering the left-hand side yields

$$\sum_{v \in V} \lambda_v^B \left(\left(\sum_{T \in \mathcal{O}^B} \alpha_T \delta_T(v) \right) - B^* \right).$$

Instead of summing over all $v \in V$, it suffices to sum over v, where $\lambda_v^B > 0$. For such v, we have that

$$\delta^\alpha_{c^{\lambda^B}} = \sum_{T \in \mathcal{O}^B} \alpha_T \delta_T(v) = B$$

by Corollary 4.2. Using $B = (1+\omega)B^*$, it follows that the left-hand side of (4.4) is equivalent to

$$\omega B^* \sum_{v \in V} \lambda_v^B,$$

and this finishes the proof of the lemma. □

4.2. The maximum degree of $\overline{T}$.

LEMMA 4.5. $\Delta_{\overline{T}} \leq (1+\omega) b B^* + \lceil \log_b n \rceil$ *for constants* $b > 1$ *and* ω.

Proof. $\overline{T}$ is a pseudo-optimal minimum-cost spanning tree with respect to cost function c^{λ^B}. From Theorem 2.9 we know that

$$\Delta_{\overline{T}} \leq b \Delta^*_{c^{\lambda^B}} + \lceil \log_b n \rceil.$$

An application of Corollary 4.3, noting that $B = (1+\omega)B^*$, yields the lemma. □

5. Conclusions.

5.1. Summary and remarks. In this paper we developed an improved approximation algorithm for the degree-bounded minimum spanning tree problem. For a positive constant B^* and an n-node graph, our method computes a spanning tree whose cost is at most a constant factor worse than the cost of the optimum degree-B^*-bounded minimum spanning tree. Additionally, we proved that the maximum degree of the resulting tree is $O(B^* + \log n)$. Our procedure solves a Lagrangean relaxation of the BMST integer program for slightly relaxed degree constraints ($(1+\omega)B^*$ instead of B^*). We showed how this slack helps to prove low cost of the resulting tree. Our algorithm also makes use of a local search technique from [5, 7]. We showed how a slight strengthening of the results in [5, 7] can be used to prove low maximum degree of the resulting tree.

5.2. Extensions and open questions. An interesting open question is whether our results extend to the case of Steiner trees and general Steiner networks. The central difficulty of such an extension stems from the fact that, in the Steiner case, the subproblem that arises from dualizing the degree constraints (the minimum-cost Steiner tree problem) itself is $\mathcal{NP}$-hard.

Another avenue for extending our work is to examine if our approach is capable of handling individual node degrees. In the current version, node degrees are assumed to be uniform. Lemma 4.5 relies on the pseudo optimality of tree $\overline{T}$ from Algorithm 2 and on results from [5, 7]. These results do not apply to nonuniform degrees. Is there an extension of the known MDST algorithms to handle individual degree bounds?

We believe that the techniques used in this paper can be generalized to apply to a broader class of multicriteria problems. A central point in the development of a more general framework is the identification of key properties of suitable optimization problems; in the BMST problem, the dualization of the degree constraints yields a tractable subproblem. Furthermore, the compact form of the objective function of this subproblem proved to be a key for the analysis.

Acknowledgment. We thank Éva Tardos for permitting us to include her simpler proof of Lemma 4.5 in the paper.

REFERENCES

[1] F. Bauer and A. Varma, *Degree-constrained multicasting in point-to-point networks*, in Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95), Los Alamitos, CA, 1995, pp. 369–376.

[2] S. Chopra, *On the spanning tree polyhedron*, Oper. Res. Lett., 8 (1989), pp. 25–29.

[3] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, *An architecture for wide-area multicast routing*, in Proceedings of the 1994 ACM SIGCOMM Conference, London, UK, 1994, pp. 126–135.

[4] J. Edmonds, *Optimum branchings*, J. Res. Nat. Bur. Standards Sect. B, 71 (1967), pp. 233–240.

[5] T. Fischer, *Optimizing the Degree of Minimum Weight Spanning Trees*, Technical report 93-1338, Department of Computer Science, Cornell University, Ithaca, NY, 1993.

[6] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang, *A reliable multicast framework for light-weight sessions and application level framing*, IEEE/ACM Trans. Networking, 5 (1997), pp. 784–803.

[7] M. Fürer and B. Raghavachari, *Approximating the minimum-degree Steiner tree to within one of optimal*, J. Algorithms, 17 (1994), pp. 409–423.

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

[9] S. Khuller, B. Raghavachari, and N. Young, *Low-degree spanning trees of small weight*, SIAM J. Comput., 25 (1996), pp. 355–368.

[10] P. N. Klein and R. Ravi, *A nearly best-possible approximation for node-weighted Steiner trees*, J. Algorithms, 19 (1995), pp. 104–115.
[11] J. Könemann and R. Ravi, *A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Portland, OR, 2000.
[12] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Bicriteria network design problems*, J. Algorithms, 28 (1998), pp. 142–171.
[13] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.
[14] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, *Many birds with one stone: Multi-objective approximation algorithms*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 438–447.
[15] A. S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice–Hall, Upper Saddle River, NJ, 1996.
[16] É. Tardos, *personal communication*, 2000.
[17] W. De Zhong, J. Kaniyil, and Y. Onozato, *A copy network with shared buffers for large-scale multicast ATM switching*, IEEE/ACM Trans. Networking, 1 (1993), pp. 157–165.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1794–1813

MAINTAINING STREAM STATISTICS OVER SLIDING WINDOWS*

MAYUR DATAR†, ARISTIDES GIONIS†, PIOTR INDYK‡, AND RAJEEV MOTWANI†

Abstract. We consider the problem of maintaining aggregates and statistics over data streams, with respect to the last N data elements seen so far. We refer to this model as the *sliding window* model. We consider the following basic problem: Given a stream of bits, maintain a count of the number of 1's in the last N elements seen from the stream. We show that, using $O(\frac{1}{\epsilon}\log^2 N)$ bits of memory, we can estimate the number of 1's to within a factor of $1+\epsilon$. We also give a matching lower bound of $\Omega(\frac{1}{\epsilon}\log^2 N)$ memory bits for any deterministic or randomized algorithms. We extend our scheme to maintain the sum of the last N positive integers and provide matching upper and lower bounds for this more general problem as well. We also show how to efficiently compute the L_p norms ($p \in [1,2]$) of vectors in the sliding window model using our techniques. Using our algorithm, one can adapt many other techniques to work for the sliding window model with a multiplicative overhead of $O(\frac{1}{\epsilon}\log N)$ in memory and a $1+\epsilon$ factor loss in accuracy. These include maintaining approximate histograms, hash tables, and statistics or aggregates such as sum and averages.

Key words. statistics, data streams, sliding windows, approximation algorithms

AMS subject classifications. 68P05, 68Q25, 68R01, 68W25

PII. S0097539701398363

1. Introduction.

Traditional database management systems (DBMSs) expect all data to be managed within some form of persistent *data sets*. For many recent applications, the concept of a *data stream*, possibly infinite, is more appropriate than a data set. By nature, a stored data set is appropriate when significant portions of the data are queried again and again, and updates are small and/or relatively infrequent. In contrast, a data stream is appropriate when the data is changing constantly (often exclusively through insertions of new elements), and it is either unnecessary or impractical to operate on large portions of the data multiple times.

One of the challenging aspects of processing over data streams is that, while the length of a data stream may be unbounded, making it impractical or undesirable to store the entire contents of the stream, for many applications, it is still important to retain some ability to execute queries that reference past data. For example, in order to detect fraudulent credit card transactions, it is useful to be able to detect when the pattern of recent transactions for a particular account differs significantly from the earlier transactional history of that account. In order to support queries of this sort using a bounded amount of storage (either in memory or in a traditional DBMS), it is necessary to devise techniques for storing summary or synoptic information about previously seen portions of data streams. Generally there is a tradeoff between the

*Received by the editors November 18, 2001; accepted for publication (in revised form) June 8, 2002; published electronically October 18, 2002. A preliminary version of this paper appeared in *Proceedings of the Thirteenth Annual ACM–SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 2002.

http://www.siam.org/journals/sicomp/31-6/39836.html

†Department of Computer Science, Stanford University, Stanford, CA 94305 (datar@cs.stanford.edu, gionis@cs.stanford.edu, rajeev@cs.stanford.edu). The research of the first author was supported in part by NSF grant IIS-0118173 and a Microsoft Graduate Fellowship. The research of the second author was supported in part by NSF grant IIS-0118173 and a Microsoft Graduate Fellowship. The research of the fourth author was supported in part by NSF grant IIS-0118173, an Okawa Foundation research grant, and Veritas.

‡MIT Laboratory for Computer Science, 545 Technology Square, NE43-373, Cambridge, MA 02139 (indyk@theory.lcs.mit.edu).

size of the summaries and the ability to provide precise answers to queries involving past data.

We consider the problem of maintaining statistics over streams with regard to the last N data elements seen so far. We refer to this model as the *sliding window* model. We identify a simple counting problem whose solution is a prerequisite for efficient maintenance of a variety of more complex statistical aggregates: Given a stream of bits, maintain a count of the number of 1's in the last N elements seen from the stream. We show that, using $O(\frac{1}{\epsilon}\log^2 N)$ bits of memory, we can estimate the number of 1's to within a factor of $1+\epsilon$. We also give a matching lower bound of $\Omega(\frac{1}{\epsilon}\log^2 N)$ memory bits for any deterministic or randomized algorithm. We extend our scheme to maintain the sum of the last N positive integers and provide matching upper and lower bounds for this more general problem as well.

We also show how to efficiently compute the L_p norms ($p \in [1,2]$) of vectors in the sliding window model using our techniques. Using our algorithm, one can adapt many other techniques to work for the sliding window model with a multiplicative overhead of $O(\frac{1}{\epsilon}\log N)$ in memory and a $1+\epsilon$ factor loss in accuracy. These include maintaining approximate histograms, maintaining hash tables, and maintaining statistics and aggregates such as sum and average. Our techniques are simple and easy to implement. We expect that it will be an attractive choice of implementation for streaming applications.

1.1. Motivation, model, and related work. Several applications naturally generate data streams as opposed to data sets. In telecommunications, for example, *call records* are generated continuously. Typically, most processing is done by examining a call record once or operating on a "window" of recent call records (e.g., to update customer billing information), after which records are archived and not examined again. For example, Cortes et al. [2] report working with AT&T long distance call records, consisting of 300 million records per day for 100 million customers. A second application is *network traffic engineering*, in which information about current network performance—latency, bandwidth, etc.—is generated online and is used to monitor and adjust network performance dynamically [7, 16]. In this application, it is generally both impractical and unnecessary to process anything but the most recent data.

There are other traditional and emerging applications in which data streams play an important and natural role, e.g., web tracking and personalization (where the data streams are web log entries or click-streams), medical monitoring (vital signs, treatments, and other measurements), sensor databases, and financial monitoring, to name but a few. There are also applications in which traditional (nonstreaming) data is treated as a stream due to performance constraints. In data mining applications, for example, the volume of data stored on disk is so large that it is only possible to make one pass (or perhaps a very small number of passes) over the data [12, 11]. The objective is to perform the required computations using the stream generated by a single scan of the data, using only a bounded amount of memory and without recourse to indexes, hash tables, or other precomputed summaries of the data. Another example along these lines occurs when data streams are generated as intermediate results of pipelined operators during evaluation of a query plan in an SQL database; without materializing some or all of the temporary results, only one pass on the data is possible [3].

In most of these applications, the goal is to make decisions based on the statistics or models gathered over the "recently observed" data elements. For example, one

might be interested in gathering statistics about packets processed by a set of routers over the last day. Moreover, we would like to maintain these statistics in a continuous fashion. This gives rise to the *sliding window model*: Data elements arrive at every instant; each data element expires after exactly N time steps; and the portion of data that is relevant to gathering statistics or answering queries is the set of the last N elements to arrive. The sliding window refers to the window of active data elements at a given time instant.

Previous work [1, 5, 13] on stream computations addresses the problems of approximating frequency moments and computing the L_p differences of streams. There has also been work on maintaining histograms [14, 10]. While Jagadish et al. [14] address the off-line version of computing optimal histograms, Guha and Koudas [10] provide a technique for maintaining near optimal time-based histograms in an on-line fashion over streaming data. The queries that are supported by histograms constructed in the latter work are range or point queries over the time attribute. In the earlier work, the underlying model is that all of the data elements seen so far are relevant. Recent work by Gilbert et al. [8] considers, among other things, the problem of maintaining *aged aggregates* over data streams. For a data stream $\ldots, a^{(-2)}, a^{(-1)}, a^{(0)}$, where $a^{(0)}$ is the most recently seen data element, the λ-*aging* aggregate is defined as $\lambda a^{(0)} + \lambda(1-\lambda)a^{(-1)} + \lambda(1-\lambda)^2 a^{(-2)} + \cdots$. Aged aggregate queries tend to get asked in the context of telecommunications data. While aging is one technique to discount for the staleness of certain data elements, we believe that the sliding window model is also important since, for most applications, one is not interested in gathering statistics over outdated data. For instance, in network management, depending upon the specific application, we may not want data that is a month old or a year old to affect our decisions. Maintaining statistics like sum/average, histograms, hash tables, frequency moments, and L_p differences over sliding windows is critical to most applications. To our knowledge, there has been no previous work that addresses these problems for the sliding window model.

1.2. Summary of results. We focus completely on the sliding window model for data streams. We formulate a basic counting problem whose solution can be used as a building block for solving most of the problems mentioned earlier.

PROBLEM 1 (BASICCOUNTING). *Given a stream of data elements, consisting of* 0*'s and* 1*'s, maintain at every time instant the count of the number of* 1*'s in the last* N *elements.*

It is easy to verify that an exact solution requires $\Theta(N)$ bits of memory. (Note that we measure space complexity in terms of the number of bits rather than the number of memory words.) For most applications, it is prohibitive to use $\Omega(N)$ memory. For instance, consider the network management application, where a large number of data packets pass through a router every second. However, in most applications, it suffices to produce an approximate answer. Thus our goal is to provide a good approximation using $o(N)$ memory.

It is interesting to observe why naïve schemes do not suffice for producing approximate answers with low memory requirement. For instance, consider the scheme in which we maintain a simple counter which is incremented upon the arrival of a data element, which is 1. The problem is that an old data element expires at every time instant, but we have no way of knowing whether that was a 0 or 1 and whether we should decrement the counter. It is also natural to consider random sampling. Just maintaining a sample of the window elements will fail in the case where the 1's are relatively sparse.

Another approach is to maintain histograms. While this is the approach that we follow, we argue that the known histogram techniques will not work. A histogram technique is characterized by the policy used to maintain the bucket boundaries. We would like to build time-based histograms in which every bucket represents a contiguous time interval and maintains the number of 1's that arrived in that interval. As with all histogram techniques, when a query is presented, we may have to interpolate in some bucket to estimate the answer because a proper subset of the buckets' elements may have expired. Let us consider some schemes of bucketizing and see why they will not work. The first scheme that we consider is that of dividing into k equiwidth buckets. The problem is that the distribution of 1's in the buckets may be nonuniform. We will incur large error when the interpolation takes place in buckets with a majority of the 1's. This suggests another scheme, in which we use buckets of nonuniform width, so as to ensure that each bucket has a near-uniform number of 1's. The problem is that the total number of 1's in the sliding window could change dramatically with time, and the current buckets may turn out to have more or less than their fair share of 1's as the window slides forward. Our solution is a form of a histogram which avoids these problems by using a set of well-structured and nonuniform bucket sizes.

In section 2, we provide a solution for BasicCounting which uses $O(\frac{1}{\epsilon}\log^2 N)$ bits of memory (equivalently, $O(\frac{1}{\epsilon}\log N)$ buckets of size $O(\log N)$) and provides an estimate of the answer at every instant that is within a $1+\epsilon$ factor of the actual answer. Moreover, our algorithm does not require an a priori knowledge of N and caters to the possibility that the window size can be changed dynamically. Our algorithm is guaranteed to work with $O(\log^2 N)$ memory as long as the window size is bounded by N. The algorithm takes $O(\log N)$ worst-case time to process each new data element's arrival but only $O(1)$ amortized time per element. Count queries can be processed in $O(1)$ time. The algorithm itself is relatively simple and easy to implement.

Section 3 presents a matching lower bound. We show that any approximation algorithm (deterministic or randomized) for BasicCounting with relative error $1+\epsilon$ must use $\Omega(\frac{1}{\epsilon}\log^2 N)$ bits of memory. This proves that our algorithm is optimal in terms of memory usage.

In section 4, we extend the technique to handle data elements with positive integer values, instead of just binary values; this is referred to as the Sum problem. We provide matching upper and lower bounds on the memory usage for this general problem as well.

In section 5, we show how our schemes extend to a model which is more suited for real-life applications and also explore some ideas for reducing the memory requirements.

In section 6, we show that we can use our techniques along with the sketching techniques of [13] to efficiently maintain the L_p ($p \in [1,2]$) norms of vectors in the sliding window model.

Finally, section 7 provides a brief discussion of the application of the BasicCounting and Sum algorithms to adapting several other problems in the sliding window model, such as maintaining histograms, hash tables, and statistics or aggregates such as averages/sums. The reduction of these problems to BasicCounting entails a multiplicative overhead of $O(\frac{1}{\epsilon}\log N)$ in memory and a $1+\epsilon$ factor loss in accuracy. This serves to illustrate the usefulness of focusing on the BasicCounting problem. We also discuss upper and lower bounds for other problems such as main-

taining min/max, distinct values estimation, and maintaining sum in the presence of positive and negative values.

2. Algorithm for BasicCounting. Our approach toward solving the Basic-Counting problem is to maintain a histogram that records the timestamp of selected 1's that are *active* in that they belong to the last N elements. We call this histogram the exponential histogram (EH) for reasons that will be clear later. Before getting into the details of our algorithms, we need to introduce some notation.

We follow the conventions illustrated in Figure 1. In particular, we assume that new data elements are coming from the right and the elements at the left are ones already seen. Note that each data element has an *arrival time*, which increments by one at each arrival, with the leftmost element considered to have arrived at time 1. But, in addition, we employ the notion of a *timestamp*, which corresponds to the position of an *active* data element in the current window. We timestamp the active data elements from right to left, with the most recent element being at position 1. Clearly, the timestamps change with every new arrival, and we do not wish to make explicit updates. A simple solution is to record the arrival times in a wraparound counter of $\log N$ bits, and then the timestamp can be extracted by comparison with the counter value of the current arrival. As mentioned earlier, we concentrate on the 1's in the data stream. When we refer to the kth 1, we mean the kth most recent 1 encountered in the data stream.

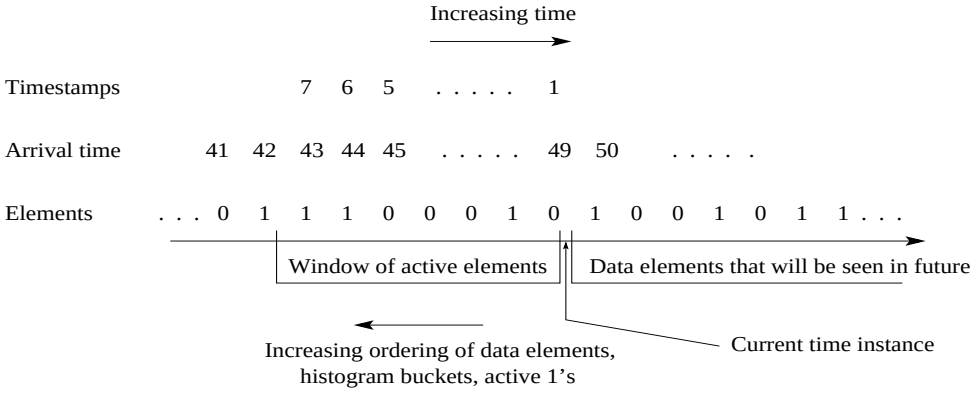


Fig. 1. *An illustration for the notation and conventions followed.*

For an illustration of this notation, consider the situation presented in Figure 1. The current time instant is 49, and the most recent arrival is a zero. The element with arrival time 48 is the most recent 1 and has timestamp 2 since it is the second most recent arrival in the current window. The element with arrival time 44 is the second most recent 1 and has timestamp 6.

We will maintain histograms for the active 1's in the data stream. For every bucket in the histogram, we keep the timestamp of the most recent 1 (called *timestamp*), and the number of 1's (called *bucket size*). For example, in our figure, a bucket with timestamp 2 and size 2 represents a bucket that contains the two most recent 1's with timestamps 2 and 6. Note that the timestamp of a bucket increases as new elements arrive. When the timestamp of a bucket expires (reaches $N + 1$), we are no longer interested in data elements contained in it, so we drop that bucket and reclaim its memory. If a bucket is still active, we are guaranteed that it contains at least a single

1 that has not expired. Thus, at any instant, there is at most one bucket (the last bucket) containing 1's which may have expired. At any time instant, we may produce an estimate of the number of active 1's as follows. For all but the last bucket, we add the number of 1's that are in them. For the last bucket, let C be the count of the number of 1's in that bucket. The actual number of active 1's in this bucket could be anywhere between 1 and C, and so we estimate it to be $C/2$. We obtain the following fact.

FACT 1. *The absolute error in our estimate is at most $C/2$, where C is the size of the last bucket.*

Note that, for this approach, the window size does not have to be fixed a priori at N. Given a window size S, we do the same thing as before except that the last bucket is the bucket with the largest timestamp less than S.

2.1. The approximation scheme. We now define the EHs and present a technique to maintain them so as to guarantee count estimates with relative error at most ϵ for any $\epsilon > 0$. Define $k = \lceil \frac{1}{\epsilon} \rceil$, and assume that $\frac{k}{2}$ is an integer; if $\frac{k}{2}$ is not an integer, we can replace $\frac{k}{2}$ by $\lceil \frac{k}{2} \rceil$ without affecting the basic results.

As per Fact 1, the absolute error in the estimate is $C/2$, where C is the size of the last bucket. Let the buckets be numbered from right to left with the most recent bucket being numbered 1. If C_i is the size of the ith bucket, we know that the true count is at least $1 + \sum_{i=1}^{m-1} C_i$ since the last bucket contains at least one 1, and the remaining buckets contribute exactly their size to the total count. Note that m is the index of the last bucket. Thus the relative estimation error is at most $(C_m/2)/(1 + \sum_{i=1}^{m-1} C_i)$. We will ensure that the relative error is at most $1/k$ by maintaining the following invariant.

INVARIANT 1. *At all times, the bucket sizes $C_1, \ldots, C_m$ are such that, for all $j \leq m$, we have $C_j/2(1 + \sum_{i=1}^{j-1} C_i) \leq \frac{1}{k}$.*

Let $N' \leq N$ be the number of 1's that are active at any instant. Then the bucket sizes must satisfy $\sum_{i=1}^{m} C_i \geq N'$. In order to satisfy this and Invariant 1 with as few buckets as possible, we maintain buckets with exponentially increasing sizes so as to satisfy the following second invariant.

INVARIANT 2. *At all times, the bucket sizes are nondecreasing, i.e., $C_1 \leq C_2 \leq \cdots \leq C_{m-1} \leq C_m$. Further, the bucket sizes are constrained to the following: $\{1, 2, 4, \ldots, 2^{m'}\}$ for some $m' \leq m$ and $m' \leq \log \frac{2N}{k} + 1$. For every bucket size other than the size of the last bucket, there are at most $\frac{k}{2} + 1$ and at least $\frac{k}{2}$ buckets of that size.*

Let $C_j = 2^r$ be the size of the jth bucket. If Invariant 2 is satisfied, then we are guaranteed that there are at least $\frac{k}{2}$ buckets, each of sizes $1, 2, 4, \ldots, 2^{r-1}$, which have indexes less than j. Consequently, $C_j \leq \frac{2}{k}(1 + \sum_{i=1}^{j-1} C_i)$. It follows that, if Invariant 2 is satisfied, then Invariant 1 is automatically satisfied. If we maintain Invariant 2, it is easy to see that, to cover all the active 1's, we would require no more than $m \leq (\frac{k}{2} + 1)(\log(\frac{2N}{k} + 1) + 1)$ buckets. Associated with the bucket is its size and a timestamp. The bucket size takes at most $\log N$ values, and hence we can maintain them using $\log \log N$ bits. Since a timestamp requires $\log N$ bits, the total memory requirement of each bucket is $\log N + \log \log N$ bits. Therefore, the total memory requirement (in bits) for an EH is $O(\frac{1}{\epsilon} \log^2 N)$. It is implied that, by maintaining Invariant 2, we are guaranteed the desired relative error and memory bounds.

The query time for the EH is $O(1)$. We achieve this by maintaining two counters: one for the size of the last bucket (LAST) and one for the sum of the sizes of all buckets (TOTAL). The estimate itself is TOTAL minus half of LAST. Both counters can be

updated in $O(1)$ time for every data element. The following is a detailed description of the update algorithm.

ALGORITHM INSERT.

1. When a new data element arrives, calculate the new expiry time. If the timestamp of the last bucket indicates expiry, delete that bucket, and update the counter LAST containing the size of the last bucket and the counter TOTAL containing the total size of the buckets.
2. If the new data element is 0, ignore it; otherwise, create a new bucket with size 1 and the current timestamp, and increment the counter TOTAL.
3. Traverse the list of buckets in order of increasing sizes. If there are $\frac{k}{2}+2$ buckets of the same size, merge the oldest two of these buckets into a single bucket of double the size. (A merger of buckets of size 2^r may cause the number of buckets of size 2^{r+1} to exceed $\frac{k}{2}+1$, leading to a cascade of such mergers.) Update the counter LAST if the last bucket is the result of a new merger.

Example 1. We illustrate the execution of the algorithm for 10 steps, where, at each step, the new data element is 1. The numbers indicate the bucket sizes from left to right, and we assume that $\frac{k}{2}=1$.

32, 32, 16, 8, 8, 4, 2, 1
32, 32, 16, 8, 8, 4, 4, 2, 1, 1 (new 1 arrived)
32, 32, 16, 8, 8, 4, 4, 2, 1, 1, 1 (new 1 arrived)
32, 32, 16, 8, 8, 4, 4, 2, 2, 1 (merged the older 1's)
32, 32, 16, 8, 8, 4, 4, 2, 2, 1, 1 (new 1 arrived)
32, 32, 16, 8, 8, 4, 4, 2, 2, 1, 1, 1 (new 1 arrived)
32, 32, 16, 8, 8, 4, 4, 2, 2, 2, 1 (merged the older 1's)
32, 32, 16, 8, 8, 4, 4, 4, 2, 1 (merged the older 2's)
32, 32, 16, 8, 8, 8, 4, 2, 1 (merged the older 4's)
32, 32, 16, 16, 8, 4, 2, 1 (merged the older 8's)

Merging two buckets corresponds to creating a new bucket whose size is equal to the sum of the sizes of the two buckets and whose timestamp is the timestamp of the more recent of the two buckets, i.e., the timestamp of the bucket that is to the right. A merger requires $O(1)$ time. Moreover, while cascading may require $\Theta(\log \frac{2N}{k})$ mergers upon the arrival of a single new element, standard arguments allow us to argue that the amortized cost of mergers is $O(1)$ per new data element. It is easy to see that the above algorithm maintains Invariant 2. We obtain the following theorem.

THEOREM 1. *The EH algorithm maintains a data structure which can give an estimate for the* BASICCOUNTING *problem with relative error at most ϵ using at most $(\frac{k}{2}+1)(\log(\frac{2N}{k}+1)+1)$ buckets, where $k=\lceil\frac{1}{\epsilon}\rceil$. The memory requirement is $\log N+\log\log N$ bits per bucket. The arrival of each new element can be processed in $O(1)$ amortized time and $O(\log N)$ worst-case time. At each time instant, the data structure provides a count estimate in $O(1)$ time.*

If, instead of maintaining a timestamp for every bucket, we maintain a timestamp for the most recent bucket and maintain the difference between the timestamps for the successive buckets, then we can reduce the total memory requirement to $O(k\log^2\frac{N}{k})$.

3. Lower bounds. We provide a lower bound which verifies that the EH algorithm is optimal in its memory requirement. We start with a deterministic lower bound of $\Omega(k\log^2\frac{N}{k})$.

THEOREM 2. *Any deterministic algorithm that provides an estimate for the* BASICCOUNTING *problem at every time instant with relative error less than $\frac{1}{k}$ for*

some integer $k \leq 4\sqrt{N}$ *requires at least* $\frac{k}{16}\log^2\frac{N}{k}$ *bits of memory.*

The proof argument will go as follows: We will show that there are a large number of arrangements of 0's and 1's such that any deterministic algorithm which provides estimates with small relative error has to differentiate between every pair of these arrangements. The number of memory bits required by such an algorithm must therefore exceed the logarithm of the number of arrangements. The above argument is formalized by the following lemma.

LEMMA 1. *For* $k/4 \leq B \leq N$, *there exist* $L = \binom{B}{k/4}^{\lfloor \log \frac{N}{B} \rfloor}$ *arrangements of* 0*'s and* 1*'s of length* N *such that any deterministic algorithm for* BASICCOUNTING *with relative error less than* $\frac{1}{k}$ *must differentiate between any two of the arrangements.*

Proof. We partition a window of size N into blocks of size $B, 2B, 4B, \ldots, 2^jB$, from right to left, for $j = \lfloor \log \frac{N}{B} \rfloor - 1$. Consider the ith block of size 2^iB, and subdivide it into B contiguous subblocks of size 2^i. For each block, we choose $\frac{k}{4}$ subblocks and populate them with 1's, placing 0's in the remaining positions. In every block, there are $\binom{B}{k/4}$ possible ways to place the 1's, and therefore the total number of distinct arrangements is $L = \binom{B}{k/4}^{\lfloor \log N/B \rfloor}$.

We now argue that any deterministic algorithm for BASICCOUNTING with relative error less than $\frac{1}{k}$ must differentiate between any pair of these arrangements. In other words, if there exists a pair of arrangements A_x, A_y such that a deterministic algorithm does not differentiate between them, then, after some time interval, the two arrangements will have different answers to the BASICCOUNTING problem, and the algorithm will give a relative error of at least $\frac{1}{k}$ for one of them. To this end, we will assume that the algorithm is presented with one of these L arrangements of length N, followed by a sequence of all 0's of length N.

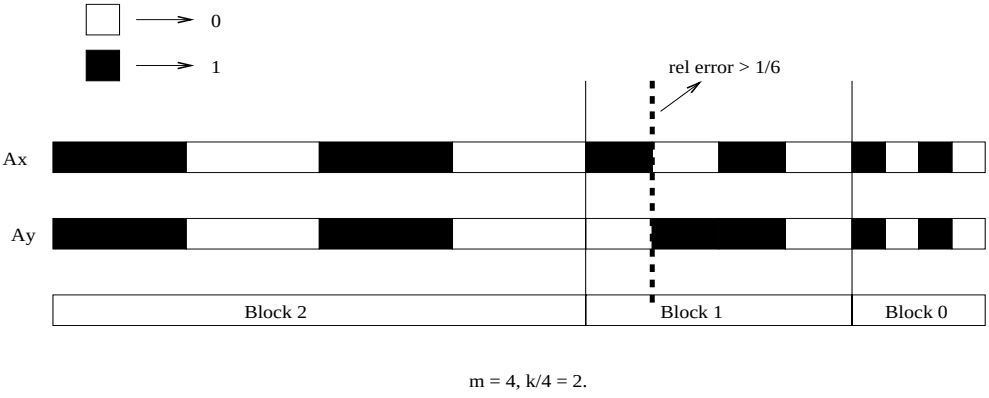


FIG. 2. *A pair of arrangements that should be differentiated by any deterministic algorithm with relative error less than* 1/8.

Refer to Figure 2 for an illustration of a pair of arrangements that should be differentiated by any deterministic algorithm with relative error less than $\frac{1}{8}$.

Consider an algorithm that does not differentiate between two of the above arrangements A_x and A_y. We will use the numerical sequences $x_0, x_1, \ldots, x_j$ and $y_0, y_1, \ldots, y_j$ for $j = \lfloor \log \frac{N}{B} \rfloor - 1$ to encode the two arrangements. The ith number in the sequence specifies the choice of the $k/4$ subblocks from the ith block which are populated with 1's. The two sequences must be distinct since the two arrangements

being encoded are distinct. Let d be an index of a point where the two sequences differ, i.e., $x_d \neq y_d$. Then the two arrangements have a different choice of $k/4$ subblocks in the dth block. Number the subblocks within block d from right to left, and let h be the highest numbered subblock that is chosen for one of the arrangements (say, A_x) but not for the other (A_y). Consider the time instant when this subblock h expires. At that instant, the number of active subblocks in block d for arrangement A_x is c, where $c+1 \leq k/4$, while the number of active subblocks in block d for A_y is $c+1$. Since the arrangements are followed by a sequence of 0's, at this time, the correct answer for A_x is $c2^d + \frac{k}{4}(2^d - 1)$, while, for A_y, the correct answer is $(c+1)2^d + \frac{k}{4}(2^d - 1)$. Thus the algorithm will give an absolute error of at least 2^{d-1} for one of the arrangements, which translates to a relative error of $\frac{1}{k}$ at that point in time. □

To prove Theorem 2, observe that, if we choose $B = \sqrt{Nk}$, then $\log L \geq \frac{k}{16} \log^2 \frac{N}{k}$. We also extend the lower bound on the space complexity to randomized algorithms.

As a reminder, a *Las Vegas algorithm* is a randomized algorithm that always produces the correct answer, although the running time of the algorithm may vary with the different random choices that the algorithm makes. On the other hand, a *Monte Carlo algorithm* is a randomized algorithm that sometimes produces an incorrect solution. We obtain the following lower bounds for these two classes of algorithms.

THEOREM 3. *Any randomized Las Vegas algorithm for* BASICCOUNTING *with relative error less than* $\frac{1}{k}$ *for some integer* $k \leq 4\sqrt{N}$ *requires at least* $\frac{k}{16} \log^2 \frac{N}{k}$ *bits of memory.*

Proof. Define an algorithm A to be ϵ-correct for an input instance I if the value returned by A on input I has relative error less than ϵ. The Yao minimax principle [15] implies that the expected space complexity of the optimal ϵ-correct deterministic algorithm for an arbitrarily chosen input distribution $\mathbf{p}$ is a lower bound on the expected space complexity of the optimal ϵ-correct Las Vegas randomized algorithm. Consider the uniform distribution over the input arrangements in Lemma 1. Then any deterministic algorithm that is ϵ-correct for all of these instances must differentiate between any two distinct arrangements. As a result, the expected space complexity of an optimal deterministic algorithm on this distribution is at least equal to the optimal coding length for the probability distribution. Since the coding length is at least equal to the entropy of the distribution, we get the same lower bound (logarithm of the number of instances) as in the case of a deterministic algorithm. This proves the generalization of Theorem 2 to Las Vegas randomized algorithms. □

THEOREM 4. *Any randomized Monte Carlo algorithm for* BASICCOUNTING *with relative error less than* $\frac{1}{k}$ *for some integer* $k \leq 4\sqrt{N}$ *with probability at least* $1 - \delta$ *(for* $\delta < \frac{1}{2}$*) requires at least* $\frac{k}{64} \log^2 \frac{N}{k} - \log(1 - \delta)$ *bits of memory.*

Proof. We use the analogous version of Yao's minimax principle for Monte Carlo randomized algorithms [15] to establish the lower bound for Monte Carlo algorithms. Consider a deterministic algorithm that is ϵ-correct with probability at least $1 - \delta$ for some $\delta < \frac{1}{2}$. As before, the input distribution $\mathbf{p}$ that we consider is the uniform distribution over all of the arrangements defined in Lemma 1. Since the deterministic algorithm is ϵ-correct with probability at least $1 - \delta$, it is ϵ-correct for at least a $1 - \delta$ fraction of the inputs. Thus, by arguments similar to those in the previous theorem, we get the same lower bound except for an additive loss of $\log(1 - \delta)$ and a multiplicative loss of $\frac{1}{4}$. Asymptotically, the lower bound does not change. □

4. Beyond 0's and 1's. Consider now the extension of BASICCOUNTING to the case where the elements are positive integers.

PROBLEM 2 (SUM). *Given a stream of data elements that are positive integers in the range* $[0 \ldots R]$*, maintain at every time instant the sum of the last* N *elements.*

One obvious way to solve the above problem would be to separately maintain a sliding window sum for each of the $\log R$ bit positions using an EH from section 2. As before, let $k = \lceil \frac{1}{\epsilon} \rceil$. The memory requirement for this approach would be $O(k \log^2 N \log R)$ bits. Next, we present a technique that has a smaller space requirement.

We assume that $\log R = o(N)$. This is a realistic assumption which simplifies our calculations. We generalize the EH to this setting as follows. View the arrival of a data element of value v as the arrival of v data elements with value 1 all at the same time, and employ the same insertion procedure as before. Note that the algorithm in section 2 does not require distinct timestamps; they are required only to be nondecreasing. While earlier there could be at most N active 1's, now there could be as many as NR. The results in section 2 imply that the EH will require at most $(\frac{k}{2} + 1)(\log(\frac{2NR}{k} + 1) + 1)$ buckets. Now each bucket will require $\log N + \log(\log N + \log R)$ bits of memory to store the timestamp and the size of the bucket. Note that there are N distinct timestamps at any point (as before) but that the bucket sizes could take on $\log N + \log R$ distinct values. Thus the number of memory bits required is

$$\left(\frac{k}{2} + 1\right)\left(\log\left(\frac{2NR}{k} + 1\right) + 1\right)(\log N + \log(\log N + \log R)) = O\left(\frac{1}{\epsilon}(\log N + \log R)(\log N)\right).$$

The only catch appears to be that we need $\Omega(R)$ time per insertion. The rest of the section is devoted to devising a scheme that requires only $O(\frac{\log R}{\log N})$ amortized time and $O(\log N + \log R)$ worst-case time per insertion. Note that, if $R = O(poly(N))$, then the amortized insertion time becomes $O(1)$, and the worst-case time becomes $O(\log N)$.

Let S be the total size of the buckets at some time instant. For $j \leq \log(\frac{2NR}{k} + 1)$, let $k_0, k_1, \ldots, k_j$ be a sequence in which k_i denotes the number of buckets of size 2^i. Then $S = \sum_{i=0}^{j} k_i 2^i$. By Invariant 2, we have $l \leq k_i \leq l+1$ for $i < j$ and $1 \leq k_j \leq l+1$, where $l = \frac{k}{2} = \frac{\lceil \frac{1}{\epsilon} \rceil}{2} \geq 1$. Given $l \geq 1$ and S, a sequence $k_0, k_1, \ldots, k_j$ satisfying the above conditions is called an l-canonical representation of S. The algorithm represents every valid sum in its l-canonical form. We claim that the l-canonical representation of any sum S is unique and can be computed in time $O(\log S)$.

LEMMA 2. *The* l*-canonical representation of any positive number* S *is unique.*

Proof. We give a proof by contradiction. Assume that $\mathbf{k} = (k_0, k_1, \ldots, k_j)$ and $\mathbf{k}' = (k'_0, k'_1, \ldots, k'_{j'})$ are two distinct l-canonical representations of S. Without loss of generality, assume that $j \leq j'$. Let d be the smallest index where the sequences differ. We have $d \leq j$ since it cannot happen that they agree on all of the indices less than or equal to j and the second sequence has nonzero components for indices greater than j, given that they have the same sum.

Case 1 ($d < j$). Since $l \leq k_d, k'_d \leq l + 1$, we have $|\sum_{i=0}^{d} k_i 2^i - \sum_{i=0}^{d} k'_i 2^i| = 2^d$. However, $|\sum_{i=d+1}^{j} k_i 2^i - \sum_{i=d+1}^{j'} k'_i 2^i| = c2^{d+1}$ for some integer $c \geq 0$, which is a contradiction since $|\sum_{i=0}^{j} k_i 2^i - \sum_{i=0}^{j'} k'_i 2^i| = 0$.

Case 2 ($d = j$). The sequence $\mathbf{k}'$ must have nonzero indices greater than j; otherwise, the two representations cannot give the same sum. Moreover, it cannot

happen that $k_j \leq k'_j$ since otherwise $\mathbf{k}'$ will have a strictly greater sum. Thus $k_j > k'_j$, and $k_j \leq l+1$. Since k'_j is not the last index, we have $k'_j \geq l$. Therefore, $|k'_j - k_j| \leq 1$, which implies $|\sum_{i=0}^{j} k_i 2^i - \sum_{i=0}^{j} k'_i 2^i| \leq 2^j$. However, $\sum_{i \geq j+1} 2^i \geq k'_i 2^{j+1}$, which gives a contradiction. □

The following procedure computes the l-canonical representation of S in time $O(\log S)$.

PROCEDURE l-CANONICAL. Given S, find the largest j such that $2^j \leq \frac{S}{l} + 1$, and let $S' = S - (2^j - 1)l$. If $S' \geq 2^j$, find m such that $m2^j \leq S' < (m+1)2^j$, and set $k_j = m$; we are guaranteed that $m < l$. Let $\widehat{S} = S' - m2^j < 2^j$. Let $b_0, \ldots, b_{j-1}$ be the binary representation of $\widehat{S}$. Set $k_i = l + b_i$ for $i < j$.

Given S and l, the l-canonical representation of S tells us the exact positions of all of the 1's where the buckets will start. Note that, since multiple 1's "belong" to the same data element, we may have multiple buckets starting at a single data element, implying that multiple buckets could have the same timestamp. The following observation is critical to the incremental maintenance of the buckets. The algorithm in section 2 guarantees that, if a certain data element (which in that case was some active 1) is not "indexed" at a certain time interval, then it will never be "indexed" in the future. By "indexed" we mean that it is the first element of some bucket, and hence its timestamp is maintained as the timestamp of that bucket. As time progresses, buckets may get merged, and some data elements may not be indexed any more. However, it never happens that an element that was not indexed at some time gets indexed later.

The preceding observation allows us to devise the following scheme to incrementally maintain the buckets with small amortized update time. Let us assume that we know the buckets at a certain time instant. We think of each data element as a series of 1's. We buffer B new elements separately and maintain the sum for these elements; that is, the EH is not updated for B steps. During this period, any query can be answered using a combination of the EH and the buffer sum. When the buffer gets full, we first delete any expired buckets in the EH. After the expired buckets are deleted, let S_1 be the sum of the sizes of the active buckets. Let S_2 be the sum of the elements in the buffer. We calculate the l-canonical ($l = \frac{k}{2}$) representation of $S_1 + S_2$ to determine the positions of the new buckets. This requires $O(\log(S_1 + S_2)) = O(\log N + \log R)$ time since $S_1 + S_2 = O(NR)$. We then create the new buckets using the timestamps and values of the elements in the buffer and the timestamps and sizes of the old buckets. The total time required to process the B elements in buffer is $O(B + \log N + \log R)$ since $O(B)$ time suffices to maintain the buffer sum and the number of buckets in the new histogram is $O(\log N + \log R)$. Since the time required to construct the new histogram is $O(\log N + \log R + B)$, the amortized update time per element is $O(1 + \frac{\log N + \log R}{B})$. Choosing $B = \Theta(\log N)$ makes the amortized update time $O(\frac{\log R}{\log N})$ and the worst-case time $O(\log N + \log R)$. The buffer needs $O(\log N(\log N + \log R))$ memory bits, which is the same as the memory requirement of the EH. Note that, if R is *poly*(N), then the amortized update time is $O(1)$ and the worst-case time is $O(\log N)$. We have obtained a memory upper bound of $O(\frac{1}{\epsilon}(\log N + \log R)(\log N))$ bits, as summarized in the following theorem.

THEOREM 5. *The generalized EH for the* SUM *problem maintains a data structure which provides estimates with relative error at most* ϵ *using at most* $(\frac{k}{2}+1)(\log(\frac{2NR}{k}+1)+1)$ *buckets, where* $k = \lceil \frac{1}{\epsilon} \rceil$. *The memory requirement is* $\log N + \log(\log N + \log R)$ *bits per bucket. The arrival of each new element can be processed in* $O(\frac{\log R}{\log N})$ *amortized*

time and $O(\log N + \log R)$ *worst-case time. At each time instant, the data structure provides a sum estimate in* $O(1)$ *time.*

We now prove a lower bound of $\Omega(\frac{1}{\epsilon}(\log N + \log R)(\log N))$ bits. If $\log N = \Omega(\log R)$, then the lower bound from section 3 applies. Thus we need only to consider the case when $R > N$. We will assume that $\log R \leq \frac{N}{k}$; in fact, we assume $\log R = o(N)$. Consider the following arrangements. We break the window of size N into $\log R$ blocks, each of size $\lfloor \frac{N}{\log R} \rfloor$. Consider the ith block for $0 \leq i < \log R$. We choose $k/4$ of the $\lfloor \frac{N}{\log R} \rfloor$ positions and place an element with value 2^i there, setting all other elements to 0. By an argument similar to the one in section 3, any deterministic algorithm with relative error less than $\frac{1}{k}$ must differentiate between any two of these arrangements. The total number of these arrangements is $\binom{N/\log R}{k/4}^{\log R} \geq (\frac{4N}{k \log R})^{\frac{k}{4} \log R}$. The number of memory bits required is at least $\frac{k}{4} \log R \log(\frac{4N}{k \log R}) = \Omega(\frac{1}{\epsilon}(\log N + \log R)(\log N))$. We assume that $R > N$ and that $\log R = O(N^\delta)$ for some $\delta < 1$. Note that the lower bounds also apply for randomized algorithms that provide an approximate answer.

5. Timestamps. In our model (given in section 2), we have assumed that data items arrive at regular time intervals and arrival time increases by one with every new data item that we have seen. However, in most real-life applications, this is not the case, and arrival rates of data items may be bursty. Moreover, we would like to define the sliding window based on real time. In other words, we may want to compute statistics based on the data items that arrived over the last hour, day, etc. It is easy to see that our algorithm can be easily adapted to do this by defining the arrival time based on *real time*; i.e., the arrival time increases by one with every clock-tick.[1] We define N (size of the sliding window) as the number of clock-ticks in the interval over which we want our sliding window to work. For example, N is 3600 if we want statistics based on the last hour, and clock-ticks occur every second. Note that the algorithm does no work except when it sees a data item, and hence it need not do anything during the clock-ticks for which no data items arrive. The invariants are automatically maintained during this period, and the algorithm never uses any extra space during this time. Hence it need not bother to delete the expired buckets until a new data item arrives since its memory requirement does not change. The memory requirement of the algorithm is $O(\frac{1}{\epsilon}(\log N)(\log N + \log R))$, where the second term $(\log N + \log R)$ is the logarithm of the maximum sum (NR) that can occur over N clock-ticks. Thus, if we are guaranteed that much less than N data items arrive over any sliding window, then the memory requirement would be less. This may happen for bursty arrival rates. In other words, our algorithm adapts its memory requirements with the amount of data that we observe.

5.1. Approximate timestamps. The EHs developed in section 2 and section 4 have a memory requirement of $O(\log N)$ for every bucket of the histogram. The timestamp that we maintain with each bucket requires $\log N$ bits and dominates the memory requirement of every bucket. We now explore the idea of maintaining a coarser timestamp with every bucket which requires only $\log \log N$ bits of memory and reduces the memory requirement for the EH from $O(\frac{1}{\epsilon}(\log^2 N))$ to $O(\frac{1}{\epsilon}(\log N)(\log \log N))$. In the case of generalized EH, the memory requirement drops to $O(\frac{1}{\epsilon}(\log N + \log R)(\log(\log N + \log R)))$. The effect of maintaining a coarser

[1] Equal length intervals, into which we partition time, that are assumed small enough so that no two data items are observed within a single interval.

timestamp is that, instead of answering the query over the last N data elements (sliding window size), we may answer the query over the last N/c elements, where $1 \leq c \leq 2$. In other words, we are approximating the window size to within a factor of 2. Note that this does not contradict the lower bound presented before. We no longer guarantee that the answer that we provide has relative error at most ϵ as compared to the correct answer over the last N elements. Instead, we guarantee that the answer will have error at most ϵ as compared to the correct answer over the last N/c data elements, where $1 \leq c \leq 2$. The factor 2 can be further improved to $1+\tau$ for $0 < \tau \leq 1$, and the memory requirement for the timestamp is $\log\log N + \log(\frac{1}{\tau})$. The generalization is obvious, and we explain the idea below for a factor 2 approximation.

We will explain the idea in terms of timestamps. However, as mentioned in section 2, we do not explicitly maintain the timestamp for every bucket and instead maintain the arrival time of the most recent (rightmost) element and calculate the timestamp using the arrival time of the current element. The idea translates to maintaining coarser arrival times. In sections 2 and 4, we maintained the exact timestamp with every bucket. Instead, here we maintain the timestamp to the closest power of 2. Thus, if the timestamp is t, where $2^{l-1} < t \leq 2^l$ $(1 \leq l \leq \lceil \log N \rceil)$, we maintain the timestamp as 2^l. In other words, we approximate the timestamp to the closest power of 2 greater than the timestamp. Since the timestamps now take $\log N$ distinct values, they can be stored using $\log\log N$ bits. The effect of this approximation is as follows: At any time instance, a bucket is active iff its timestamp is less than N. Any bucket whose exact timestamp is less than $N/2$ will still be considered active since the timestamp will be approximated to a value less than $N/2$. On the other hand, buckets whose timestamps are greater than $N/2$ may be wrongly considered inactive and hence deleted as their timestamp will be approximated to a value no less than N. Thus, in the worst case, we are answering the query over the last $N/2$ elements instead of N. If, instead, we approximate the timestamp to the closest power of 2 *less* than the exact timestamp, we get that we will be answering the query over the last cN elements, where $1 \leq c \leq 2$.

6. Computing L_p norms for vectors. We now extend the EH technique and combine it with the sketching technique from Indyk [13] to compute the L_p norms of vectors in the sliding window model. Assume that the window is broken into smaller contiguous buckets. These are numbered right to left and are denoted by $B_1, B_2, \ldots, B_m$. Consider a function f, defined over the intervals, with the following properties:

- P1. $f(B_i) \geq 0$.
- P2. $f(B_i) \leq poly(|B_i|)$.
- P3. $f(B_1 + B_2) \geq f(B_1) + f(B_2)$, where $B_1 + B_2$ denotes the concatenation of adjacent buckets B_1 and B_2.
- P4. $f(B_1 + B_2) \leq C_f(f(B_1) + f(B_2))$, where $C_f \geq 1$ is a constant.
- P5. The function $f(B)$ admits a "sketch" which requires $g_f(|B|)$ space and is composable; i.e., the sketch for $f(B_1 + B_2)$ can be composed efficiently from the sketches for $f(B_1)$ and $f(B_2)$.

If the function f admits these properties, then we can efficiently estimate it for sliding windows using the EH technique. We maintain buckets with the following two invariants; we also associate with every bucket a timestamp and the sketch. For now, we assume that the sketches provide the exact value of the function f. We will shortly relax this requirement and show that our technique works even if the sketches provide only an approximation to the actual function value.

INVARIANT 3. $f(B_{n+1}) \leq \frac{C_f}{k} \sum_{i=1}^{n} f(B_i)$.

INVARIANT 4. $f(B_{n+2}) + f(B_{n+1}) > \frac{1}{k} \sum_{i=1}^{n} f(B_i)$.

Observation 1. We estimate the function f for the current window by composing the sketches of all but the earliest (leftmost) bucket. The leftmost bucket may have certain expired data elements along with a suffix of data elements which are active. Let B_x be the part (suffix) of the leftmost bucket that is active and was ignored (i.e., did not contribute to the estimate). Let B_y be the concatenation of all of the other buckets whose sketch we compose using the sketches of the individual buckets. Then $B_x + B_y$ is the current window, and the exact answer is $f(B_x + B_y)$. However, we estimate the answer as $f(B_y)$; thus we always underestimate. The relative error $\boldsymbol{E_r}$ is $\frac{f(B_x+B_y)-f(B_y)}{f(B_x+B_y)} > 0$ by P1 and P3. Also, we have

$$\begin{aligned}
\boldsymbol{E_r} &\leq \frac{f(B_x + B_y) - f(B_y)}{f(B_y)} && \text{(P1, P3)} \\
&\leq \frac{C_f(f(B_x) + f(B_y)) - f(B_y)}{f(B_y)} && \text{(P4)} \\
&= \frac{C_f f(B_x)}{f(B_y)} + C_f - 1 \\
&\leq \frac{C_f f(B_{n+1})}{\sum_{i=1}^{n} f(B_i)} + C_f - 1 && \text{(P1, P3)} \\
&\leq \frac{C_f^2}{k} + C_f - 1 && \text{(Invariant 3).}
\end{aligned}$$

Observation 2. Invariant 4 and property P2 imply that the number of buckets will be $O(k \log N)$, where N is the size of the window. Thus the memory required to maintain the timestamp and the sketches for all of the buckets will be $O(k \log N(\log N + g_f(N)))$.

Hence, if we maintain the invariants along with the timestamp and the sketches, we can estimate the function f with relative error $0 \leq \boldsymbol{E_r} \leq \frac{C_f^2}{k} + C_f - 1$ using $O(k \log N(\log N + g_f(N)))$ memory bits. We can maintain the invariants along with the timestamp and the sketches as new data elements are added. The algorithm to do this is very similar to that for the EH.

1. When a new data element arrives, calculate the new expiry time. If the timestamp of the last bucket indicates expiry, delete that bucket.
2. Create a new bucket with just the new data element.
3. Traverse the list of buckets from right to left. If Invariant 4 is violated for a pair of buckets (B_{n+1}, B_{n+2}), merge them into a new bucket B'_{n+1}. The sketch for this bucket is composed from the sketches for B_{n+1} and B_{n+2}. We may need to do more than one merge.

We argue that the algorithm maintains Invariants 3 and 4. Adding a new bucket does not violate Invariant 3, as we increase only the size of the suffix. Whenever Invariant 4 is violated, the two buckets involved satisfy $f(B_{n+2}) + f(B_{n+1}) \leq \frac{1}{k} \sum_{i=1}^{n} f(B_i)$. When we merge them, property P4 guarantees that $f(B'_{n+1}) \leq C_f(f(B_{n+2}) + f(B_{n+1})) \leq \frac{C_f}{k} \sum_{i=1}^{n} f(B_i)$, and hence Invariant 3 is valid for the new bucket B'_{n+1}. The algorithm may need to do a lot of merges—as many as the number of buckets (i.e., $O(k \log N)$). However, the amortized time is $O(1)$. We omit details dealing with the fact that the function f for a window of size 1 may be greater than 1 although bounded by some constant R.

THEOREM 6. *A function f with properties* P1–P5 *can be estimated over sliding windows with relative error* $0 \leq \boldsymbol{E_r} \leq \frac{C_f^2}{k} + C_f - 1$ *using* $O(k \log N(\log N + g_f(N)))$ *bits of memory.*

So far we have assumed that the sketches compute the function value f exactly. Instead, in most sketching techniques, the sketches provide a $1+\hat{\epsilon}$ approximation $s(B)$ of the actual function value $f(B)$, i.e., $(1-\hat{\epsilon})f(B) \leq s(B) \leq (1+\hat{\epsilon})f(B)$. In that case, we maintain buckets with Invariants 3 and 4, replacing f with s. Invariant 4, property P2, and the fact that $s(B) \leq (1+\hat{\epsilon})f(B)$ guarantee that the number of buckets will be $O(k \log N)$ as before. We will now analyze the effect of approximation due to sketches on the error.

Maintaining the invariant $s(B_{n+1}) \leq \frac{C_f}{k} \sum_{i=1}^{n} s(B_i)$ guarantees that $f(B_{n+1}) \leq (1+\hat{\epsilon})^2 \frac{C_f}{k} \sum_{i=1}^{n} f(B_i)$. We will use the same technique (please refer to Observation 1) to estimate the function f using sketch estimates s instead of f. Thus, instead of providing the exact answer $f(B_x + B_y)$, we provide the estimate as $s(B_y)$. The relative error $\boldsymbol{E_r}$ is $\frac{f(B_x+B_y)-s(B_y)}{f(B_x+B_y)}$. We have

$$\begin{aligned}
\boldsymbol{E_r} &= \frac{f(B_x + B_y) - s(B_y)}{f(B_x + B_y)} \\
&\leq \frac{f(B_x + B_y) - f(B_y) + f(B_y) - s(B_y)}{f(B_y)} && \text{(P1, P3)} \\
&\leq \frac{f(B_x + B_y) - f(B_y)}{f(B_y)} + \frac{f(B_y) - s(B_y)}{f(B_y)} \\
&\leq \frac{C_f f(B_{n+1})}{\sum_{i=1}^{n} f(B_i)} + C_f - 1 + \hat{\epsilon} && \text{(proved earlier)} \\
&\leq (1+\hat{\epsilon})^2 \frac{C_f^2}{k} + C_f - 1 + \hat{\epsilon}.
\end{aligned}$$

This gives the following theorem.

THEOREM 7. *A function f with properties* P1–P5 *can be estimated over sliding windows with relative error* $0 \leq \boldsymbol{E_r} \leq (1+\hat{\epsilon})^2 \frac{C_f^2}{k} + C_f - 1 + \hat{\epsilon}$ *using* $O(k \log N(\log N + g_f(N)))$ *bits of memory, where $\hat{\epsilon}$ is the bound on relative error of the sketches.*

If $\hat{\epsilon}$ can be made arbitrarily close to 0, keeping the space requirement for the sketches (i.e., $g_f(|B|)$) small, we can get the same error as in the previous theorem by increasing k by a small constant factor.

6.1. L_p norms. We now argue that L_p norms (for $p \in [1,2]$) of vectors under a restricted model admit the properties P1–P5 and hence can be efficiently computed for sliding windows. Consider the restricted model [13] in which the jth data element is a pair (i_j, a_j), where $i_j \in [d] = \{0 \ldots d-1\}$ and $a_j \in \{0 \ldots M\}$ represents an increment to the i_jth dimension of an underlying vector. Every window B represents a vector, and its $L_p(B)$ norm is given by $L_p(B) = (\sum_{i \in [d]} |s_i|^p)^{1/p}$, where $s_i = \sum_{i_j = i, j \in B} a_j$ is the sum of unexpired increments to the ith dimension.

Note that the case in which $p = 1$ is the same as the SUM problem. If the dimension d of the underlying vector is small, one obvious way to maintain the L_p norm is to maintain the approximate sum for each dimension using the techniques in section 4. It would require $O(\frac{1}{\epsilon}(\log N + \log M)(\log N)d)$ bits of memory and give a relative error of ϵ. However, for high dimensional vectors, we propose the use of sketches. We denote $(L_p(B))^p$ by f_p and estimate f_p for $p \in [1,2]$. The function f_p clearly admits properties P1–P4, assuming $M \leq N^{O(1)}$. For P5, $f_p(B)$ admits a sketching technique

which requires $O(\log M \log(1/\delta)/\hat{\epsilon}^2)$ memory bits per sketch and is composable. The technique also requires $O(\log M \log(d/\delta) \log(1/\delta)/\hat{\epsilon}^2)$ random bits, which are common to all sketches. (See Theorem 2 in [13].) The sketches for computing the function are not exact; instead, they provide an approximation with relative error less than $\hat{\epsilon}$ with probability $1 - \delta$.

From Theorem 7, we have that, under the restricted model, we can compute f_p (i.e., $(L_p(B))^p$) with relative error at most $(1+\hat{\epsilon})^2 \frac{4}{k} + 1 + \hat{\epsilon}$ using $O(k \log N(\log N + \log M \log(1/\delta)/\epsilon^2))$ bits of memory. As mentioned before, an additional $O(\log M \log(d/\delta)$ $\log(1/\delta)/\hat{\epsilon}^2)$ bits of memory, which are common to all the sketches, are required. The estimate is approximately correct with high probability. Note that computing $(L_p(B))^p$ with a small relative error translates to computing $L_p(B)$ with a small relative error.

6.2. Lower bounds. The BasicCounting and Sum problems are the special cases of computing L_p norms, where the underlying vector has a single dimension. Thus the lower bounds for these problems apply to the problem of computing the L_p norm. Note that the upper bounds obtained in this section match the lower bounds asymptotically. The L_p norm for $p = 0$ is defined as the distinct value problem, and we deal with this problem in section 7.

7. Applications. We briefly discuss how the EH algorithm for BasicCounting can be used as a building block to adapt several techniques to the sliding window model with a multiplicative overhead of $O(\frac{1}{\epsilon} \log N)$ in memory and a $1+\epsilon$ factor loss in accuracy. The basic idea is that, to adapt to the sliding window setting a scheme relying on exact counters for positive integers, we will use an EH to play the role of a counter. A counter would have required $\Omega(\log N)$ bits of memory, while an EH requires $O(\frac{1}{\epsilon} \log^2 N)$ bits of memory and maintains the count with $1+\epsilon$ error.

7.1. Hash tables. This is the simplest case. Every data element gets hashed to a bucket, and the goal is to maintain the count of elements in each hash bucket. Instead of maintaining a counter for each bucket, we use the EH to maintain approximate counts of the number of data elements hashed into the bucket from the last N data elements in the stream.

7.2. Sums and averages. In section 4, we showed how to maintain the sum of positive integer data elements using the generalized version of the EHs. This requires $O(\frac{1}{\epsilon} \log N(\log R + \log N))$ bits of memory. Since maintaining the sum would require $\log N + \log R$ bits, the multiplicative overhead is $O(\frac{1}{\epsilon} \log N)$. Maintaining averages is similar. The average of the most recent N data elements is just the sum divided by N. If the items are inserted irregularly in real time and we want the average value to represent the sum divided by the number of insertions in the last N clock-ticks, an easy solution would be to use a second instance of the EH that maintains the count (number of insertions) approximately. Since both the sum and count have small relative error, so will their quotient.

7.3. Histograms. Given the bucket boundaries in a histogram, we can maintain the sum, average, and other statistics corresponding to each bucket using generalized EH. Finding the optimal bucket boundaries to optimize the memory requirement is an orthogonal problem. Also, equiwidth histograms are a natural choice of histograms for which the bucket boundaries are fixed. Note that, unlike the histograms discussed in [10], these are not time-based histograms but instead could be based on any attribute of the data.

7.4. Min and max. We prove a lower bound for the memory requirement of an algorithm that maintains min or max over a sliding window. While we argue the lower bound for the case of min, the argument for max is similar. The lower bound is based on a counting argument like the one used to prove the lower bound for BASICCOUNTING in Lemma 1. Let the data elements be drawn from a set of R distinct numbers. Consider all *nondecreasing* arrangements of N numbers. The number of such arrangements is $\binom{N+R-1}{N}$. Consider an algorithm that has seen one of these arrangements. We claim that any deterministic algorithm that gives the correct answer at every time instance henceforth must differentiate between any two such arrangements. To this end, we assume that we will present the algorithm with a sequence consisting of the highest number in the set, similarly to how we present the algorithm with a sequence of 0's in Lemma 1. Since the numbers presented to the algorithm were nondecreasing, at any time instance, the correct answer is the value of the oldest or least recent element which will expire in the next step. As a result, for every pair of arrangements, there will be a time when their oldest elements differ, and hence they have different correct answers. This proves our claim and establishes a lower bound on the number of memory bits required, which is $\log \binom{N+R-1}{N} \geq N \log(R/N)$. This lower bound is also valid for any randomized algorithm by arguments similar to the one in section 3. If $R = poly(N)$, then the lower bound says that we have to store all of the last N elements. The easiest way to maintain the exact minimum over sliding windows is to do the following: Keep the subsequence of data elements in which the leftmost item is the current minimum and the right neighbor or any element (in the subsequence) is the minimum of the elements to the right of the element in the stream. Such a subsequence can be maintained as a list of pairs (value, timestamp), where the list satisfies the property that both the value and the timestamp are strictly increasing. This scheme has a worst-case space requirement of $O(N \log R)$ bits. If the data elements arrive in a random order, then the list that we would maintain is analogous to the right spine of a "treap" where the timestamps are fully ordered and the values of the data elements are heap-ordered. In that case, the expected length of the list is $O(\log N)$, and the space complexity is given by $O(\log N \log R)$.

7.5. Distinct values. It is easy to adapt the technique of Flajolet and Martin [6] to estimate the number of distinct elements in the last N data elements. Their *probabilistic counting technique*[2] maintains a bitmap of size $O(\log R)$, where R is an upper bound on the number of distinct values in the data set. In the case of sliding windows, $R \leq N$, and a bitmap of size $O(\log N)$ suffices. We also maintain with each bit a timestamp of size $O(\log N)$. Whenever a bit is (re)set by a data element, we update the timestamp to that of the data element. This enables us to keep track of the bits that were set by the last N elements. Consequently, we can estimate the number of distinct elements with an expected relative accuracy of $O(\frac{1}{\sqrt{m}})$ using $O(m \log^2 N)$ bits of memory. Note that the lower bound for the BASICCOUNTING problem applies to the distinct value problem. Given an instance of the BASICCOUNTING problem, we can create an input where a 0 is mapped to 0 while every 1 is mapped to some distinct value (the *arrival time* of the element, for instance). Then the number of distinct values is one more than the number of 1's. This reduction shows that the lower bounds for the BASICCOUNTING problem apply to the distinct value problem.

Consider the problem of estimating the number of distinct values over sliding

[2]The technique assumes perfect hash functions. However, it suffices to use hash functions which do not have complete independence.

windows in the presence of deletions. We prove a space lower bound of $\Omega(N)$ bits, where N is the window size. Every data element consists of a value and a bit that indicates if the value is being "inserted" or "deleted." Consider the last N elements that form the current window. These define a collection of values and multiplicities. The multiplicity of a value is the number of times it is inserted in the current window minus the number of times it is deleted from the current window. It is possible for a value to have negative multiplicity since it may have been deleted more times than it was inserted. The number of distinct values is the number of values that have multiplicity greater than zero. Our lower bound holds even if we define the number of distinct values as those with nonzero multiplicity. Given this model we claim the following.

CLAIM 1. *An algorithm that estimates the number of distinct values within a factor* 2 *of the correct answer, over sliding windows and in the presence of deletions, requires* $\Omega(N)$ *bits of space, where* N *is the size of the sliding window.*

Proof. Consider an algorithm A that estimates the number of distinct values to within a factor of 2, over sliding windows and in the presence of deletions. Given any arbitrary bit vector $\boldsymbol{X} = \{x_1, x_2, \ldots, x_N\}$, we present the algorithm with the following input. Every bit x_i is mapped to a value y_i as follows: If x_i is 1, then y_i is set to i. Otherwise, y_i is set to 0. These values are input to the algorithm in the order y_1 to y_N along with the additional bit that represents that these values are being inserted. After the N values have been inserted, let S be the state of the algorithm. We claim that we can recover the last $N/2$ bits of the vector $\boldsymbol{X}$ (i.e., $\{x_{N/2+1}, x_{N/2+2}, \ldots, x_N\}$) using the state S of the algorithm. This proves that the information content of the state S is at least $N/2$ bits, and hence it would require $\Omega(N)$ bits of space. Given the state S, we recover the last bit (x_N) as follows: We insert $N-1$ elements with value 0. The current window now contains $N-1$ inserts of value 0 and an insert of value y_N, which, depending upon the value of x_N, is either 0 or N. The correct answer (i.e., number of distinct values) in that case is either 1 or 2, depending upon whether x_N is 0 or 1. Since the algorithm A estimates to within a factor of 2, it will distinguish between the two cases, and we can infer the last bit x_N. If the algorithm estimates the number of distinct values to be less than 2, then $x_N = 0$; otherwise, $x_N = 1$. Having inferred the last bit x_N, we can infer the previous bit x_{N-1} as follows: We "rewind" to the state S (state after inserting y_1 to y_N). In other words, we run the algorithm from state S again, by storing the state S. We input an element with value equal to y_N and bit set to delete, followed by $N-3$ elements with value equal to 0 and bit set to insert. In other words, we are deleting the value y_N that was inserted last. The current window now consists of $N-3$ inserts of value 0 and an insert of value y_{N-1}, which, depending on x_{N-1}, is either 0 or $N-1$. Note that the current window also contains a pair of elements that insert and delete the value y_N. By similar arguments as before, we can now infer the bit x_{N-1} since the algorithm A gives a factor 2 approximation. We can proceed this way to infer the last $N/2$ bits in the order $x_N, x_{N-1}, \ldots, x_{N/2+1}$. To infer the x_{N-i}th bit (having inferred $x_{N-i+1}, \ldots, x_N$), start with the state S again. Input elements with values $y_{N-i+1}, y_{N-i+2}, \ldots, y_N$ and bit set to delete, followed by $N-2i-1$ inserts of value 0. The current window will then contain $N-2i-1$ inserts of value 0, an insert of value y_{N-i}, and i pairs of inserts and deletions of values $y_{N-i+1}, y_{N-i+2}, \ldots, y_N$. Again, we can infer the bit x_{N-i}. We can do this for $i < N/2$.

The argument above proves that the state S essentially encodes the last $N/2$ bits and probably more, proving the lower bound for the space requirement of S. ∎

7.6. General sum. Consider the problem of maintaining the sum of the last N integers when the integers could be positive or negative. We prove that, even if we restrict the set of integers to $\{1, 0, -1\}$, to approximate the sum within a constant factor requires $\Omega(N)$ bits of memory. Moreover, it is easy to maintain the sum by storing the last N integers, which requires $O(N)$ bits of memory. We assume that the storage required for every integer is a constant independent of the window size N. This proves that the complexity of the problem in the general case (i.e., allowing positive and negative integers) is $\Theta(N)$. We now argue the lower bound of $\Omega(N)$. Consider an algorithm A that provides a constant factor approximation to the problem of maintaining the general sum. Given a bit vector of size $N/2$, we present the algorithm A with the pair $(-1, 1)$ for every 1 in the bit vector and the pair $(1, -1)$ for every 0. Consider the state (i.e., time instant) after we have presented all of the $N/2$ pairs to the algorithm. We claim that we can completely recover the original bit vector by presenting a sequence of 0's henceforth and querying the algorithm on every odd time instant. If the current time instant is T (after having presented the $N/2$ pairs), then it is easy to see that the correct answer at time instant $T + 2i - 1$ $(1 \leq i \leq N/2)$ is 1 iff the ith bit was 1 and -1 iff the ith bit was 0. Since the algorithm A gives a constant factor approximation, its estimate would be positive if the correct answer is 1 and negative if the correct answer was -1. Since the state of the algorithm after feeding the $N/2$ pairs enables us to recover the bit vector exactly for any arbitrary bit vector, it must be using at least $N/2$ bits of memory. This proves the lower bound. We can state the following theorem.

THEOREM 8. *The space complexity of any algorithm that gives a constant factor approximation, at every instant, to the problem of maintaining the sum of last N integers, which appear as a stream of data elements and could be positive or negative, is equal to $\Theta(N)$.*

8. Conclusion. In conclusion, this paper takes the first step toward computing over data streams in the sliding window model. We consider the problem of maintaining statistics over sliding windows and provide upper and lower space bounds for various problems. It remains to consider problems like maintaining other statistics (for instance, variance), clustering (maintaining k-medians), etc. in the sliding window model.

Acknowledgments. The authors would like to thank Yossi Matias for suggesting the idea of maintaining coarser timestamps presented in section 5. They would also like to thank the anonymous referees for their suggestions in improving the presentation.

REFERENCES

[1] N. ALON, Y. MATIAS, AND M. SZEGEDY, *The space complexity of approximating the frequency moments*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 20–29.

[2] C. CORTES, K. FISHER, D. PREGIBON, AND A. ROGERS, *Hancock: A language for extracting signatures from data streams*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2000, pp. 9–17.

[3] S. CHAUDHURI, R. MOTWANI, AND V. R. NARASAYYA, *On random sampling over joins*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, New York, 1999, pp. 263–274.

[4] M. FANG, H. GARCIA-MOLINA, R. MOTWANI, N. SHIVAKUMAR, AND J.D. ULLMAN, *Computing iceberg queries efficiently*, in Proceedings of the 24th International Conference on Very Large Data Bases, New York, NY, 1998, pp. 299–310.

[5] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, *An approximate L1-difference algorithm for massive data streams*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 501–511.

[6] P. Flajolet and G. Martin, *Probabilistic counting*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1983, pp. 76–82.

[7] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi, *Architecture of a Passive Monitoring System for Backbone IP Networks*, Technical report TR00-ATL-101-801, Sprint Advanced Technology Laboratories, Burlingame, CA, 2000.

[8] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, *Surfing wavelets on streams: One-pass summaries for approximate aggregate queries*, in Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy, 2001, pp. 79–88.

[9] S. Guha and N. Koudas, *Approximating a data stream for querying and estimation: Algorithms and performance evaluation*, in Proceedings of the Eighteenth International Conference on Data Engineering, San Jose, CA, 2002, pp. 567–576.

[10] S. Guha and N. Koudas, *Data-streams and histograms*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 471–475.

[11] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 359–366.

[12] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on Data Streams*, Technical report TR 1998-011, Compaq Systems Research Center, Palo Alto, CA, 1998.

[13] P. Indyk, *Stable distributions, pseudorandom generators, embeddings and data stream computation*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 189–197.

[14] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel, *Optimal histograms with quality guarantees*, in Proceedings of the 24th International Conference on Very Large Data Bases, New York, NY, 1998, pp. 275–286.

[15] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[16] Cisco Systems, *Netflow Services and Applications*, White paper, Cisco Systems, San Jose, CA, 2000; also available online from http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1814–1851

CURVATURE-CONSTRAINED SHORTEST PATHS IN A CONVEX POLYGON*

PANKAJ K. AGARWAL†, THERESE BIEDL‡, SYLVAIN LAZARD§, STEVE ROBBINS¶, SUBHASH SURI‖, AND SUE WHITESIDES¶

Abstract. Let B be a point robot moving in the plane, whose path is constrained to have curvature at most 1, and let $\mathcal{P}$ be a convex polygon with n vertices. We study the collision-free, optimal path-planning problem for B moving between two *configurations* inside $\mathcal{P}$. (A configuration specifies both a location and a direction of travel.) We present an $O(n^2 \log n)$ time algorithm for determining whether a collision-free path exists for B between two given configurations. If such a path exists, the algorithm returns a shortest one. We provide a detailed classification of curvature-constrained shortest paths inside a convex polygon and prove several properties of them, which are interesting in their own right. For example, we prove that any such shortest path is comprised of at most eight segments, each of which is a circular arc of unit radius or a straight-line segment. Some of the properties are quite general and shed some light on curvature-constrained shortest paths amid obstacles.

Key words. nonholonomic motion planning, shortest paths, mobile robot, curvature constraint, bounded radius of curvature, Dubins path, computational geometry

AMS subject classification. 68U05

PII. S0097539700374550

1. Introduction.

The *path-planning* problem, a central problem in robotics, involves planning a collision-free path for a robot moving amid obstacles and has been widely studied (see, e.g., the book by Latombe [21] and the survey papers by Schwartz and Sharir [31] and Halperin, Kavraki, and Latombe [17]). In the simplest form, given a moving point robot B, a set of obstacles, and a pair of configurations I and F specifying locations for B, we wish to find a continuous, collision-free path for B from I to F. This formulation, however, does not take into account the so-called *nonholonomic constraints* (for instance, bounds on acceleration or curvature) imposed on a robot by its physical limitations (see [21] for a more detailed discussion). Although there has been considerable recent work reported in the robotics and nonlinear-control literature on nonholonomic motion-planning problems (see [3, 5, 19, 20, 22, 24, 32, 42, 43] and

*Received by the editors June 27, 2000; accepted for publication (in revised form) May 2, 2002; published electronically October 18, 2002.

http://www.siam.org/journals/sicomp/31-6/37455.html

†Center for Geometric Computing, Computer Science Department, Duke University, Box 90129, Durham, NC 27708–0129 (pankaj@cs.duke.edu, http://www.cs.duke.edu/∼pankaj/). This author's work was supported in part by National Science Foundation research grants EIA–9870734, EIA–997287, and CCR–9732787, by the Army Research Office MURI grant DAAH04–96–1–0013, by a Sloan fellowship, and by a grant from the U.S.–Israeli Binational Science Foundation.

‡Department of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada (biedl@uwaterloo.ca). This author's research began during a postdoctoral tenure at McGill University, Montreal, PQ, H3A 2T5, Canada.

§INRIA Lorraine - LORIA, 615 rue du jardin botanique, B.P. 101, 54602 Villers-les-Nancy Cedex, France (lazard@loria.fr, http://www.loria.fr/∼lazard/). This author's research began during a postdoctoral tenure at McGill University, Montreal, PQ, H3A 2T5, Canada.

¶School of Computer Science, McGill University, Montreal, PQ, H3A 2T5, Canada (stever@cs.mcgill.ca, sue@cs.mcgill.ca). The work of Steve Robbins was supported by an FCAR scholarship. The work of Sue Whitesides was supported by NSERC and FCAR research grants.

‖Department of Computer Science, University of California, Santa Barbara, CA 93106 (suri@cs.ucsb.edu, http://www.cs.ucsb.edu/∼suri/). This author's research was partially supported by NSF grant CCR-9501494.

references therein), relatively little theoretical work has been done in this important area from an algorithmic perspective.

In this paper, we study the path-planning problem for a point robot whose configurations are specified by giving both a location and a direction of travel. This means that any solution to the path-planning problem for given initial and final configurations I and F must respect the directions of travel specified by I and F as well as the locations they specify. Furthermore, we require the path of the robot to have curvature at most 1. This curvature constraint arises naturally when the point robot models a real-world robot with a minimum turning radius; see, for example, [21]. Recently, Reif and Wang [29] confirmed that the problem of deciding whether there exists a collision-free curvature-constrained path for B between two given configurations amid polygonal obstacles is NP-hard. As a first step toward understanding which environments admit an efficient solution to this NP-hard problem, we study curvature-constrained path planning inside convex polygons.

We establish several new properties of shortest paths inside convex polygons and use them to characterize shortest paths in such environments. Using these properties and some geometric data structures [10], we present an efficient algorithm that, given initial and desired final configurations I and F in a convex polygon $\mathcal{P}$, determines whether there exists a curvature-constrained path from I to F that lies inside $\mathcal{P}$ and if so, computes a shortest one.

1.1. Previous results. Dubins [15] was perhaps the first to study curvature-constrained shortest paths. He proved that, in the absence of obstacles, a curvature-constrained shortest path from any start configuration to any final configuration consists of at most three segments, each of which is either a straight-line segment or an arc of a circle of unit radius, assuming that the curvature of the path is upper bounded by 1. Using ideas from control theory, Boissonnat, Cérézo, and Leblond [5] and, at the same time, Sussmann and Tang [38] gave an alternative proof. Further characterizations of shortest paths in an obstacle-free environment have been made by Boissonnat and Bui in [4] and Bui, Souères, Boissonnat, and Laumond in [8].

In the presence of polygonal obstacles, Jacobs and Canny [18] proved that, if there exists a curvature-constrained path between two configurations, there exists in the closed free space a curvature-constrained shortest path between these two configurations. Fortune and Wilfong [16] gave a $2^{poly(n,m)}$ time algorithm that decides whether a path exists, without finding one, where n is the total number of vertices in the polygons defining the obstacles and m is the number of bits of precision with which all points are specified. Jacobs and Canny [18], Wang and Agarwal [41], and Sellen [33, 34] gave approximation algorithms for computing an *ε-robust* path. (Informally, a path is ε-robust if it may be deformed by ε in a certain way while remaining feasible.) For the restricted case of pairwise disjoint *moderate obstacles*, i.e., convex obstacles whose boundaries have curvature upper bounded by 1, Agarwal, Raghavan, and Tamaki [1] gave efficient approximation algorithms. Boissonnat and Lazard [6] gave an $O(n^2 \log n)$ time algorithm for computing an exact shortest path for the case when the edges of the pairwise disjoint moderate obstacles consist of n segments that are circular arcs of unit radius or line segments. Their algorithm can be used to compute an optimal curvature-constrained path inside a convex polygon in time $O(n^7)$. Recently, Ahn, Cheong, Matoušek, and Vigneron [2] characterized the region of all points that can be reached from a given configuration in a convex polygon. Wilfong [42, 43] studied a restricted problem in which the robot must stay on one of m line segments (thought of as "lanes") except to turn between lanes. For a scene with

n obstacle vertices, his algorithm preprocesses the scene in time $O(m^2(n^2 + \log m))$, following which queries are answered in time $O(m^2)$.

There has also been work on characterizing shortest curvature-constrained paths when B is allowed to make reversals, that is, to back up. Reeds and Shepp [28] were the first to compute the characterization of such shortest paths in an obstacle-free environment. Boissonnat, Cérézo, and Leblond [5] and Sussmann and Tang [38] presented an alternative proof using ideas from control theory. Souères and Laumond refined the characterization of shortest paths [36]. In the presence of obstacles, Desaulniers proved that a *shortest* path does not necessarily exist even when paths exist [13]. Finally, much work has been done on computing curvature-constrained paths in the presence of obstacles when reversals are allowed; see, e.g., [3, 19, 20, 21, 22, 23, 24, 25, 32, 39, 40].

Other more general dynamic constraints have been considered in [3, 11, 12, 14, 26]. In particular, Sussmann [37] extended the characterization of curvature-constrained shortest paths to the 3-dimensional case.

1.2. Our model and results. Let B be a point robot and $\mathcal{P}$ a closed convex polygon with n vertices. For simplicity, we assume that the edges of $\mathcal{P}$ are in the general position: no two edges are parallel and no unit-radius circle (in $\mathbb{R}^2$) is tangent to three edges of $\mathcal{P}$. We believe that our techniques can be extended to carry through without this general-position assumption, although the technical details would be daunting. A *configuration* X for B is a pair $(\textsc{loc}(X), \psi(X))$, where $\textsc{loc}(X)$ is a point in the plane representing the location of the robot and $\psi(X)$ is an angle between 0 and 2π representing its orientation. When the meaning is clear, we generally write X instead of $\textsc{loc}(X)$.

The image of a differentiable function $\Pi : [0, l] \to \mathbb{R}^2$ is called a *path*. We denote by Π both the function and the path it defines. We regard a path Π as oriented from $\Pi(0)$ to $\Pi(l)$. We assume a path Π is parameterized by its arc length, and we let $\|\Pi\|$ denote its length. We say that Π is a path from a configuration X to another configuration Y if $\Pi(0) = \textsc{loc}(X)$, $\Pi(l) = \textsc{loc}(Y)$, and the oriented angles (with respect to the positive x-axis) of $\Pi'(0)$ and $\Pi'(l)$ are $\psi(X)$ and $\psi(Y)$, respectively. A path is called *moderate* if its average curvature is at most 1 in every positive-length interval.[1] This implies that the curvature is defined almost everywhere and that it is at most 1 wherever it is defined. Indeed, the derivative of a moderate path Π satisfies a Lipschitz condition, and therefore the derivative is differentiable almost everywhere [30, Theorem 8.19].

Any curve that lies entirely within the closed polygon $\mathcal{P}$ is called *free*. A path is *feasible* if it is moderate and free. A path Π from a configuration X to another configuration Y is *optimal* if it is feasible and its length is minimum among all feasible paths from X to Y. (If a feasible path from X to Y exists, then an optimal such path also exists [18].)

Throughout the paper, we say that we *compute* an object as a short way of saying that we compute such an object if one exists and return a flag of nonexistence otherwise.

Main results. Let $\mathcal{P}$ be an n-vertex convex polygon in the plane, and let I and F be two configurations inside $\mathcal{P}$.

(i) We give a classification of optimal paths from I to F.

(ii) We prove that an optimal path from I to F consists of at most eight maximal segments, each of which is either a line segment or a circular arc of unit radius.

[1] The *average curvature* of a path Π in the interval $[s_1, s_2]$ is defined by $\|\Pi'(s_1) - \Pi'(s_2)\|/|s_1 - s_2|$.

(iii) We give an $O(n^2 \log n)$ time algorithm to determine whether a feasible path from I to F exists. If such a path exists, then the algorithm returns an optimal path from I to F. If there are only k edges of $\mathcal{P}$ at a distance of at most 6 from both I and F, then the running time of our algorithm becomes $O((n + k^2) \log n)$.

Note that result (ii) above is actually quite surprising. Indeed, it means that the complexity of optimal paths inside a convex polygon is constant and does not depend on the number of edges of the polygon.

Note also that our algorithm for computing optimal paths is significantly faster than the algorithm implicit in the work of Boissonnat and Lazard [6] on computing an optimal path amid overlapping moderate obstacles, whose running time would be $O(n^7)$.

Some of the properties of moderate paths we prove are interesting in their own right. For example, one of these properties identifies a type of narrow region, called a "pocket," from which a moderate path cannot escape once it enters from outside. The conclusion will highlight this and another of these properties, which require technical definitions from later sections to describe in detail.

Our paper is organized as follows. In section 2, we present basic definitions, notation, and useful known results. In section 3, we give a classification of the optimal paths. In sections 4 and 5, we describe our algorithm. We conclude in section 6 with a discussion and some open problems.

2. Geometric preliminaries. Given a configuration X, let L_X denote the oriented line passing through $\text{LOC}(X)$ with orientation $\psi(X)$. A configuration X *belongs* to an oriented path (or curve) Π if $\text{LOC}(X) \in \Pi$ and L_X is the oriented tangent line to Π at $\text{LOC}(X)$. Note that a configuration X belongs to two oriented unit-radius circles. We will use $\mathcal{C}_X^+$ and $\mathcal{C}_X^-$ to denote the two circles of unit radius, oriented counterclockwise and clockwise, respectively, to which the configuration X belongs.

If X and Y are two points on a simple closed curve γ, then $\gamma^+[X, Y]$ (respectively, $\gamma^-[X, Y]$) denotes the portion of γ from X to Y in the counterclockwise (respectively, clockwise) direction, including X and Y; we will use $\gamma^+(X, Y), \gamma^-(X, Y)$ to denote portions excluding X, Y. Similarly, for a path Π and two configurations $X, Y \in \Pi$, we will use $\Pi[X, Y]$ to denote the portion of Π from X to Y.

Segments and Dubins paths. Let Π be a feasible path. We call a nonempty subpath of Π a *C-segment* (respectively, *S-segment*) if it is a circular arc of unit radius (respectively, line segment) and maximal; i.e., it is not strictly contained in a longer circular arc (respectively, line segment) of the path. A *segment* is either a C-segment or an S-segment. A C-segment on a path Π is called a C^+-segment (respectively, C^--segment) if Π induces a counterclockwise (respectively, clockwise) orientation on it. Suppose Π consists of a C-segment, an S-segment, and a C-segment; then we will say that Π is of type CSC or C_1SC_2 or $C_1S_2C_3$ if, for ease of reference, we want to number segments in order of their appearance in the sequence; superscripts $+$ and $-$ will be used to specify the orientations of C-segments of Π. Abusing the notation slightly, we often use the same symbol to denote both the type of a segment and the segment itself. Thus we may denote the first C-segment occurring in some path of type C_1SC_2 by C_1. The above notation can be generalized to an arbitrarily long sequence. Recall that, throughout the paper, C-segments and S-segments have positive length. Dubins [15] proved the following result.

LEMMA 2.1 (Dubins [15]). *In an obstacle-free environment, an optimal path between any two configurations is of type CCC or CSC or is a substring thereof.*

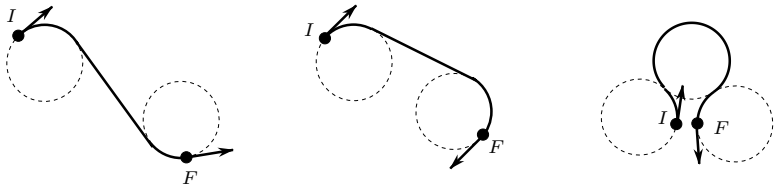


FIG. 2.1. *Different types of Dubins paths.*

We will refer to paths of type CCC or CSC or substrings thereof as *Dubins paths* (see Figure 2.1). In the presence of obstacles, Jacobs and Canny [18] observed that any subpath of an optimal path that does not touch any obstacle except at the endpoints is a Dubins path. In particular, they proved the following.

LEMMA 2.2 (Jacobs and Canny [18]). *Let Ω be a closed polygonal environment, I an initial configuration, and F a final configuration. Then an optimal path from I to F in Ω consists of a sequence $\Pi_1 \cdots \Pi_k$ of feasible paths, where each Π_i is a Dubins path from a configuration X_{i-1} to a configuration X_i, such that $X_0 = I$, $X_k = F$, and, for $0 < i < k$, $\text{LOC}(X_i) \in \partial\Omega$.*

The above lemma implies that an optimal path in a closed polygonal environment consists of C- and S-segments. In the following, we will consider only those paths that are formed by C- and S-segments. We will refer to circles and circular arcs of unit radius simply as circles and circular arcs. Notationally, we distinguish between a C-segment and its supporting circle, that is, the circle on which the circular arc lies, by using the calligraphy font $\mathcal{C}$ for the latter. While the symbols C and $\mathcal{C}$ are similar in appearance, it will be clear from the context whether we are referring to a supporting circle (denoted $\mathcal{C}$) or to a circular arc (denoted C).

Terminal and nonterminal segments. A segment of a feasible path Π is called *terminal* if it is the first or the last segment of Π; otherwise, it is called *nonterminal*. We apply the adjectives terminal and nonterminal to subpaths as well. If the first or last segment in Π is a C-segment, we will refer to it as a C_I-segment or a C_F-segment, respectively. Circles $\mathcal{C}_I^+$, $\mathcal{C}_I^-$, $\mathcal{C}_F^+$, and $\mathcal{C}_F^-$ are called *terminal circles* (see Figure 2.3).

The following lemma states some basic known properties of optimal paths; see [1, 15, 18].

LEMMA 2.3. *In an optimal path inside a convex polygon $\mathcal{P}$,*

(i) *any nonterminal C-segment has length greater than π,*
(ii) *any nonterminal C-segment is tangent to $\partial\mathcal{P}$ or to a terminal circle in at least one point, and*
(iii) *no nonterminal subpath has type CCC.*

Next we prove a property of a CS-subpath in an optimal path, which will be useful for our analysis.

LEMMA 2.4. *Let Π be an optimal path that contains a subpath of type CS, where the C-segment is not terminal. Then the C-segment is tangent to $\partial\mathcal{P}$, and the length of C between its first point and its last point of tangency with $\partial\mathcal{P}$ is greater than π.*

Proof. Let Π be an optimal path containing a subpath Π' of type $C_1C_2S_3$ or $S_1C_2S_3$. See Figure 2.2. Let X be the common endpoint of the first and second segments of Π', let Y be the last point of tangency of C_2 with $\partial\mathcal{P}$ along Π, and let X' be the point antipodal to X on C_2. ($X' \in C_2$ because the length of C_2 is greater than π by Lemma 2.3(i).)

Dubins [15] proved that the perturbations shown in Figure 2.2 (transforming paths $C_1C_2S_3$ and $S_1C_2S_3$ into paths of type $CCCS$ and $SCCS$ by reducing the length of

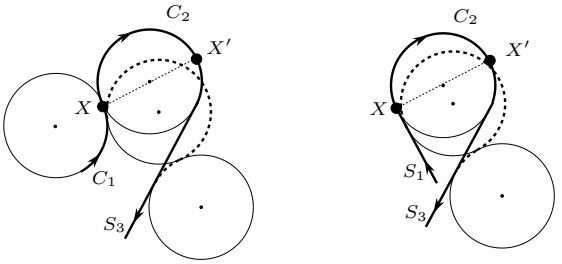


FIG. 2.2. *Length-reducing perturbations.*

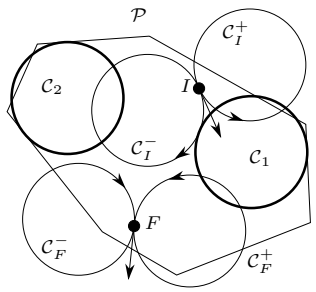


FIG. 2.3. *$\mathcal{PC}$-anchored ($\mathcal{C}_1$) and $\mathcal{PP}$-anchored ($\mathcal{C}_2$) circles.*

the first segment by any arbitrarily small value) shorten the paths in an obstacle-free environment. These shortenings can be done unless an obstacle obstructs the shortcut, i.e., unless $\partial\mathcal{P}$ is tangent to C_2 after X' (along Π). Thus, if $\|\Pi[X,Y]\| \leqslant \pi$, then Π can be shortened in $\mathcal{P}$, which contradicts the optimality of Π. □

Anchored segments. We define here the notion of anchored segments in a way that is similar to the notion introduced in [1]. A C-segment or circle is called *anchored* if it has at least two points of tangency with $\partial\mathcal{P}$ and the terminal circles. The terminal circles are not considered anchored. An anchored C-segment is denoted by $\bar{\bar{C}}$. By our general-position assumption on $\mathcal{P}$, there are a finite number of anchored circles. A C-segment with at least one point of tangency with $\partial\mathcal{P}$ is denoted by $\bar{C}$.

An anchored C-segment or circle is *$\mathcal{PP}$-anchored* if it is tangent to $\partial\mathcal{P}$ at two points and *$\mathcal{PC}$-anchored* if it is tangent to $\partial\mathcal{P}$ at one point and tangent to a terminal circle at another point; see Figure 2.3.

A circular arc is called *long* if its length is greater than π; otherwise, it is called *short*. A $\mathcal{PP}$-anchored C-segment is called *strongly $\mathcal{PP}$-anchored* if it contains the long arc defined by the points of tangency of its supporting circle with $\partial\mathcal{P}$ (see Figure 2.4b). Similarly, a $\mathcal{PC}$-anchored C-segment is called *strongly $\mathcal{PC}$-anchored* if it contains the long arc between a point of tangency of its supporting circle $\mathcal{C}$ with $\partial\mathcal{P}$ and a point of tangency of $\mathcal{C}$ with a terminal circle (see Figure 3.1a).

Closed moderate paths. We state here the following results about closed moderate paths.

PROPOSITION 2.5. *The region bounded by a simple moderate path Π whose initial and final locations coincide (the initial and final orientations may differ) contains at least one disk of unit radius. Moreover, if the initial and final orientations also coincide and if Π is not a circle of unit radius, then the region bounded by Π contains at least two distinct (but possibly overlapping) disks of unit radius.*

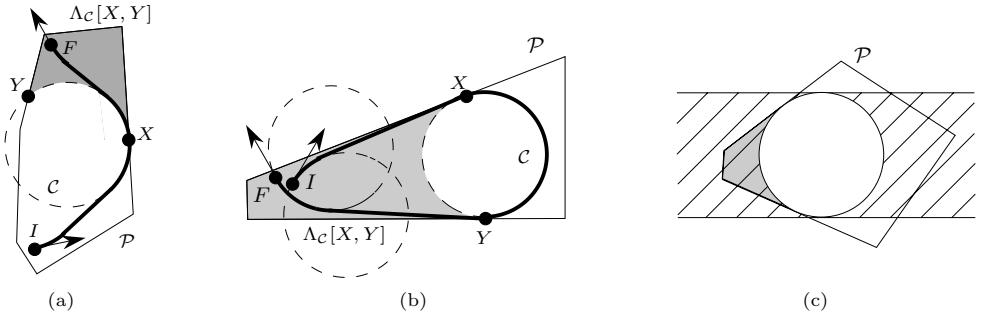


FIG. 2.4. *Pockets.*

The second statement in the proposition above was proved by Pestov and Ionin [27] for the restricted class of closed C^2 continuous curves (i.e., twice differentiable curves with continuous second derivative) whose curvatures are bounded everywhere. At the time we submitted this paper, Ahn et al. [2] presented a simple generalization of Pestov and Ionin's result, which yields the first statement of Proposition 2.5; furthermore, it appears straightforward to generalize their proof to obtain the second statement. Rather than carrying out that generalization here, we include in the appendix the proof we gave in the original version of this paper.

Pockets. We introduce here the notion of pockets. Let $\mathcal{C}$ be a circle intersecting $\partial\mathcal{P}$ at two or more points, and let X, Y be two consecutive[2] intersection points of $\partial\mathcal{P}$ with $\mathcal{C}$ so that the short arc of $\mathcal{C}$ joining X and Y lies inside $\mathcal{P}$. If $\mathcal{C}^+[X,Y]$ is the short arc and the turning angle[3] of $\partial\mathcal{P}^+(X,Y)$ is less than π, then the closed region bounded by $\partial\mathcal{P}^+[X,Y]$ and $\mathcal{C}^+[X,Y]$ is called a *pocket* (see Figure 2.4) and is denoted by $\Lambda_{\mathcal{C}}[X,Y]$. Similarly, we define the pocket $\Lambda_{\mathcal{C}}[X,Y]$ for the case in which $\mathcal{C}^-[X,Y]$ is the shorter arc. We will mostly be interested in pockets for which $\mathcal{C}$ is tangent to $\partial\mathcal{P}$ at X.

We first prove the following simple property of pockets.

LEMMA 2.6. *A pocket cannot contain a unit circle.*

Proof. Any pocket is contained in an open strip of width 2 (see Figure 2.4c) which does not contain any circle of unit radius. Thus there is no room for a unit circle in a pocket. □

We prove the following lemma, which will be crucial for characterizing the optimal paths containing a strongly anchored C-segment.

LEMMA 2.7. *If a feasible path enters the interior of a pocket, then it cannot escape the pocket.*

Proof. For a contradiction, let Π be a feasible path that enters the interior of a pocket $\Lambda_{\mathcal{C}}$ at X and escapes it at Y. See Figure 2.5. Let $\mathcal{C}$ denote the circle defining the pocket $\Lambda_{\mathcal{C}}$, and let $\mathcal{D}$ be the closed disk whose boundary is $\mathcal{C}$. If $\mathcal{C}$ intersects $\mathcal{C}_X^+$ at exactly two points, let them be denoted by X and X^+. If $\mathcal{C} = \mathcal{C}_X^+$, let X^+ denote the point on $\mathcal{C}$ antipodal to X. If the intersection of the two circles consists of exactly one point, denote this point by $X = X^+$. We define X^-, Y^+, and Y^- similarly. Let O_X^+, O_X^-, and O be the centers of the circles $\mathcal{C}_X^+, \mathcal{C}_X^-$, and $\mathcal{C}$, respectively.

We first prove that the segments X^+X^- and Y^+Y^- are diameters of $\mathcal{C}$. See

[2]Since $\mathcal{C}$ and $\partial\mathcal{P}$ are both convex, two consecutive intersection points of $\partial\mathcal{P}$ and $\mathcal{C}$ are consecutive on both $\partial\mathcal{P}$ and $\mathcal{C}$.

[3]The *turning angle* of a convex polygonal chain is $\sum_i(\pi - \theta_i)$, where θ_i is the interior angle at vertex i.

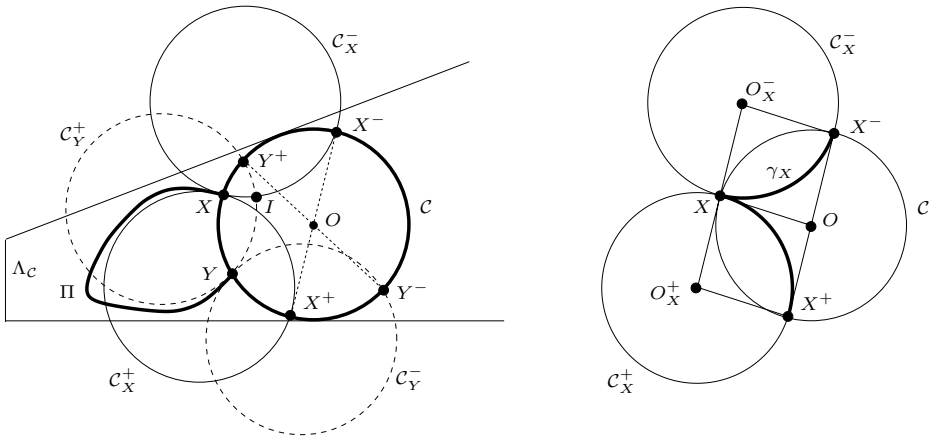


FIG. 2.5. *Illustration of the proof of Lemma* 2.7.

Figure 2.5. The quadrilaterals $O_X^+XOX^+$ and $O_X^-XOX^-$ are parallelograms. (These parallelograms flatten to line segments when $\mathcal{C}$ is equal to $\mathcal{C}_X^+$ or $\mathcal{C}_X^-$, but then X^+X^- is a diameter of $\mathcal{C}$ by definition.) Since the two parallelograms share the edge XO and the edges O_X^+X and XO_X^- are collinear, the two parallelograms are congruent. Therefore, O is the middle point of the segment X^+X^-. Similarly, Y^+Y^- is also a diameter of $\mathcal{C}$.

Let γ_X be the union of the arcs of $\mathcal{C}_X^+$ and $\mathcal{C}_X^-$ in $\mathcal{D}$, i.e., $\gamma_X = (\mathcal{C}_X^+ \cup \mathcal{C}_X^-) \cap \mathcal{D}$, and let γ_Y be defined similarly. The set γ_X either is equal to $\mathcal{C}$ or consists of two unit-radius circular arcs XX^- and XX^+; the same is true for γ_Y. The set γ_X belongs to $\mathcal{D}$, and segments X^+X^- and Y^+Y^- are diameters of $\mathcal{C}$; thus γ_X separates (not necessarily strictly) Y^+ and Y^- in $\mathcal{D}$. Since γ_Y also belongs to $\mathcal{D}$, γ_X and γ_Y intersect by the Jordan curve theorem.

First, note that the region $\Lambda_\mathcal{C} \cup \mathcal{D}$ cannot contain any unit-radius disk except $\mathcal{D}$; this is because O is the only point in $\Lambda_\mathcal{C} \cup D$ that is a distance of at least 1 from the boundary of $\Lambda_\mathcal{C} \cup D$.

Second, note that Π must be simple. Otherwise, Π would contain a simple, moderate subpath Π', lying in $\Lambda_\mathcal{C}$, with equal initial and final locations. This contradicts Proposition 2.5 since, by Lemma 2.6, a pocket cannot contain a unit disk.

Third, locations X and Y cannot be equal. Otherwise, Π would be simple, moderate, and lying in $\Lambda_\mathcal{C}$, with equal initial and final locations. As before, this contradicts Proposition 2.5.

Now we describe how to obtain a path Π' that leads to a contradiction. Consider first the case in which γ_X and γ_Y each consist of two unit-radius circular arcs XX^+, XX^- and YY^+, YY^-, respectively. Let I be the intersection point between γ_X and γ_Y that is the closest to X on γ_X (see Figure 2.5). This ensures that the circular arcs $IX \subset \gamma_X$ and $YI \subset \gamma_Y$ intersect only at I. Let Π' be the concatenation of the circular arc $IX \subset \gamma_X$, the path Π, and the circular arc $YI \subset \gamma_Y$. Path Π' is simple because Π is simple and the arcs IX and IY intersect only at I. Thus Π' is simple, moderate, and contained in the region $\Lambda_\mathcal{C} \cup \mathcal{D}$, which does not contain any unit-radius disk except $\mathcal{D}$. Thus, by Proposition 2.5, Π' encloses $\mathcal{D}$. It follows that the circular arcs IX and IY defining Π' cannot intersect the interior of $\mathcal{D}$. Thus the arcs IX and IY are reduced to a point $I = X = Y$, which is a contradiction.

Consider now the case in which exactly one of γ_X, γ_Y is equal to $\mathcal{C}$. Assume without loss of generality that $\gamma_X = \mathcal{C}$ and γ_Y consists of two unit-radius circular

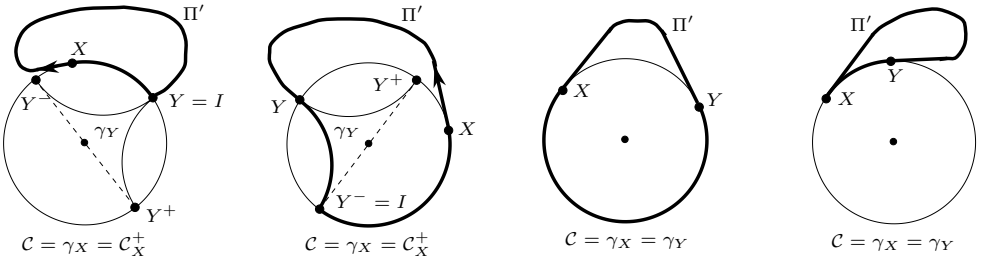


FIG. 2.6. *Illustration of the proof of Lemma* 2.7.

arcs YY^+ and YY^-. See, for example, Figure 2.6a and b. Since $\gamma_X = \mathcal{C}$, circle $\mathcal{C}$ is equal to $\mathcal{C}_X^+$ or $\mathcal{C}_X^-$; assume without loss of generality that $\mathcal{C} = \mathcal{C}_X^+$. Let I be the point in $\gamma_X \cap \gamma_Y = \{Y^-, Y, Y^+\}$ such that $\mathcal{C}^+[I, X]$ has minimum length. It follows that the arc $\mathcal{C}^+[I, X]$ intersects the arc $YI \subset \gamma_Y$ only at I. As before, we define Π' as the concatenation of $\mathcal{C}^+[I, X]$, Π, and YI, and, again, Π' is simple and moderate and must enclose $\mathcal{D}$. It follows that the circular arc YI defining Π' cannot intersect the interior of $\mathcal{D}$, and thus $I = Y$. Thus path Π' is the concatenation of $\mathcal{C}^+[I, X]$ and Π. Path Π lies in $\Lambda_{\mathcal{C}}$, and $\Lambda_{\mathcal{C}} \cap \mathcal{D}$ is an arc of $\mathcal{C}$ of length smaller than π. Thus, since Π' must enclose $\mathcal{D}$, the arc $\mathcal{C}^+[I, X]$ is the long arc of $\mathcal{C}$ joining I and X. Since Y^+Y^- is a diameter of $\mathcal{C}$, the arc $\mathcal{C}^+[I, X]$ strictly contains Y^+ or Y^-, contradicting the definition of I.

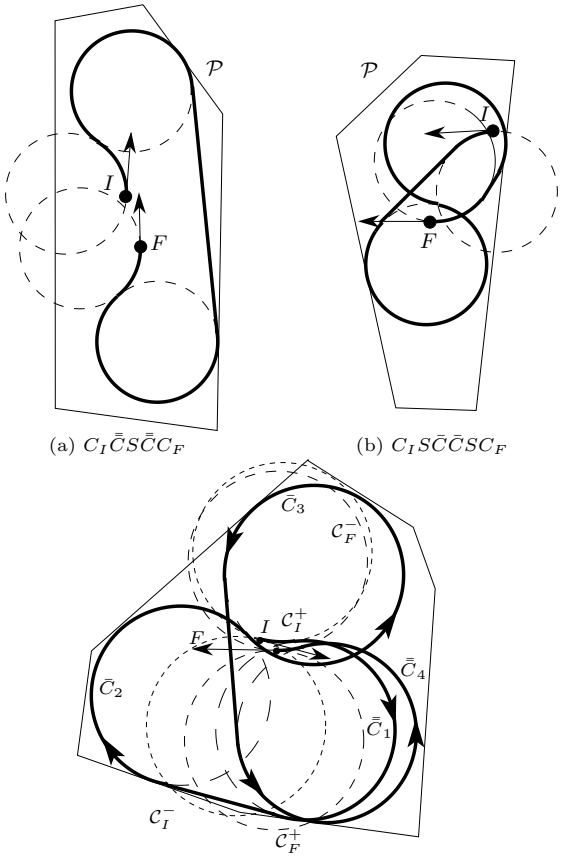
The remaining case is when both γ_X and γ_Y are equal to $\mathcal{C}$. Let Π' be the concatenation of Π and the arc XY of $\mathcal{C}$ joining X to Y such that, if possible, Π' is moderate everywhere (see, for example, Figure 2.6c), or if not possible, XY is the short arc (see Figure 2.6d). In the first case, Π' encloses at least two unit-radius disks by Proposition 2.5, contradicting the fact that $\Lambda_{\mathcal{C}} \cup \mathcal{D}$ contains Π' and only one unit-radius disk $\mathcal{D}$. In the second case, Π' encloses one unit-radius disk by Proposition 2.5 and lies in $\Lambda_{\mathcal{C}}$ because the short arc of $\mathcal{C}$ joining X and Y lies in $\Lambda_{\mathcal{C}}$. Thus $\Lambda_{\mathcal{C}}$ contains a unit disk, contradicting Lemma 2.6. □

3. Classification of optimal paths. The goal of this section is to prove the first of our main results, namely, a detailed characterization of optimal paths in convex polygons. We show that any optimal path is of type $C_I CSCCSCC_F$ or a subsequence of this form. However, not every subsequence of the above sequence can form an optimal path. The following theorem gives a more refined description of optimal path types. Recall that a segment has nonzero length by definition. In the following, we use $\cdot$ to denote a subpath of zero length.

THEOREM 3.1. *An optimal path* Π *inside* $\mathcal{P}$ *either is a Dubins path or is one of the types listed below. Except in case* (B.i), *all of the* C*-segments labeled* $\bar{\bar{C}}$ *are strongly anchored.*

(A) *If* Π *has no nonterminal* CC *subpath, then* Π *is one of the following types:*
  (A.i) $\Pi_I S \bar{\bar{C}} S \Pi_F$, *where* $\Pi_I \in \{C_I, \cdot\}$ *and* $\Pi_F \in \{C_F, \cdot\}$ *(see Figure* 2.4b*),*
  (A.ii) $\Pi_I S \Pi_F$, *where* $\Pi_I \in \{C_I \bar{\bar{C}}, C_I, \cdot\}$ *and* $\Pi_F \in \{\bar{\bar{C}} C_F, C_F, \cdot\}$ *(see Figure* 3.1a*).*

(B) *If* Π *has a nonterminal* CC *subpath, then* Π *is one of the following types:*
  (B.i) $C_I C \bar{\bar{C}} C_F$ *or* $C_I \bar{\bar{C}} C C_F$,
  (B.ii) $\Pi_I S \bar{\bar{C}} C C_F$ *or* $C_I C \bar{\bar{C}} S \Pi_F$, *where* $\Pi_I \in \{C_I, \cdot\}$ *and* $\Pi_F \in \{C_F, \cdot\}$,
  (B.iii) $\Pi_I \bar{C} \bar{C} \Pi_F$, *where* $\Pi_I \in \{C_I \bar{\bar{C}} S, C_I S, C_I, S\}$ *and* $\Pi_F \in \{S \bar{\bar{C}} C_F, S C_F, C_F, S\}$ *(see Figures* 3.1b, c*).*



(a) $C_I\bar{C}S\bar{C}C_F$ (b) $C_IS\bar{C}\bar{C}SC_F$

(c) $C_I\bar{C}_1S\bar{C}_2\bar{C}_3S\bar{C}_4C_F$

FIG. 3.1. *Examples of shortest paths.*

Remark 3.2. We informally checked by a case analysis on figures that, for the polygon and configurations I and F shown in Figure 3.1c, no path of type (A) or (B) is feasible except paths of type $C_ICS\bar{C}\bar{C}SCC_F$. This leads us to believe that the type $C_ICS\bar{C}\bar{C}SCC_F$, having eight segments, does occur as an optimal path type.

The proof of Theorem 3.1 is based on the following lemmas.

LEMMA 3.3 (Agarwal, Raghavan, and Tamaki [1]). *An optimal path has at most one nonterminal CC subpath. Moreover, any nonterminal C-segment that precedes (respectively, follows) a C_1C_2 subpath is oriented the same way as C_1 (respectively, C_2).*

Next we state a lemma whose proof we postpone until section 3.1.

LEMMA 3.4. (i) *If an optimal path has a subpath of type SCS, then the C-segment in that subpath is strongly $\mathcal{PP}$-anchored.*

(ii) *If an optimal path has a subpath of type $C_1C_2C_3S$ (or $SC_3C_2C_1$) so that the C-segment C_2 does not touch $\partial\mathcal{P}$, then C_3 is strongly $\mathcal{PP}$-anchored (see Figure* 3.5*).*

We next characterize some optimal paths that contain a strongly anchored C-segment.

LEMMA 3.5. (i) *If an optimal path Π contains a strongly $\mathcal{PP}$-anchored C-segment $\bar{\bar{C}}$, then $\Pi = \Pi_I\bar{\bar{C}}\Pi_F$, where Π_I and Π_F lie in some pockets defined by $\bar{\bar{C}}$.*

(ii) *If an optimal path Π contains a strongly $\mathcal{PC}$-anchored C-segment $\bar{\bar{C}}$ whose supporting circle is not free, then Π is of type $C_I\bar{\bar{C}}\Pi_F$ or $\Pi_I\bar{\bar{C}}C_F$, where Π_I and Π_F lie in some pockets defined by $\bar{\bar{C}}$.*

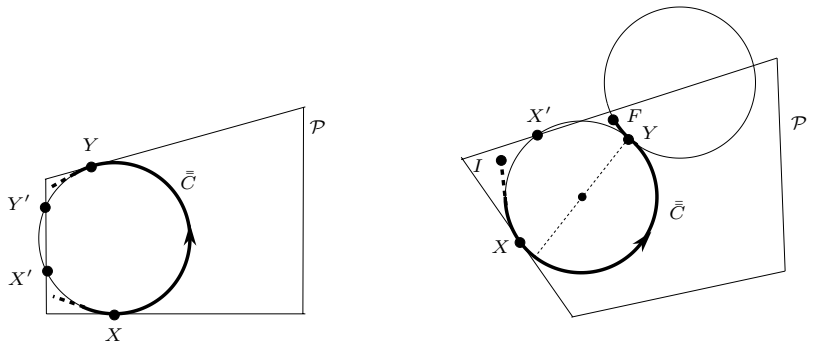


FIG. 3.2. *Illustration of the proof of Lemma* 3.5. *In* (a), *an optimal path containing a strongly $\mathcal{PP}$-anchored C-segment must start and end in a pocket.*

Proof. In case (i), the segment $\bar{\bar{C}}$ is strongly $\mathcal{PP}$-anchored, so it is tangent to $\partial\mathcal{P}$ at two points. Let X be the first point of tangency encountered along $\bar{\bar{C}}$, and let Y be the second point of tangency. See Figure 3.2a. By the general-position assumption, $\bar{\bar{C}}$ does not touch $\partial\mathcal{P}$ at any point other than X and Y. Without loss of generality, let $\bar{\bar{C}}$ be oriented counterclockwise. Let $\bar{\bar{\mathcal{C}}}$ be its supporting circle, and define X' as the first point of intersection between $\bar{\bar{\mathcal{C}}}$ and $\partial\mathcal{P}$ encountered moving clockwise along $\bar{\bar{\mathcal{C}}}$ from X. Notice that X' belongs to $\bar{\bar{\mathcal{C}}}^-(X,Y]$ since the intersection between $\bar{\bar{\mathcal{C}}}$ and $\partial\mathcal{P}$ contains Y.

Segment $\bar{\bar{C}}$ is strongly anchored, so $\|\Pi[X,Y]\| = \|\bar{\bar{\mathcal{C}}}^+[X,Y]\| > \pi$. Point X' is on the short arc $\bar{\bar{\mathcal{C}}}^+[Y,X]$, so $\|\bar{\bar{\mathcal{C}}}^+[X',X]\| < \pi$. The turning angle of $\partial\mathcal{P}^+[X,Y]$ is equal to $\psi(Y)-\psi(X) \pmod{2\pi}$, which, in turn, is equal to $\|\Pi[X,Y]\| > \pi$. The total turning angle around $\partial\mathcal{P}$ is 2π, so the turning angle of $\partial\mathcal{P}^+[X',X] < \pi$. Finally, $\bar{\bar{\mathcal{C}}}$ is tangent to $\partial\mathcal{P}$ at X, so $\bar{\bar{\mathcal{C}}}^+[X',X]$ lies inside the polygon. Thus arc $\bar{\bar{\mathcal{C}}}^+[X',X]$ forms the pocket $\Lambda_{\bar{\bar{\mathcal{C}}}}[X',X]$.

The pocket $\Lambda_{\bar{\bar{\mathcal{C}}}}[X',X]$ contains $\Pi[I,X]$ and thus Π_I, by Lemma 2.7. Indeed, otherwise, the path Π_I contains $\bar{\bar{\mathcal{C}}}^+[X',X]$, and $\partial\mathcal{P}$ is tangent to $\bar{\bar{\mathcal{C}}}$ at X'. This contradicts either the optimality of Π (if $X' = Y$) or the fact that X is the first point of tangency between $\bar{\bar{\mathcal{C}}}$ and $\partial\mathcal{P}$ encountered along Π (if $X' \neq Y$).

Similarly, Π_F is contained in a pocket $\Lambda_{\bar{\bar{\mathcal{C}}}}[Y,Y']$, where Y' is the first point of intersection between $\bar{\bar{\mathcal{C}}}$ and $\partial\mathcal{P}$ encountered moving counterclockwise along $\bar{\bar{\mathcal{C}}}$ from Y.

For case (ii), we assume that $\Pi = \Pi_I\bar{\bar{C}}C_F$; the case in which $\Pi = C_I\bar{\bar{C}}\Pi_F$ is symmetrical. Let X be the first point along $\bar{\bar{C}}$ that is tangent to $\partial\mathcal{P}$, and let Y be the last point of $\bar{\bar{C}}$. See Figure 3.2b. Then the proof is exactly the same as in (i), once noting that X' always exists on $\bar{\bar{\mathcal{C}}}^-(X,Y]$ because $\bar{\bar{\mathcal{C}}}$ is not free by assumption. ∎

LEMMA 3.6. *Let Π be an optimal path that contains a subpath of type $C_1S\bar{\bar{C}}$ or $\bar{\bar{C}}SC_1$, where $\bar{\bar{C}}$ is either a strongly $\mathcal{PP}$-anchored C-segment or a strongly $\mathcal{PC}$-anchored C-segment whose supporting circle is not free. Then C_1 is terminal.*

Proof. We consider the case in which Π contains a subpath of type $C_1S\bar{\bar{C}}$; the case of a subpath of type $\bar{\bar{C}}SC_1$ is symmetrical. Suppose that C_1 is not terminal. If $\bar{\bar{C}}$ is strongly $\mathcal{PP}$-anchored or is a strongly $\mathcal{PC}$-anchored C-segment whose supporting circle is not free, then $\bar{\bar{\mathcal{C}}}$ defines one or more pockets. By Lemma 3.5, C_1 belongs to one of the pockets defined by $\bar{\bar{\mathcal{C}}}$, say, $\Lambda_{\bar{\bar{\mathcal{C}}}}$.

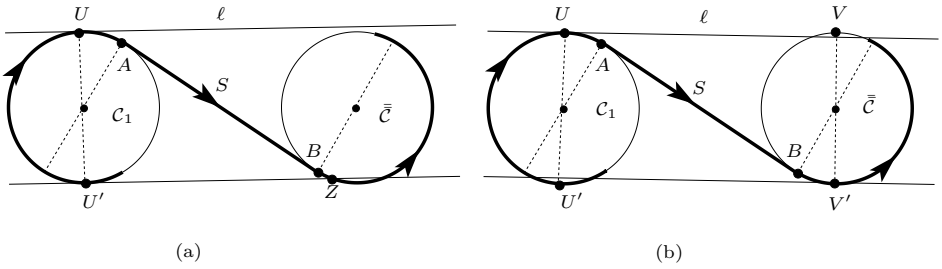


FIG. 3.3. *Illustration of the proof of Lemma* 3.6.

We claim that the circle $\mathcal{C}_1$ supporting the C-segment C_1 is not free. Indeed, otherwise, there would exist a feasible path that enters $\Lambda_{\bar{\bar{\mathcal{C}}}}$ on $\mathcal{C}_1$ (since no circle of unit radius is entirely contained in a pocket by Lemma 2.6) and escapes the pocket on $\bar{\bar{C}}$, contradicting Lemma 2.7.

Since C_1 is not terminal, its length is greater than π by Lemma 2.3(i). The length of $\bar{\bar{C}}$ is also greater than π because anchored circles are not terminal by definition.

Suppose first that the C-segments C_1 and $\bar{\bar{C}}$ have the same orientation along Π. Since the lengths of C_1 and $\bar{\bar{C}}$ are greater than π, the convex hull of C_1 and $\bar{\bar{C}}$ contains $\mathcal{C}_1$. By convexity of $\mathcal{P}$, the convex hull of C_1 and $\bar{\bar{C}}$ and thus $\mathcal{C}_1$ lies inside $\mathcal{P}$, which contradicts the above claim that $\mathcal{C}_1$ is not free.

Suppose now that the C-segments C_1 and $\bar{\bar{C}}$ have opposite orientation along Π (see Figure 3.3). Let A and B be the first and last points of the S-segment, respectively, in the subpath $C_1S\bar{\bar{C}}$. Let U be the last point of tangency along Π between C_1 and $\partial\mathcal{P}$, and let U' be the point antipodal to U on $\mathcal{C}_1$. By Lemma 2.4, U' belongs to C_1. Note that the arc on C_1 joining U' and A is longer than π. Let ℓ be the line tangent to $\mathcal{C}_1$ at U; ℓ contains an edge of $\mathcal{P}$.

If ℓ does not intersect $\bar{\bar{\mathcal{C}}}$, then the line tangent to C_1 at U' (which is parallel to ℓ) intersects the subpath $S\bar{\bar{C}}$ at a point Z (see Figure 3.3a). It follows that the convex hull of $\Pi[U', Z]$ contains $\mathcal{C}_1$. Thus $\mathcal{C}_1$ is free, again contradicting the claim that $\mathcal{C}_1$ is not free. On the other hand, if ℓ intersects $\bar{\bar{\mathcal{C}}}$, then let $V, V' \in \bar{\bar{\mathcal{C}}}$ such that $\overrightarrow{VV'} = \overrightarrow{UU'}$ (see Figure 3.3b). Points B and V lie on opposite sides of ℓ, and $\mathcal{P}$ lies in one of the half-planes bounded by ℓ. Since $B \in \Pi \subset \mathcal{P}$, we conclude that $V \notin \mathcal{P}$. Recall that $\|\bar{\bar{C}}\| \geqslant \pi$; therefore, $V' \in \bar{\bar{C}}$. The point on $\bar{\bar{C}}$ that is antipodal to the last point of $\bar{\bar{C}}$ lies on $\Pi[B, V']$. Thus, by Lemma 2.4 applied on the reversed path of Π, $\bar{\bar{C}}$ is tangent to $\partial\mathcal{P}$ at a point on $\Pi[B, V']$. Since the line tangent to $\bar{\bar{C}}$ at V' separates U' and A, any line tangent to $\Pi[B, V']$ separates U' and A. Thus $U' \notin \mathcal{P}$, which in turn implies that $U' \notin C_1$, thereby contradicting the claim that $U' \in C_1$. □

We now prove Theorem 3.1.

Proof of Theorem 3.1. The proof proceeds by considering how a nonterminal C-segment may appear in Π. If there is no nonterminal C-segment in Π, then Π is of type C_ISC_F or a substring thereof; i.e., Π is a Dubins path.

Assume now that there is a nonterminal C-segment in Π. Then such a segment belongs to a subpath of Π of type either SCS or CC. Suppose Π contains a subpath of type SCS. By Lemma 3.4, the C-segment in SCS must be strongly $\mathcal{PP}$-anchored. Thus, by Lemma 3.6, Π is of type $C_IS\bar{\bar{C}}SC_F$ or substrings (containing $S\bar{\bar{C}}S$) thereof. In other words, Π is of type (A.i).

If Π contains a nonterminal C-segment but not a subpath of type SCS, we know it must contain a subpath of type CC. There are two cases to consider, depending on whether the CC subpath is terminal.

Case 1. Π does not contain any nonterminal subpath of type CC. Thus one of the C-segments in any CC subpath must be a terminal segment. Either Π is of type $C_I C_F$ or $C_I C C_F$, or any nonterminal C-segment of Π is also adjacent to an S-segment. Π must then be of type $C_I C S C C_F$ or any substring thereof containing S and a terminal CC. By Lemma 2.4, the nonterminal C-segments are strongly $\mathcal{PC}$-anchored. All of these types of paths are thus either Dubins paths or paths of type (A.ii).

Case 2. Π contains a nonterminal subpath of type CC. By Lemma 3.3, it is the unique nonterminal CC-subpath in Π. Thus Π has the form $\Pi_I CC \Pi_F$, where Π_I, Π_F do not contain a nonterminal subpath of type CC. Thus any nonterminal C-segment in Π_I must be followed by an S-segment; otherwise, $\Pi_I CC$ will contain a nonterminal CCC-subpath, which contradicts the optimality of Π (Lemma 2.3(iii)). Furthermore, since we have no SCS subpath in Π, a nonterminal C-segment must be preceded by a terminal C-segment. This means that $\Pi_I = C_I CS$ or a subsequence of it. The subsequence cannot be empty, for otherwise the middle CC-subpath of Π would in fact be terminal; nor can it be simply CC, as noted above. Thus $\Pi_I \in \{C_I CS, C_I S, C_I, S\}$. Similarly, $\Pi_F \in \{SCC_F, SC_F, C_F, S\}$.

If $\Pi_I = C_I CS$ or $\Pi_F = SCC_F$, then the nonterminal C-segment in Π_I or Π_F is strongly anchored by Lemma 2.4.

If both Π_I and Π_F contain an S-segment, then the nonterminal CC-subpath in Π is preceded and followed by an S-segment. Thus both C-segments of the nonterminal CC-subpath in Π touch $\partial\mathcal{P}$. Indeed, otherwise Π contains a subpath of type SCC or CCS that does not touch $\partial\mathcal{P}$, which contradicts Lemma 2.2. Hence, if both Π_I and Π_F contain an S, then Π is of type (B.iii).

If neither Π_I nor Π_F contains an S-segment, then the path is of type $C_I CCC_F$. By Lemma 2.2, one of the nonterminal C-segments must touch $\partial\mathcal{P}$. This C-segment is also tangent to a terminal circle and is therefore $\mathcal{PC}$-anchored. Thus the path is of type (B.i). Note that, if both nonterminal C-segments touch $\partial\mathcal{P}$, then the path is of type $C_I \bar{\bar{C}} \bar{\bar{C}} C_F$, which can be considered as type (B.i) or (B.iii).

The last case to consider is when exactly one of Π_I or Π_F contains an S-segment. Say $\Pi_I = C_I$ and $\Pi_F \neq C_F$. The path has the form $C_I C_1 C_2 \Pi_F$, where Π_F starts with an S-segment. We know that C_2 must touch $\partial\mathcal{P}$ by Lemma 2.3(ii). If C_1 also touches $\partial\mathcal{P}$, then the path Π is of type (B.iii). Otherwise, if C_1 does not touch $\partial\mathcal{P}$, then, by Lemma 3.4(ii), C_2 must be strongly $\mathcal{PP}$-anchored. Lemma 3.6 then restricts the path Π to be of type (B.ii). Similarly, if $\Pi_I \neq C_I$ and $\Pi_F = C_F$, the path Π is of type (B.ii) or (B.iii). □

3.1. Proof of Lemma 3.4.

3.1.1. Proof of Lemma 3.4(i). Here we prove Lemma 3.4(i), which states

If an optimal path has a subpath of type SCS, then the C-segment in that subpath is strongly $\mathcal{PP}$-anchored.

Consider an optimal path $\Pi = S_1 C_3 S_5$ from configuration I to configuration F. We show that the path Π can be shortened unless the C-segment C_3 is strongly $\mathcal{PP}$-anchored. Assume without loss of generality that the C_3 is oriented counterclockwise, and refer to Figure 3.4.

First perturbation. We first apply the perturbation shown in Figure 3.4a which transforms the path $\Pi = S_1 C_3 S_5$ into a path $\Pi'(\varepsilon_1) = S_1' C_2' C_3' S_5'$ by reducing the

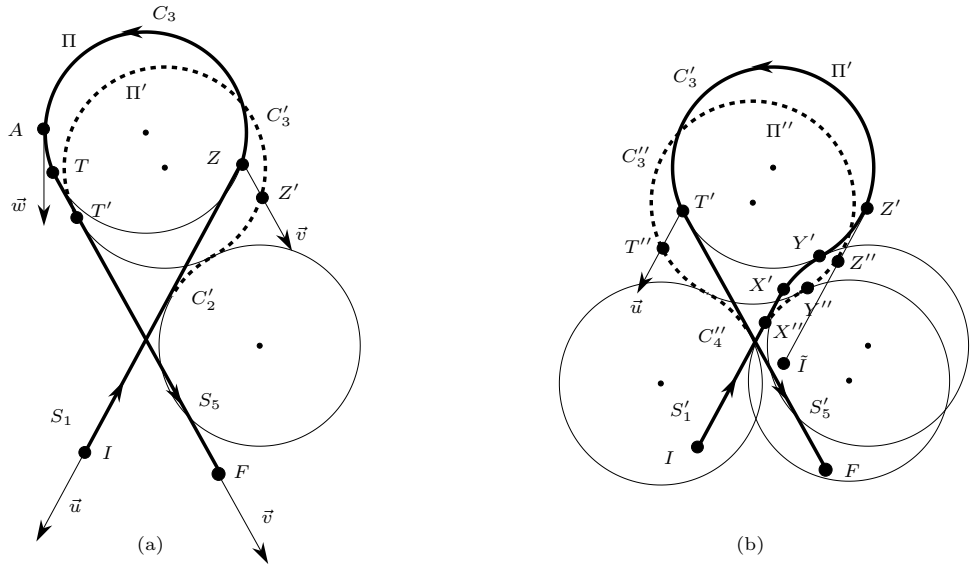


FIG. 3.4. *For the proof of Lemma* 3.4(i).

length of S_5 by a small value ε_1 (i.e., $\|S_5'\| = \|S_5\| - \varepsilon_1$), such that the new segment C_2' is oriented clockwise and is smaller than π. (Segment C_3' is naturally oriented counterclockwise.) For simplicity, in what follows, we denote $\Pi'(\varepsilon_1)$ by Π'.

CLAIM 3.7 (Dubins [15]). *In an obstacle-free environment, path* Π' *is shorter than* Π *for any* $\varepsilon_1 > 0$ *small enough.*

This directly yields that, if the optimal path $S_1C_3S_5$ is tangent to $\partial\mathcal{P}$ at the junction between C_3 and S_5, then the perturbed path Π' has shortened in $\mathcal{P}$ unless C_3 is strongly $\mathcal{PP}$-anchored. By symmetry, we get the same result if the path Π is tangent to $\partial\mathcal{P}$ at the junction between S_1 and C_3. Therefore, we can assume in the following that neither S_1 nor S_5 is touching $\partial\mathcal{P}$.

Second perturbation. We now transform (see Figure 3.4b) the path $\Pi' = S_1'C_2'C_3'S_5'$ into a path $\Pi''(\varepsilon_1, \varepsilon_2) = S_1''C_2''C_3''C_4''S_5''$ by reducing the length of S_1' by a small value ε_2 (i.e., $\|S_1''\| = \|S_1'\| - \varepsilon_2$), such that the length of C_2' does not change (i.e., $\|C_2''\| = \|C_2'\|$) and the new segment C_4'' is oriented clockwise and is smaller than π. (Segments C_2'' and C_3'' are naturally oriented clockwise and counterclockwise, respectively.) For simplicity, we denote $\Pi''(\varepsilon_1, \varepsilon_2)$ by Π''.

CLAIM 3.8. *In an obstacle-free environment, path* Π'' *is shorter than* Π' *for any* $\varepsilon_2 > 0$ *small enough.*

Proof. Refer to Figure 3.4b. Let Y' be the first point of C_3', and let Z' be the first point on C_3' such that the tangent to C_3 at Z' is parallel to the line segment S_1. Point Z' exists on C_3' because, by construction, $\|C_2'\| < \pi$ and $\|C_3'\| > \pi$. Furthermore, $\|C_2'\| = \|C_3'[Y', Z']\|$. We define similarly points Y'' and Z'' on C_3''. Then it follows from $\|C_2'\| = \|C_2''\|$ that $\|C_3'[Y', Z']\| = \|C_3''[Y'', Z'']\|$. Thus, if X' and X'' denote the first point of the segments C_2' and C_2'', respectively, we get $\|\Pi'[X', Z']\| = \|\Pi''[X'', Z'']\|$.

Now let $\tilde{I}$ be the point defined by $\overrightarrow{\tilde{I}Z'} = \overrightarrow{IX'}$ (or $\overrightarrow{I\tilde{I}} = \overrightarrow{X'Z'} = \overrightarrow{X''Z''}$). Let $\tilde{\Pi}'$ be the path from $\tilde{I}$ to F that is the concatenation of the line segment $\tilde{I}Z'$ and $\Pi'[Z', F]$. Similarly, let $\tilde{\Pi}''$ be the path from $\tilde{I}$ to F that is the concatenation of the line segment

$\tilde{I}Z''$ and $\Pi''[Z'', F]$. We get, by construction, that $\|\tilde{\Pi}'\| = \|\Pi'\| - \|\Pi'[X', Z']\|$ and $\|\tilde{\Pi}''\| = \|\Pi''\| - \|\Pi''[X'', Z'']\|$. Since $\|\Pi'[X', Z']\| = \|\Pi''[X'', Z'']\|$, $\|\tilde{\Pi}'\| - \|\tilde{\Pi}''\| = \|\Pi'\| - \|\Pi''\|$. However, we know that $\tilde{\Pi}''$ is shorter than $\tilde{\Pi}'$ because $\tilde{\Pi}''$ is obtained by the Dubins' length-reducing perturbation, shown in Figure 3.4a, applied on the reversed path of $\tilde{\Pi}'$, which is of type SCS. Thus Π'' is shorter than Π' in an obstacle-free environment for any $\varepsilon_2 > 0$ small enough. □

CLAIM 3.9. *If the C-segment C_3 in the optimal path $\Pi = S_1C_3S_5$ is not strongly $\mathcal{PP}$-anchored, then we can choose ε_1 and ε_2 arbitrarily small such that Π'' is free in $\mathcal{P}$.*

Proof. Let $\vec{u}$ be the unit vector whose direction is opposite the direction of S_1, and let $\vec{v}$ be the unit vector whose direction is the same as the direction of S_5. See Figure 3.4a.

In the two perturbations described above, the lengths of C_3 and C_3' increase. More precisely, the translated copy of C_3 by vector $\varepsilon_1\vec{v}$ is part of C_3', and the translated copy of C_3' by vector $\varepsilon_2\vec{u}$ is part of C_3''. Thus, if Z'' and T'' are the translated copies of the endpoints of C_3 by vector $\varepsilon_1\vec{v}+\varepsilon_2\vec{u}$, Π'' is the concatenation of $\Pi''[I, Z'']$, $\Pi''[Z'', T'']$, and $\Pi''[T'', F]$, where $\Pi''[Z'', T'']$ is the translated copy of C_3 by vector $\varepsilon_1\vec{v} + \varepsilon_2\vec{u}$. On the other hand, any arbitrarily small neighborhood around S_1 (respectively, S_5) contains $\Pi''[I, Z'']$ (respectively, $\Pi''[T'', F]$) for any ε_1 and ε_2 small enough. Thus, since we assumed that neither S_1 nor S_5 touches $\partial\mathcal{P}$, neither $\Pi''[I, Z'']$ nor $\Pi''[T'', F]$ touches $\partial\mathcal{P}$ for any ε_1 and ε_2 small enough. Thus it is sufficient to show that, if C_3 is not strongly $\mathcal{PP}$-anchored, then we can choose ε_1 and ε_2 arbitrarily small such that the translated copy of C_3 by vector $\varepsilon_1\vec{v} + \varepsilon_2\vec{u}$ is free.

Let A be the last point of tangency on C_3 with $\partial\mathcal{P}$. Let $\vec{w}$ be the unit vector tangent to C_3 at A (with direction corresponding to the orientation of C_3). If C_3 is not strongly $\mathcal{PP}$-anchored, then, for any $\mu > 0$ small enough, the translated copy of C_3 by $\mu\vec{w}$ is free. On the other hand, for λ_1 and λ_2 such that $\vec{w} = \lambda_1\vec{v} + \lambda_2\vec{u}$, the λ_1 and λ_2 are nonnegative because, by Lemma 2.4, the distance on C_3 between the first point of C_3 and A is greater than π. Therefore, if C_3 is not strongly $\mathcal{PP}$-anchored, then, for any $\mu > 0$ small enough, the path Π'' defined with $\varepsilon_1 = \mu\lambda_1$ and $\varepsilon_2 = \mu\lambda_2$ is free in $\mathcal{P}$. □

By Claims 3.7, 3.8, and 3.9, if the C-segment C_3 in the optimal path $\Pi = S_1C_3S_5$ is not strongly $\mathcal{PP}$-anchored, then we can choose ε_1 and ε_2 arbitrarily small so that Π'' is free and shorter than Π. This concludes the proof of Lemma 3.4(i).

3.1.2. Proof of Lemma 3.4(ii). We prove Lemma 3.4(ii), which states

> *If an optimal path has a subpath of type $C_1C_2C_3S$ (or $SC_3C_2C_1$) so that the C-segment C_2 does not touch $\partial\mathcal{P}$, then C_3 is strongly $\mathcal{PP}$-anchored.*

Consider an optimal path $\Pi = C_1C_2C_3S$ from configuration I to configuration F so that the C-segment C_2 does not touch $\partial\mathcal{P}$. We prove that the C-segment C_3 is strongly $\mathcal{PP}$-anchored. Assume without loss of generality that C_1 is oriented counterclockwise. We assume, for a contradiction, that C_3 is not strongly $\mathcal{PP}$-anchored and show that the path Π can be shortened in $\mathcal{P}$.

Notation. Let O_i, $i = 1, 2, 3$, denote the center of the circle supporting the C-segment C_i of Π. See Figure 3.5a. Let X denote the point of tangency between the C-segments C_2 and C_3, and let Y denote the antipodal point of X on C_3 ($\|C_3\| > \pi$ by Lemma 2.3). Let Z be the point of tangency between the C-segment C_3 and the S-segment. Let T be the last point of tangency with $\partial\mathcal{P}$ on the (oriented) circular arc $\Pi[Y, Z]$; such a point exists by Lemma 2.4. Let T' be the antipodal point of T on C_3.

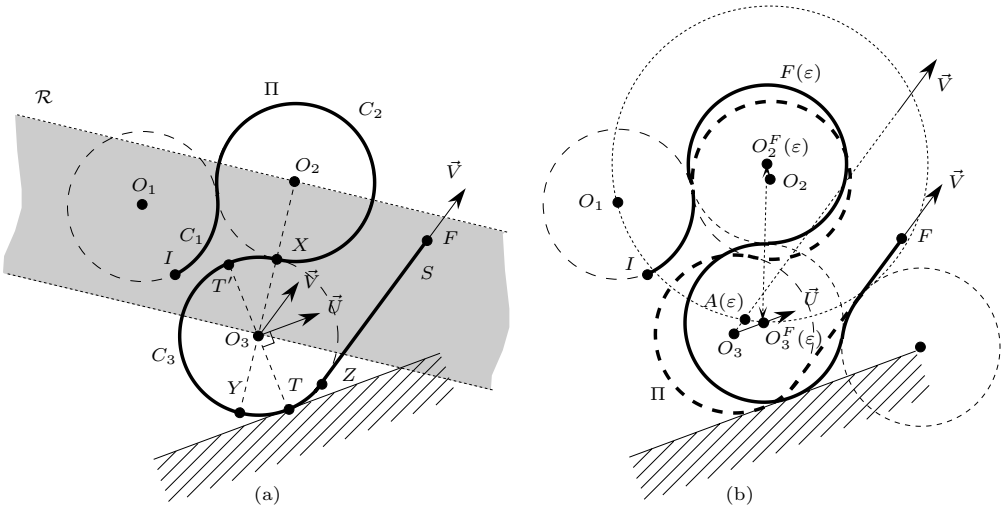


FIG. 3.5. (a) *Path* Π. (b) *A shorter path* $F(\varepsilon)$.

By definition, the circular arc $\Pi(T, Z]$ does not touch $\partial\mathcal{P}$. Furthermore, the circular arc $\Pi[X, T')$ does not touch $\partial\mathcal{P}$ because otherwise C_3 would be strongly $\mathcal{PP}$-anchored. Let $\vec{U}$ be a unit vector tangent to Π at T, and let $\vec{V}$ be a unit vector tangent to Π at any point on the S-segment of Π. Finally, let $ray(O_3, \vec{U})$ and $ray(O_3, \vec{V})$ denote the rays starting at O_3 in the directions $\vec{U}$ and $\vec{V}$, respectively.

First perturbation. Consider the perturbation shown as a thick solid path in Figure 3.5b, where Π has been perturbed into a path $F(\varepsilon)$ of type $CCCCS$. We define this perturbed path $F(\varepsilon)$ by specifying the position of the center $O_3^F(\varepsilon)$ of the supporting circle of its third circular arc, namely, $O_3^F(\varepsilon) = O_3 + \varepsilon\vec{U}$. The path $F(\varepsilon)$ is well defined for any ε small enough because the unit circle centered at $O_3^F(\varepsilon)$ intersects the S-segment of Π (by definition of $\vec{U}$); thus the fourth circular arc of $F(\varepsilon)$ is defined. Also, since the second C-segment of Π and the arcs $\Pi[X, T')$ and $\Pi(T, Z]$ do not touch $\partial\mathcal{P}$, $F(\varepsilon)$ is free for any sufficiently small values of $\varepsilon > 0$.

Second perturbation. To prove that the length of $F(\varepsilon)$ is a decreasing function of ε, we define a path $K_h(k)$ with two perturbation steps. The first step perturbs Π to a path $H(h)$, and the second step then perturbs $H(h)$ to a path $K_h(k)$. As we will show later, ε, h, and k can be chosen so that $F(\varepsilon) = K_h(k)$. Furthermore, we will show that $K_h(k)$ is shorter than Π.

Below, let $O_i^H(h)$, $O_i^K(k)$, $i = 1, 2, 3, 4$, denote the center of the circle supporting the ith C-segment of paths $H(h)$ and $K_h(k)$, respectively.

First step. The path $H(h)$ (see Figure 3.6a) is of type $CCCS$ joining I to F such that $O_1^H(h)$ is identically equal to O_1, $O_3^H(h) = O_3 + h\vec{V}$, and the length of the second circular arc is greater than π. For any h small enough, $H(h)$ is well defined.

Second step. The path $K_h(k)$ (see Figure 3.7) is defined as follows. For a given $H(h)$ and for a sufficiently small $k > 0$, path $K_h(k)$ is of type $CCCCS$ from I to F such that $O_1^K(k)$ and $O_2^K(k)$ are identically equal to O_1 and $O_2^H(h)$, respectively, $O_3^K(k)$ is at distance k counterclockwise from $O_3^H(h)$ along the circular arc of radius 2 centered at $O_2^K(k)$, and the fourth circular arc has length less than π.

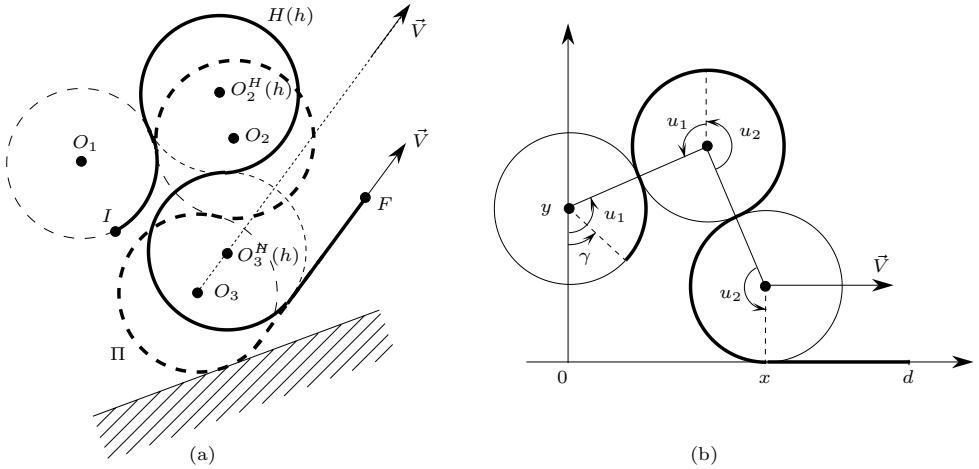


FIG. 3.6. *Length reducing perturbation: The path $H(h)$ is shorter than* Π.

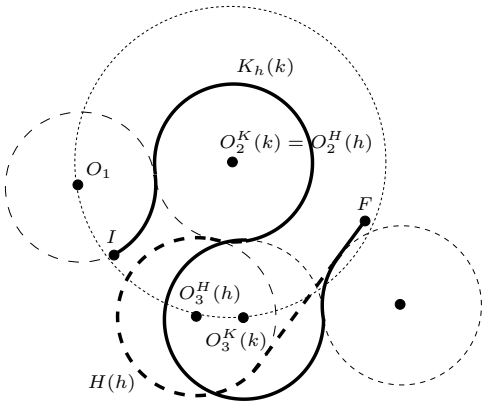


FIG. 3.7. *Dubins' length reducing perturbation: The path $K_h(k)$ is shorter than $H(h)$.*

CLAIM 3.10. *For a given sufficiently small $\varepsilon > 0$, h and k can be chosen such that $F(\varepsilon) = K_h(k)$.*

Proof. Let $\mathcal{R}$ be the open strip bounded by two lines perpendicular to the line O_2O_3 and passing through O_2 and O_3, respectively (see Figure 3.5a). Since the lengths of the circular arcs $\Pi[X,T]$ and $\Pi[X,Z]$ are greater than π by Lemmas 2.4 and 2.3(i), respectively, $ray(O_3,\vec{U})$ and $ray(O_3,\vec{V})$ are directed into $\mathcal{R}$. Thus, for any sufficiently small ε, $O_3^F(\varepsilon)$ belongs to $\mathcal{R}$, and the distance between $O_3^F(\varepsilon)$ and O_2 is less than 2. Consequently, it can be shown that $O_2^F(\varepsilon)$ must lie outside $\mathcal{R}$ on the circle of radius 2 centered at O_1. It follows that, for any open neighborhood $\mathcal{N}$ of O_3, any choice of ε sufficiently small ensures that the circle of radius 2 centered at $O_2^F(\varepsilon)$ intersects $ray(O_3,\vec{U})$ and $ray(O_3,\vec{V})$ in $\mathcal{N}$ (see Figure 3.5b). Let $A(\varepsilon)$ denote the intersection of that circle with $ray(O_3,\vec{V})$ in $\mathcal{N}$; recall that the intersection, in $\mathcal{N}$, of that circle with $ray(O_3,\vec{U})$ is $O_3^F(\varepsilon)$. The polar angle of $ray(O_3,\vec{V})$ is bigger than the polar angle of $ray(O_3,\vec{U})$ by an amount smaller than π. Thus, for ε small enough, the counterclockwise oriented arc, denoted $arc(A(\varepsilon),O_3^F(\varepsilon))$, of the circle of radius 2 centered at $O_2^F(\varepsilon)$ starting at $A(\varepsilon)$ and ending at $O_3^F(\varepsilon)$ is also contained in

the neighborhood $\mathcal{N}$; indeed, since $A(\varepsilon)$ and $O_3^F(\varepsilon)$ converge to O_3 when ε tends to 0, $arc(A(\varepsilon), O_3^F(\varepsilon))$ also tends to O_3 when ε tends to 0. Therefore, we can choose ε such that the line segment $[O_3, A(\varepsilon)]$ and the circular arc $arc(A(\varepsilon), O_3^F(\varepsilon))$ are arbitrarily small.

Choose h equal to the length of the line segment $[O_3, A(\varepsilon)]$, and choose k equal to the length of the circular arc $arc(A(\varepsilon), O_3^F(\varepsilon))$. Then $O_3^H(h) = A(\varepsilon)$ and $O_3^K(k) = O_3(\varepsilon)$, and therefore $K(k) = F(\varepsilon)$. Moreover, we have shown that we can choose ε small enough such that h and k are arbitrarily small. □

CLAIM 3.11. *The length of $K_h(k)$ is strictly smaller than the length of Π for any h and k sufficiently small.*

Proof. The length of $K_h(k)$ has been shown by Dubins [15] to be strictly shorter than the length of $K_h(0) = H(h)$ for any fixed h and for any small enough $k > 0$. Furthermore, the length of $K_h(0) = H(h)$ has been shown in [7] to be strictly shorter than the length of Π. For completeness, we give the proof here. Consider a path of type $CCCS$ such that the length of the second circular arc is greater than π. With the notation of Figure 3.6b, the length of the path is equal to $L = 2(u_1 + u_2) - \gamma + d - x$, where γ and d are some constants. Furthermore, we have

$$\begin{cases} \sin(u_1) + \sin(u_2) = x/2, \\ \cos(u_1) - \cos(u_2) = (y-1)/2. \end{cases}$$

By computing the derivative of each equation with respect to x and solving the system, we obtain the following solution (which is defined, since $(u_1 + u_2) \in (\pi, 2\pi)$, by hypothesis):

$$\begin{cases} \dfrac{\partial u_1}{\partial x} = \dfrac{\sin(u_2)}{2\sin(u_1+u_2)}, \\ \dfrac{\partial u_2}{\partial x} = \dfrac{\sin(u_1)}{2\sin(u_1+u_2)}. \end{cases}$$

Therefore,

$$\frac{\partial L}{\partial x} = \frac{\sin(u_1) + \sin(u_2)}{\sin(u_1+u_2)} - 1 = \frac{\cos\left(\frac{u_1-u_2}{2}\right) - \cos\left(\frac{u_1+u_2}{2}\right)}{\cos\left(\frac{u_1+u_2}{2}\right)}.$$

Since u_1 and u_2 are positive and $(u_1 + u_2) \in (\pi, 2\pi)$, $0 \leqslant \frac{|u_1-u_2|}{2} < \frac{|u_1+u_2|}{2} < \pi$, and thus $\cos\left(\frac{u_1-u_2}{2}\right) > \cos\left(\frac{u_1+u_2}{2}\right)$. Furthermore, $\cos\left(\frac{u_1+u_2}{2}\right) < 0$ since $\frac{u_1+u_2}{2} \in (\frac{\pi}{2}, \pi)$. Therefore, $\partial L/\partial x$ is negative, which means that, as long as $H(h)$ is of type $CCCS$ with the second circular arc greater than π, the length of $H(h)$ is a decreasing function of h. Hence we have shown that the length of $K_h(k)$ is smaller than the length of Π for any h and k sufficiently small. □

To complete the proof of Lemma 3.4(ii), recall that $F(\varepsilon)$ is free in $\mathcal{P}$ for any ε small enough. By Claims 3.10 and 3.11, there exist arbitrarily small values of ε, h, k such that $F(\varepsilon) = K_h(k)$, where the length of $K_h(k)$ is strictly less than the length of Π. This contradicts the optimality of Π and completes the proof.

4. A simple algorithm. Theorem 3.1 can be used to obtain the following simple algorithm for computing an optimal path inside $\mathcal{P}$. We enumerate candidate paths of types described in Theorem 3.1. Our candidate set is guaranteed to contain an optimal path, if any exist. For each such path, we check whether it is feasible and, if so, compute its length. Finally, we either return the shortest feasible path or report that no feasible path exists.

In order to determine whether a path is feasible, we rely on the circle-shooting data structure by Cheng et al. [10] that preprocesses $\mathcal{P}$ in $O(n \log n)$ time into a data structure of linear size that makes it possible to determine in $O(\log n)$ time whether a given circular arc of unit radius intersects $\partial\mathcal{P}$. This immediately implies the following lemma.

LEMMA 4.1. *$\mathcal{P}$ can be preprocessed in $O(n \log n)$ time into a data structure of linear size that enables us to determine in $O(m \log n)$ time whether a given path consisting of m C- and S-segments is feasible.*

To bound the running time of this simple algorithm, we must count the number of candidate paths to check. We note that, once a path type is given and the supporting circles for C-segments are known, there are $O(1)$ candidate paths. These are determined by the choices of the orientations for the C-segments. Hence we are interested in the number of possible supporting circles for each path type. Note that there may be $\Omega(n^2)$ $\mathcal{PP}$-anchored circles (see Lemma 5.3) and $\Omega(n)$ $\mathcal{PC}$-anchored circles.

There are $O(1)$ Dubins path candidates. For paths of types (A.i) and (B.ii), once the $\mathcal{PP}$-anchored circle is chosen, there are $O(1)$ choices for other supporting circles and hence $O(1)$ candidate paths. Since there are $O(n^2)$ $\mathcal{PP}$-anchored circles, there are $O(n^2)$ candidate paths for these two path types.

A path of type (A.ii) may have up to two $\mathcal{PC}$-anchored segments. Once their supporting circles are chosen, there are $O(1)$ path candidates. There are $O(n)$ potential $\mathcal{PC}$-anchored circles. If both anchored segments are present, we have $O(n^2)$ paths to check; otherwise, we have only $O(n)$. Paths of type (B.i) are also determined by a $\mathcal{PC}$-anchored circle; hence there are $O(n)$ of them.

Paths of type (B.iii), i.e., of type $C_I\bar{\bar{C}}_1S\bar{C}_2\bar{C}_3S\bar{\bar{C}}_4C_F$, present a special problem. If we know the supporting circles of the $\bar{C}\bar{C}$ subpath, the rest of the path is determined by a pair of $\mathcal{PC}$-anchored circles $\mathcal{C}_1, \mathcal{C}_4$, for which there are $O(n^2)$ possibilities. Unfortunately, there is an infinite family of supporting circles for the $\bar{C}\bar{C}$ subpath. The following result by Boissonnat and Lazard [6] allows us to consider only a finite set of $\bar{C}\bar{C}$ subpaths.

LEMMA 4.2 (Boissonnat and Lazard [6]). *Given two configurations X and Y and two edges e, e' of $\mathcal{P}$, we can compute[4] in $O(1)$ time a finite set of paths from X to Y of type $C_1S\bar{C}_2\bar{C}_3SC_4$, where $\bar{C}_2$ and $\bar{C}_3$ are tangent to edges e and e', respectively. This set contains all optimal paths from X to Y of type $C_1S\bar{C}_2\bar{C}_3SC_4$.*

Given a pair of edges e, e' and a pair of $\mathcal{PC}$-anchored circles $\mathcal{C}_1, \mathcal{C}_4$, tangent to C_I and C_F, respectively, we choose X to be the configuration determined by the intersection of C_I and $\mathcal{C}_1$ and Y to be the configuration determined by C_F and $\mathcal{C}_4$. By the above lemma, we can compute in $O(1)$ time a constant number of candidate paths for this pair of edges and anchored circles. Doing this for all possible pairs of edges (e, e') and pairs of supporting circles $(\mathcal{C}_1, \mathcal{C}_4)$, we determine $O(n^4)$ path candidates of type (B.iii) in $O(n^4)$ time.

In summary, the simple algorithm examines $O(n^4)$ candidate paths and, for each, spends $O(\log n)$ time checking feasibility, by Lemma 4.1 with $m \leqslant 8$. Therefore, the overall running time is $O(n^4 \log n)$.

PROPOSITION 4.3. *Given a convex polygon $\mathcal{P}$, an initial configuration I, and a final configuration F, an optimal path from I to F inside $\mathcal{P}$ can be computed in time $O(n^4 \log n)$.*

[4]The computation is performed by solving four algebraic systems of three equations in three indeterminates.

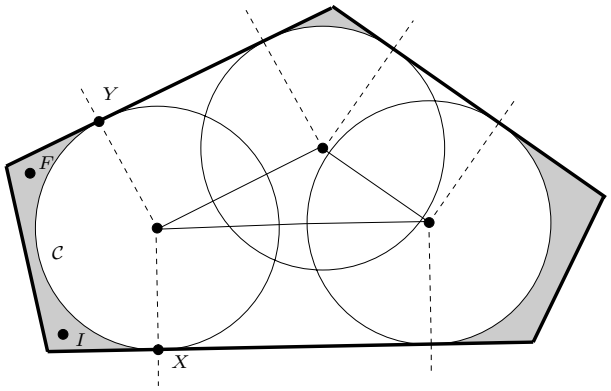


FIG. 5.1. *The free circle $\mathcal{C}$ is centered on the retracted polygon of $\mathcal{P}$.*

5. An efficient algorithm. In this section, we prove additional properties of optimal paths that significantly reduce the number of candidates to examine. We have already shown that we need to consider only $O(1)$ Dubins paths and $O(n)$ candidates for paths of type (B.i). We will show that it suffices to consider only $O(1)$ candidate paths of types (A.i) and (B.ii), $O(n)$ candidate paths of type (A.ii), and $O(n^2)$ candidate paths of type (B.iii).

5.1. Computing paths of types (A.i) and (B.ii). The paths of types (A.i) and (B.ii) contain a strongly $\mathcal{PP}$-anchored C-segment $\bar{\bar{C}}$. The circle $\bar{\bar{\mathcal{C}}}$ supporting $\bar{\bar{C}}$ defines one or two pockets that contain a point of tangency of $\bar{\bar{\mathcal{C}}}$ with $\partial\mathcal{P}$ (see Figures 2.4b and 3.2). By Lemma 2.7, we know that I and F must belong to these pockets. The following lemma states that there exist at most two circles with these properties and that they can be computed efficiently.

LEMMA 5.1. *For a fixed pair of locations I, F, there exist at most two circles that can support a strongly $\mathcal{PP}$-anchored C-segment appearing in an optimal path from I to F, and they can be computed in $O(n)$ time.*

Proof. Consider a strongly $\mathcal{PP}$-anchored segment that lies in an optimal path. Let X and Y denote its points of tangency with $\partial\mathcal{P}$, and let $\mathcal{C}$ denote its supporting circle. Assume without loss of generality that the short arc on $\mathcal{C}$ joining X and Y is $\mathcal{C}^-[X, Y]$ (see Figure 5.1). The proof of the lemma is divided into two cases, which depend on whether or not $\mathcal{C}$ is free.

Case 1. *$\mathcal{C}$ is free.* The center of $\mathcal{C}$ lies at a vertex of the retracted polygon of $\mathcal{P}$ (i.e., the set of points p such that the unit circle centered at p lies inside $\mathcal{P}$). By computing the retracted polygon of $\mathcal{P}$ in linear time, we get (in linear time) the set of the $O(n)$ free $\mathcal{PP}$-anchored circles which contains $\mathcal{C}$. Each of these circles defines one pocket, and all of these pockets are pairwise disjoint (see Figure 5.1). Thus only one of these pockets contains I and F; hence, by Lemma 2.7, $\mathcal{C}$ must be the circle defining this pocket. For each of the $O(n)$ free $\mathcal{PP}$-anchored circles, we can easily check, in $O(1)$ time, whether I and F belong to the corresponding pocket. Indeed (see Figure 5.1), $I, F \in \mathcal{P}$ belong to the pocket if and only if I and F are outside the circle and are in the small wedge defined by the rays emanating from the center of the circle and passing through the points of tangency of the circle with $\partial\mathcal{P}$.

Case 2. *$\mathcal{C}$ is not free.* $\mathcal{C}$ defines two pockets $\Lambda_{\mathcal{C}}[X, X']$ and $\Lambda_{\mathcal{C}}[Y, Y']$ (see Figure 5.2). By Lemma 2.7, one of these pockets contains I, and the other contains F. Thus $I \neq F$, and the line L_{IF} through I and F is defined. Let A and B denote the intersection points of L_{IF} with $\partial\mathcal{P}$.

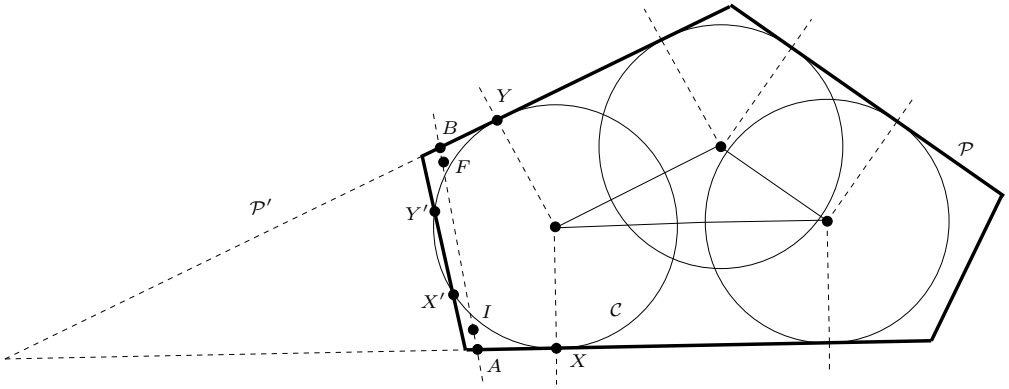


FIG. 5.2. *The nonfree circle $\mathcal{C}$ is centered on the retracted chain of $\partial\mathcal{P}^+[A,B]$.*

With no loss of generality, suppose $I \in \Lambda_{\mathcal{C}}[X,X']$ and $F \in \Lambda_{\mathcal{C}}[Y,Y']$. The segment $[I,F]$ must pass through $\mathcal{C}$ twice since I and F are in distinct pockets. Thus the ray emanating from I in the direction $\overrightarrow{FI}$ cannot intersect $\mathcal{C}$ and therefore leaves $\Lambda_{\mathcal{C}}[X,X']$ through $\partial\mathcal{P}^-[X,X']$. Thus $A \in \partial\mathcal{P}^-[X,X']$. A similar argument shows that $B \in \partial\mathcal{P}^+[Y,Y']$. Hence $X', Y' \notin \partial\mathcal{P}^+[A,B]$, and $X, Y \in \partial\mathcal{P}^+[A,B]$.

The chain $\partial\mathcal{P}^+[A,B]$ does not properly intersect $\mathcal{C}$. Indeed (see Figure 5.2), it properly intersects neither the long arc $\mathcal{C}^+[X,Y]$, by assumption, nor the small arc $\mathcal{C}^-[X,Y]$ because the first intersection between $\mathcal{C}^-[X,Y]$ (respectively, $\mathcal{C}^+[Y,X]$) and $\partial\mathcal{P}$ is X' (respectively, Y'), which does not belong to $\partial\mathcal{P}^+[A,B]$. It then follows from $X, Y \in \partial\mathcal{P}^+[A,B]$ that the circle $\mathcal{C}$ is a free anchored circle in the polygon $\mathcal{P}'$ obtained by extending the two edges of $\partial\mathcal{P}^+[A,B]$ that end at A, B (see Figure 5.2). Moreover, the pocket defined by $\mathcal{C}$ in $\mathcal{P}'$ contains the two pockets $\Lambda_{\mathcal{C}}[X,X']$ and $\Lambda_{\mathcal{C}}[Y,Y']$ and thus contains I and F. As before, at most one free anchored circle in $\mathcal{P}'$ defines a pocket containing I and F, and, given $\mathcal{P}'$, it can be computed in $O(n)$ time.

Note finally that polygon $\mathcal{P}'$ can be determined in $O(n)$ time. This is because I and F determine the points A and B, and the turning angle of $\partial\mathcal{P}^+[A,B]$ is bigger than π. Thus, independent of any assumption about the orientation of the short arc on $\mathcal{C}$ joining X and Y, whether $\mathcal{P}'$ is the polygon obtained by extending the two edges (ending at A, B) of $\partial\mathcal{P}^+[A,B]$ or of $\partial\mathcal{P}^-[A,B]$ can be determined simply by checking which of the turning angles of $\partial\mathcal{P}^+[A,B]$ and $\partial\mathcal{P}^-[A,B]$ is bigger than π.

Hence we have proved that, for a fixed pair of locations I and F, there exist at most two $\mathcal{PP}$-anchored circles (one free and the other nonfree) that can appear in an optimal path from I to F, and they can be computed in $O(n)$ time. □

By the above lemma, we can compute, in $O(n)$ time, a set of $O(1)$ candidate paths of types (A.i) and (B.ii). The candidate paths may be checked for feasibility in $O(\log n)$ time. Therefore, we have the following lemma.

LEMMA 5.2. *An optimal path of type* (A.i) *or* (B.ii) *can be computed in $O(n)$ time.*

The following proposition shows that Lemma 5.1 is essential for establishing the linear running time given in Lemma 5.2; indeed, an algorithm that checked all candidate strongly $\mathcal{PP}$-anchored circles to produce candidate paths would have running time $\Omega(n^2)$.

PROPOSITION 5.3. *There exist convex n-polygons that have $\Omega(n^2)$ strongly $\mathcal{PP}$-anchored circles.*

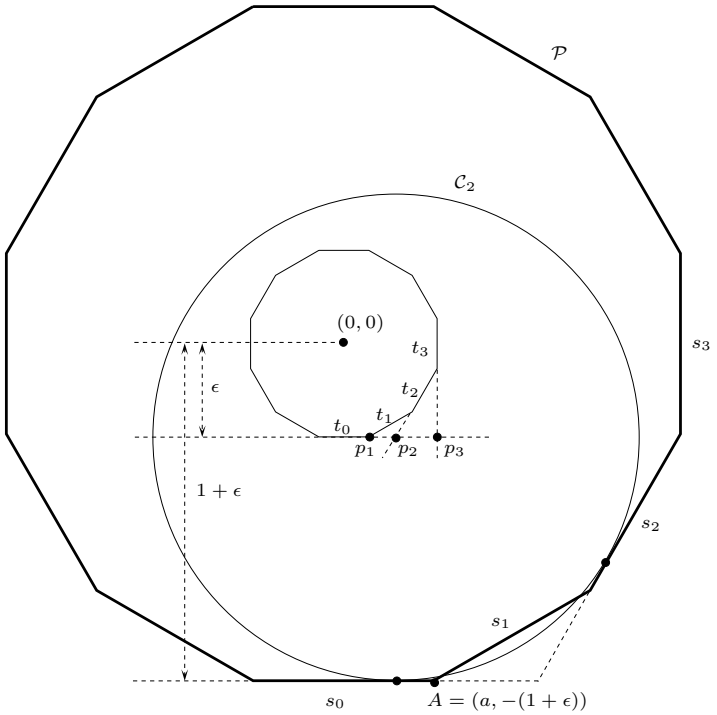


FIG. 5.3. *The circles $\mathcal{C}_i$ centered at p_i, $i = 1,\ 2,\ 3 = \frac{n}{4}$, are strongly $\mathcal{PP}$-anchored: They are tangent to s_0 and s_i, and their long arcs are free.*

Proof. Let $n \geqslant 8$ be a multiple of 4, and let $0 < \epsilon < \tan(\pi/n)$ be a real parameter. Let $\mathcal{P}$ be an n-regular polygon centered at the origin with in-radius $(1+\epsilon)$; i.e., the distance from the origin to each side of $\mathcal{P}$ is $(1+\epsilon)$. We assume that one of the edges, say, s_0, of $\mathcal{P}$ is parallel to the x-axis and lies below the x-axis; see Figure 5.3. The coordinates of the right endpoint of s_0 are $(a, 0)$, where $a = (1+\epsilon)\tan(\pi/n)$. Let the edges of $\mathcal{P}$ in the counterclockwise sense be $s_0, s_1, s_2, \ldots, s_{n-1}$.

The retracted polygon of $\mathcal{P}$ (i.e., the set of points p such that the unit circle centered at p lies inside $\mathcal{P}$) is an n-regular polygon with radius ϵ. Denote its sides by $t_0, t_1, \ldots, t_{n-1}$, where t_i is the retraction of s_i for $0 \leqslant i < n$. For $i = 1, \ldots, n/4$, denote by $p_i = (x_i, -\epsilon)$ the intersection point of the lines supporting t_i and t_0 and by $\mathcal{C}_i$ the unit circle centered at p_i. It is easily seen that $0 < x_1 < \cdots < x_{n/4} \leqslant \epsilon$.

Since the x-coordinate of the point at which $\mathcal{C}_i$ touches the line supporting s_0 is x_i and $x_i < \epsilon < \tan(\pi/n) < a$, $\mathcal{C}_i$ is tangent to s_0 for any $1 \leqslant i \leqslant n/4$. A symmetric argument shows that $\mathcal{C}_i$ is tangent to s_i.

Therefore, we can assign $n/4$ $\mathcal{PP}$-anchored circles to every side of $\mathcal{P}$, and the number of $\mathcal{PP}$-anchored circles of $\mathcal{P}$ is $\Omega(n^2)$. It remains to show that these $\mathcal{PP}$-anchored circles are strongly $\mathcal{PP}$-anchored.

As can be seen from Figure 5.3, the point p_i is a vertex of the polygon defined by the two lines through t_0 and t_i and the edges $t_{i+1}, \ldots, t_{n-1}$. Thus p_i is in the retracted polygon of the polygon formed by the two lines through s_0 and s_i and the edges $s_{i+1}, \ldots, s_{n-1}$. Therefore, $\mathcal{C}_i$ does not properly intersect any of $s_{i+1}, \ldots, s_{n-1}$, and its long arc is free, so $\mathcal{C}_i$ is indeed a strongly $\mathcal{PP}$-anchored circle. ∎

5.2. A monotonicity property of CCSC paths. Subpaths of type $CCSC$ occur in both (A.ii) and (B.iii) path types. In this subsection, we ignore the polygon $\mathcal{P}$ and study paths from X to Y of type $C_1C_2SC_3$, with specified orientations on C_1 and C_3; the orientation of C_1 fixes the orientation of C_2; namely, if C_1 is oriented clockwise (respectively, counterclockwise), then C_2 is oriented counterclockwise (respectively, clockwise).

The positions of circles $\mathcal{C}_1$ and $\mathcal{C}_3$, supporting C_1 and C_3, respectively, are considered fixed, while the position of circle $\mathcal{C}_2$ is determined by M, its point of tangency with $\mathcal{C}_1$. The orientations of these three circles are fixed by the orientations of the corresponding C-segments. The S-segment is determined by the appropriate tangent line, given the orientations on C_2 and C_3. This tangent, if it exists, is unique.

For each $M \in \mathcal{C}_1$, there is at most one path, denoted $\Pi(M)$, from X to Y of type $C_1C_2SC_3$ with the specified orientations on the C_1- and C_3-segments. We are interested in how the path length $\|\Pi(M)\|$ changes as M moves along $\mathcal{C}_1$ in the same direction as the segment C_1.

For certain positions of M, one or more of the segments of $\Pi(M)$ may vanish. For example, when $M = X$, the length of the first segment C_1 changes discontinuously from 2π to 0. At such points, the path length may change discontinuously, so these positions of M are called *singular points* of $\Pi(M)$.

LEMMA 5.4. *Given two configurations X and Y, the paths $\Pi(M)$ of type $C_1C_2SC_3$ from X to Y with specified orientations on the C_1- and C_3-segments admit at most six singular points, and their locations can be computed in $O(1)$ time.*

Proof. We enumerate the possibilities for a segment to vanish in the paths $\Pi(M)$. Figure 5.4 illustrates the six singular points in a path of type $C_1^+C_2^-SC_3^+$.

Segment C_1 vanishes if and only if $M = X$, so X is the only singular point such that C_1 vanishes.

Segment C_2 vanishes if and only if the path type degenerates to C_1SC_3. Then the point M on $\mathcal{C}_1$ is also on the S-segment. Since there is at most one S-segment tangent to $\mathcal{C}_1$ and $\mathcal{C}_3$ that respects their specified orientations, there is at most one singular point $M_1 \in \mathcal{C}_1$ such that C_2 vanishes.

Segment S vanishes if and only if the path type degenerates to either $C_1C_2C_3$ if C_2 and C_3 have opposite orientations or C_1C_2 otherwise. Thus there are at most two singular points $M_2, M_3 \in \mathcal{C}_1$ (common point of C_1 and C_2) such that S vanishes.

Segment C_3 vanishes if and only if the path type degenerates to C_1C_2S. This means that L_Y, the line passing through the configuration Y, is tangent to $\mathcal{C}_2$. There are at most two circles $\mathcal{C}_2$ tangent to $\mathcal{C}_1$ and L_Y that respect the orientations of $\mathcal{C}_1$ and L_Y. Thus there are at most two singular points M_4, M_5 such that C_3 vanishes.

In total, there are no more than six singular points, and they can clearly be computed in $O(1)$ time. □

We next state the monotonicity property of the paths $\Pi(M)$.

LEMMA 5.5. *$\|\Pi(M)\|$ strictly increases as M moves along the oriented circle $\mathcal{C}_1$ between singular points, except when $\mathcal{C}_1$ and $\mathcal{C}_3$ are identical with the same orientation, in which case $\|\Pi(M)\|$ is constant as M moves between singular points.*

Proof. There are four possible orientation assignments for the circles: $C_1^+C_2^-SC_3^+$, $C_1^+C_2^-SC_3^-$, $C_1^-C_2^+SC_3^-$, and $C_1^-C_2^+SC_3^+$. We prove the claim for the first two cases; the other two cases can be proved using a symmetric argument.

Consider a path $\Pi(M)$. Let $\alpha_1 = \|\mathcal{C}_1^+[X, M]\|$ be the length of the first C-segment. Since there is a one-to-one mapping between α_1 and M, we can parameterize Π by α_1. Let $\alpha_2 = \alpha_2(\alpha_1)$, $\alpha_3 = \alpha_3(\alpha_1)$ be the length of the second and third C-segments

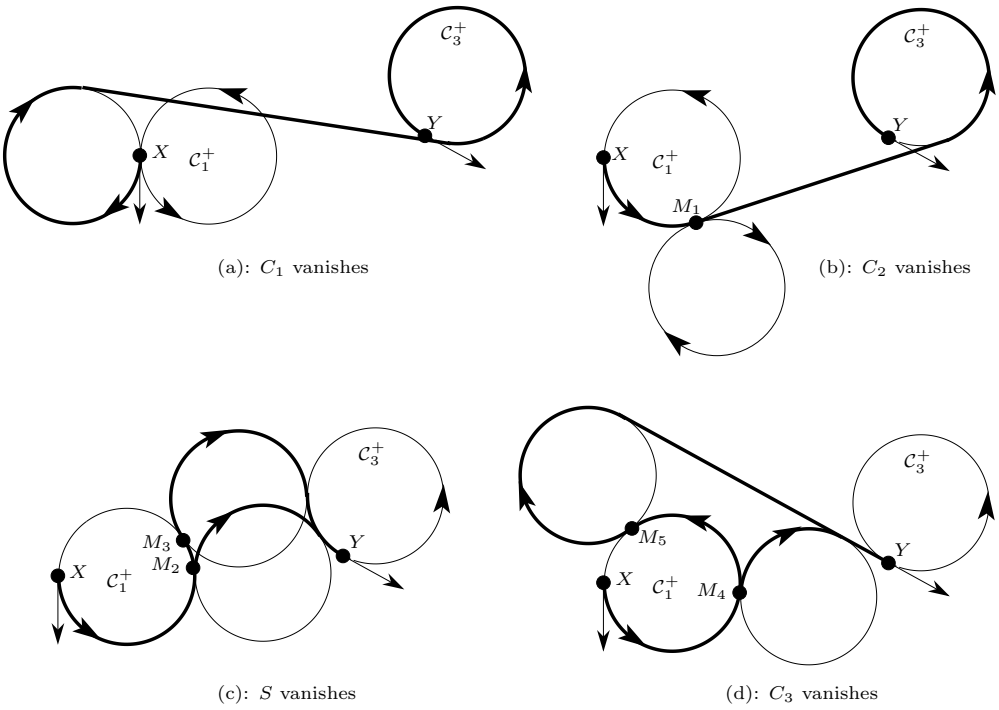


FIG. 5.4. *Paths of type $C_1^+C_2^-SC_3^+$ from X to Y and the six singular points X, M_1, M_2, M_3, M_4, and M_5 on $\mathcal{C}_X^+$.*

of $\Pi(\alpha_1)$, and let $2s = 2s(\alpha_1)$ be the length of the S-segment of $\Pi(\alpha_1)$. Let O_i be the center of the circle $\mathcal{C}_i$ supporting C_i, $i = 1, 2, 3$. Although O_1 and O_3 are fixed, O_2 depends on α_1. By definition,

$$L(\alpha_1) = \|\Pi(\alpha_1)\| = \alpha_1 + \alpha_2 + 2s + \alpha_3. \tag{5.1}$$

As M moves continuously on $\mathcal{C}_1$, the length of every segment in path $\Pi(M)$ changes continuously, except at singular points and at points for which $\Pi(M)$ is not defined (i.e., when $\mathcal{C}_2$ and $\mathcal{C}_3$ have opposite orientation and properly intersect). It follows that the segment lengths are piecewise differentiable functions of α_1 and that L is a piecewise differentiable function of α_1 on the intervals of $[0, 2\pi)$ where the path $\Pi(\alpha_1)$ is defined. For a function $f(\alpha_1)$, we will use $f'(\alpha_1)$ to denote $\partial f/\partial\alpha_1$. Then

$$L'(\alpha_1) = 1 + \alpha_2' + 2s' + \alpha_3'. \tag{5.2}$$

We call a value of α_1 singular if the corresponding point M on $\mathcal{C}_1$ is singular. The lemma can now be restated as follows: In the open intervals between singular points, $L' > 0$ almost everywhere (i.e., at all but a finite number of points) except when $O_1 = O_3$ and $\Pi(M)$ is of type $C_1^+C_2^-SC_3^+$, in which case $L' = 0$ everywhere. The proof is divided into two cases depending on whether O_1 and O_3 are equal.

Case 1. *O_1 is distinct from O_3.* Consider the triangle $\triangle O_1O_2O_3$. See Figure 5.5. We have $\|O_1O_2\| = 2$; let $d = \|O_1O_3\|$, and let $2t = \|O_2O_3\|$. We also define two (counterclockwise) oriented angles $\alpha = \angle O_3O_1O_2$ and $\beta = \angle O_1O_2O_3$. Both angles depend on α_1. Since C_1 is oriented counterclockwise, $\alpha_1 - \alpha$ is a constant, and therefore $\alpha' = 1$.

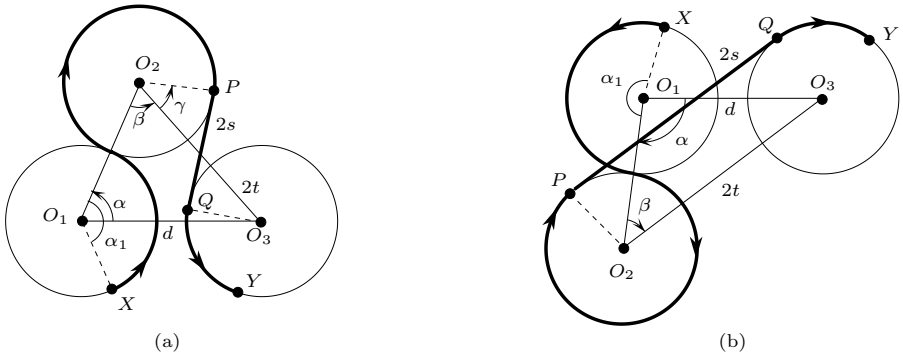


FIG. 5.5. *Paths* $\Pi(M)$ *of types* (a) $C_1^+C_2^-SC_3^+$ *and* (b) $C_1^+C_2^-SC_3^-$.

In view of the above discussion, it is sufficient to prove that $L'(\alpha_1) > 0$ for any nonsingular value of α_1 such that the path $\Pi(\alpha_1)$ is defined and $\alpha \not\equiv 0 \pmod{\pi}$; indeed, there are only a finite number of values α_1 such that $\alpha \equiv 0 \pmod{\pi}$. Since α_1 is not singular, we can assume in the following that $t \neq 0$.

By applying the cosine law to $\triangle O_1O_2O_3$, we obtain

$$4t^2 = 4 + d^2 - 4d\cos\alpha.$$

By differentiating the above equality and noting that $\alpha' = 1$, we get $2tt' = d\sin\alpha$. Applying the sine law to $\triangle O_1O_2O_3$ gives $2t\sin\beta = d\sin\alpha$, because α and β have the same sign, by definition (see Figure 5.5). It follows that

$$t' = \sin\beta. \tag{5.3}$$

We first consider the case when $\Pi(M)$ is of type $C_1^+C_2^-SC_3^-$. Recall that $\psi(X)$ is the polar angle of the tangent vector for configuration X. This angle is constant for all configurations along an S-segment. On the other hand, the angle increases by δ after traversing a C^+-segment of length δ and decreases by the same amount upon traversing a C^--segment of the same length. We therefore have the following:

$$\psi(Y) \equiv \psi(X) + \alpha_1 - \alpha_2 - \alpha_3 \pmod{2\pi}. \tag{5.4}$$

Since X and Y are fixed, we have $1 - \alpha_2' - \alpha_3' = 0$ or $\alpha_2' + \alpha_3' = 1$. Substituting this into (5.2) gives

$$L' = 2 + 2s'.$$

The S-segment is a translation of the segment O_2O_3 (see Figure 5.5b). Thus $s = t$, and hence $s' = t' = \sin\beta$ by (5.3). Thus $L' = 2 + 2\sin\beta$. Since $\Pi(M)$ is of type $C^+C^-SC^-$,

$$\beta + \pi/2 + \alpha_2 \equiv 0 \pmod{2\pi}$$

(see Figure 5.5b); indeed, $\beta = \angle O_1O_2O_3$, $\pi/2 = \angle O_3O_2P$, and $\alpha_2 = \angle PO_2O_1$. Thus $\beta \equiv 3\pi/2 \pmod{2\pi}$ if and only if $\alpha_2 \equiv 0 \pmod{2\pi}$, which occurs only at a singular point (by definition of singular points). Therefore, $L'(\alpha_1) > 0$ for any nonsingular value of α_1 for which the path $\Pi(\alpha_1)$ is defined.

We now turn to the case in which $\Pi(M)$ is of type $C_1^+C_2^-SC_3^+$. Then (5.4) is replaced by

$$\psi(Y) \equiv \psi(X) + \alpha_1 - \alpha_2 + \alpha_3 \pmod{2\pi},$$

so $\alpha_3' = \alpha_2' - 1$. Substituting this into (5.2) gives

$$L' = 2(s' + \alpha_2').$$

In order to find an expression for α_2', it is convenient to define the oriented angle $\gamma = \angle O_3O_2P$, where P is the common point between the segments C_2 and S (see Figure 5.5a). Recall that $\alpha_2 = \angle PO_2O_1$ and $\beta = \angle O_1O_2O_3$. Thus $\gamma + \alpha_2 + \beta \equiv 0 \pmod{2\pi}$, which implies that $\gamma' + \alpha_2' + \beta' = 0$ and

$$L' = 2(s' - \gamma' - \beta'). \tag{5.5}$$

We now find expressions for $s' - \gamma'$ and β'.

With P and Q denoting, respectively, the first and last points of the S-segment, it is easy to see that the segments O_2O_3, PQ, O_2P, and O_3Q form two congruent right triangles (see Figure 5.5a). Thus we have $s^2 + 1 = t^2$, whence $ss' = tt' = t\sin\beta$, using (5.3). Further, $\tan\gamma = s$, so $\gamma' = s'\cos^2\gamma$. Combining the two results,

$$s' - \gamma' = s'\sin^2\gamma = s'\left(\frac{s}{t}\right)^2 = \frac{s}{t}\sin\beta. \tag{5.6}$$

The final derivative needed is β', which again follows from the cosine law applied to triangle $\triangle O_1O_2O_3$ (see Figure 5.5a):

$$d^2 = 4 + 4t^2 - 8t\cos\beta.$$

After a differentiation and rearrangement, this yields $t\beta'\sin\beta = t'(\cos\beta - t)$. Substituting for t' using (5.3) and noting that $\alpha \not\equiv 0 \pmod{\pi}$ implies $\sin\beta \neq 0$, we obtain

$$\beta' = \frac{1}{t}(\cos\beta - t). \tag{5.7}$$

Combining (5.5), (5.6), and (5.7) yields

$$L' = \frac{2}{t}(s\sin\beta + t - \cos\beta).$$

$\Pi(\alpha_1)$ is defined only when $\mathcal{C}_2$ and $\mathcal{C}_3$ do not properly intersect. Thus $t \geqslant 1$ wherever L is defined, thereby implying that $(t - \cos\beta) \geqslant 0$ and that

$$\begin{aligned} L' \leqslant 0 &\Leftrightarrow t - \cos\beta \leqslant -s\sin\beta \\ &\Rightarrow t^2 - 2t\cos\beta + \cos^2\beta \leqslant s^2\sin^2\beta = (t^2 - 1)\sin^2\beta \\ &\Rightarrow t^2\cos^2\beta - 2t\cos\beta + 1 \leqslant 0 \\ &\Rightarrow (t\cos\beta - 1)^2 \leqslant 0 \\ &\Rightarrow \cos\beta = \frac{2}{2t}. \end{aligned}$$

Hence $L' \leqslant 0$ only if $\angle O_3O_1O_2 = \alpha \equiv \pi/2 \pmod{\pi}$ (see Figure 5.5a). Therefore, $L' > 0$ almost everywhere it is defined.

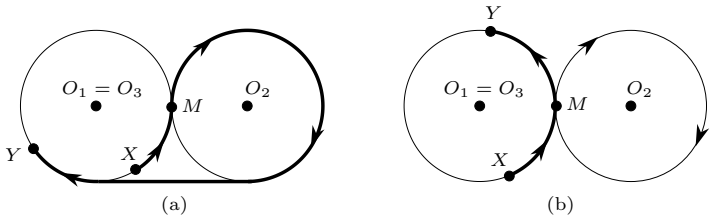


FIG. 5.6. *When $O_1 = O_3$: (a) $\Pi(M)$ is of type $C_1^+C_2^-SC_3^-$; (b) $\Pi(M)$ of type $C_1^+C_2^-SC_3^+$ degenerates to a single C-segment for any $M \in \mathcal{C}_1$.*

Case 2. *O_1 and O_3 are equal.* When $\Pi(M)$ is of type $C_1^+C_2^-SC_3^-$ (see Figure 5.6a), $\alpha_2 = 3\pi/2$ and $s = 2$; thus $L' = 1 + \alpha_3'$. Equation (5.4) still holds, and thus $\alpha_3' = 1$. Therefore, $L' = 2 > 0$ everywhere it is defined.

When $\Pi(M)$ is of type $C_1^+C_2^-SC_3^+$ (see Figure 5.6b), the circles $\mathcal{C}_1$ and $\mathcal{C}_3$ coincide and have the same orientation. Thus both segments C_2 and S vanish, $\Pi(M)$ degenerates to a C-segment, and, consequently, $L' = 0$ everywhere except when $M = X$ or Y, where L is not differentiable. ∎

5.3. Computing type (A.ii) paths. As mentioned in section 4, we can compute in $O(n \log n)$ time the feasible candidates of type (A.ii) paths with at most one $\mathcal{PC}$-anchored segment. If the path is of type $C_I\bar{\bar{C}}S\bar{\bar{C}}C_F$, a naive analysis gives $O(n^2)$ candidates to check. Using Lemma 5.5, we reduce the number of candidates to $O(n)$ and compute them in $O(n \log n)$ time, as follows.

Fix the orientations of the terminal C-segments, and let $\mathcal{C}_I$ and $\mathcal{C}_F$ denote the circles supporting C_I and C_F, respectively. Let us assume that $\mathcal{C}_I$ is oriented counterclockwise. Let $\mathcal{K}_I$ be the sequence of $\mathcal{PC}$-anchored circles that touch $\mathcal{C}_I$ and that are free, sorted by their points of tangency with $\mathcal{C}_I$. The set $\mathcal{K}_F$ is defined analogously for $\mathcal{PC}$-anchored circles tangent to $\mathcal{C}_F$. The sets $\mathcal{K}_I$ and $\mathcal{K}_F$ can be computed in $O(n \log n)$ time, and they have $O(n)$ elements.

By Lemma 3.6, the circles $\bar{\bar{\mathcal{C}}}_1$ and $\bar{\bar{\mathcal{C}}}_2$ supporting the $\bar{\bar{C}}$-segments in an optimal path Π of type $C_I\bar{\bar{C}}_1S\bar{\bar{C}}_2C_F$ are free. (Otherwise, $\bar{\bar{C}}_1$ or $\bar{\bar{C}}_2$ would be a terminal C-segment.) Therefore, the $\bar{\bar{C}}_1$-segment of Π lies on a circle of $\mathcal{K}_I$, and the $\bar{\bar{C}}_2$-segment lies on a circle of $\mathcal{K}_F$. Suppose $\mathcal{C}_2 \in \mathcal{K}_F$ supports the $\bar{\bar{C}}_2$-segment of Π. This fixes the terminal configuration of the subpath $C_I\bar{\bar{C}}_1S\bar{\bar{C}}_2$. By Lemma 5.4, there are at most six singular points, say, $\Sigma_0 = I, \ldots, \Sigma_5$, on $\mathcal{C}_I$ with respect to $\mathcal{C}_2$, sorted in the counterclockwise sense.

For $0 \leqslant i \leqslant 5$, let $\mathcal{K}_I(i) \subseteq \mathcal{K}_I$ be the contiguous subsequence of circles that touch $\mathcal{C}_I$ at a point in the arc $\mathcal{C}_I[\Sigma_i, \Sigma_{i+1}]$. By Lemma 5.5, only the first circle of $\mathcal{K}_I(i)$ is a candidate for $\mathcal{C}_1$. Hence at most six circles in $\mathcal{K}_I$ are candidates for $\mathcal{C}_1$. For each $0 \leqslant i \leqslant 5$, by performing a binary search on $\mathcal{K}_I$, we can find, in $O(\log n)$ time, the first circle of $\mathcal{K}_I$ whose point of tangency with $\mathcal{C}_I$ lies after Σ_i in the counterclockwise sense. Obviously, this is the first circle of $\mathcal{K}_I(i)$. We can therefore compute in $O(\log n)$ time at most six candidate paths for a fixed $\mathcal{C}_2 \in \mathcal{K}_F$. By examining each $\mathcal{C}_2 \in \mathcal{K}_F$ in turn, we compute $O(n)$ candidate paths in $O(n \log n)$ time. We repeat a similar procedure for all possible orientations of C_I and C_F. We can therefore conclude the following.

LEMMA 5.6. *An optimal path of type* (A.ii) *can be computed in $O(n \log n)$ time.*

5.4. Computing type (B.iii) paths. Let Π be an optimal path of the form $\Pi_I\bar{C}_2\bar{C}_3\Pi_F$, i.e., of type (B.iii). Suppose we know the edges e, e' that are tangent to $\bar{C}_2$ and $\bar{C}_3$, respectively.

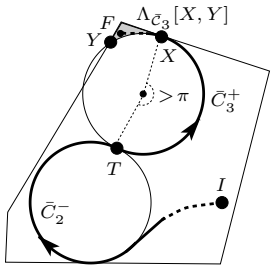


FIG. 5.7. *Illustration of the proof of Lemma* 5.7.

If Π does not contain any $\bar{\bar{C}}$-segment in Π_I or Π_F, then Π is of type $C_I S \bar{C}_2 \bar{C}_3 S C_F$. We can compute Π in $O(\log n)$ time using Lemmas 4.1 and 4.2.

Next consider the case in which Π_I and Π_F each contain a $\bar{\bar{C}}$-segment, i.e., Π is of type $C_I \bar{\bar{C}} S \bar{C}_2 \bar{C}_3 S \bar{\bar{C}} C_F$. We show that, given e and e', we can compute, in $O(\log n)$ time, a set of $O(1)$ candidate circles that contains the $\bar{\bar{C}}$-segments of Π. Given this set, we can compute in $O(\log n)$ time the shortest feasible path of the above type, using Lemmas 4.1 and 4.2. Thus, by considering all $\Theta(n^2)$ pairs of edges of $\mathcal{P}$, we can compute in $O(n^2 \log n)$ time a set of $O(n^2)$ candidate paths for this case. However, we will see later (in Lemma 5.15) that it suffices to consider fewer pairs of edges of $\mathcal{P}$ in some cases.

5.4.1. Properties of paths. We first establish some simple properties of an optimal path Π of type $C_I \bar{\bar{C}}_1 S \bar{C}_2 \bar{C}_3 S \bar{\bar{C}}_4 C_F$. Assume without loss of generality that $\bar{C}_2$ and $\bar{C}_3$ are oriented clockwise and counterclockwise, respectively. By Lemma 3.3, the $\bar{\bar{C}}_1$-segment is oriented clockwise, and the $\bar{\bar{C}}_4$-segment is oriented counterclockwise; i.e., Π is of type $C_I^+ \bar{\bar{C}}_1^- S \bar{C}_2^- \bar{C}_3^+ S \bar{\bar{C}}_4^+ C_F^-$. Let $\bar{\bar{\mathcal{C}}}_1$, $\bar{\mathcal{C}}_2$, $\bar{\mathcal{C}}_3$, and $\bar{\bar{\mathcal{C}}}_4$ denote the circles supporting the C-segments $\bar{\bar{C}}_1$, $\bar{C}_2$, $\bar{C}_3$, and $\bar{\bar{C}}_4$, respectively.

LEMMA 5.7. *If an optimal path is of type $C_I \bar{\bar{C}}_1 S \bar{C}_2 \bar{C}_3 S \bar{\bar{C}}_4 C_F$, then the circles $\bar{\bar{\mathcal{C}}}_1$, $\bar{\mathcal{C}}_2$, $\bar{\mathcal{C}}_3$, and $\bar{\bar{\mathcal{C}}}_4$ are free.*

Proof. Lemma 3.6 directly yields that $\bar{\bar{\mathcal{C}}}_1$ and $\bar{\bar{\mathcal{C}}}_4$ are free. Suppose now, for a contradiction, that $\bar{\mathcal{C}}_3$ is not free. As before, we assume that the orientations are such that $\Pi = C_I^+ \bar{\bar{C}}_1^- S \bar{C}_2^- \bar{C}_3^+ S \bar{\bar{C}}_4^+ C_F^-$. Let T be the point of tangency between $\bar{C}_2$ and $\bar{C}_3$. Moving along $\bar{C}_3^+$, let X be the last point of tangency between $\bar{C}_3$ and $\partial\mathcal{P}$ (see Figure 5.7). Starting at X and moving along $\bar{\mathcal{C}}_3^+$, let Y be the first proper intersection point between $\bar{\mathcal{C}}_3$ and $\partial\mathcal{P}$.

By Lemma 2.4, the length of $\bar{C}_3$ between T and X is greater than π; i.e., $\|\bar{C}_3^+[T,X]\| > \pi$. It follows that $\bar{\mathcal{C}}_3$, X, and Y define a pocket $\Lambda_{\bar{\mathcal{C}}_3}[X,Y]$ (see Figure 5.7). By Lemma 2.7, this pocket contains $\Pi[X,F]$ and therefore contains the C-segment $\bar{\bar{C}}_4$. However, the pocket does not contain the circle $\bar{\bar{\mathcal{C}}}_4$, by Lemma 2.6. The path $\bar{C}_3 S \bar{\bar{C}}_4$ enters the pocket at X, and, since $\bar{\bar{\mathcal{C}}}_4$ is free, it is possible to escape the pocket by extending segment $\bar{\bar{C}}_4$. This contradicts Lemma 2.7, establishing that $\bar{\mathcal{C}}_3$ is free. A symmetric argument shows that circle $\bar{\mathcal{C}}_2$ is free. □

We now introduce the following simple definitions. Given a circle $\mathcal{C}$, a point $M \in \mathcal{C}$ is called *free on* $\mathcal{C}$ if and only if the circle tangent to $\mathcal{C}$ at M is free. Given a circle $\mathcal{C}$ and a point $X \in \mathcal{C}$, a point $M \in \mathcal{C}$ is called the *first free point after* X *along* $\mathcal{C}^+$ if and only if M is free on $\mathcal{C}$ and no point on the arc $\mathcal{C}^+[X,M)$ is free. In Figure 5.8, M^* is the first free point after M_L along $\mathcal{C}_I^+$. Note that M and X might coincide. The circle tangent to $\mathcal{C}$ at the first free point after X is called the *first free circle after* X *along* $\mathcal{C}^+$.

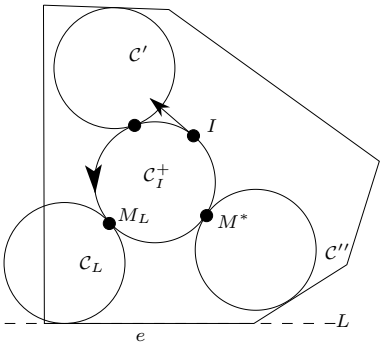


FIG. 5.8. *Definition of $\mathcal{C}'$ and $\mathcal{C}''$.*

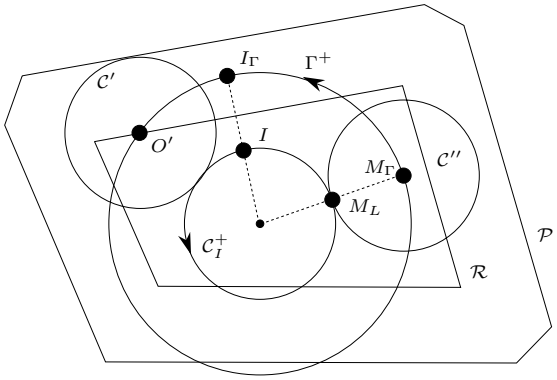


FIG. 5.9. *Illustration of the proof of Lemma* 5.8.

We now show that, given I, F, e, and e', we can compute in $O(\log n)$ time a set of $O(1)$ candidate circles that contain the $\bar{\bar{C}}$-segments of any optimal path from I to F of type $C_I^+\bar{\bar{C}}_1^-S\bar{C}_2^-\bar{C}_3^+S\bar{\bar{C}}_4^+C_F^-$ such that $\bar{C}_2^-$ and $\bar{C}_3^+$ are tangent to e and e', respectively. We show how to compute candidate circles for $\bar{\bar{\mathcal{C}}}_1$; computing candidate circles for $\bar{\bar{\mathcal{C}}}_4$ is similar.

5.4.2. Computing $\bar{\bar{\mathcal{C}}}_1$. We identify two circles $\mathcal{C}'$ and $\mathcal{C}''$ that are the candidate circles for $\bar{\bar{\mathcal{C}}}_1$. See Figure 5.8. $\mathcal{C}'$ is the first free circle after I along $\mathcal{C}_I^+$. Such a circle $\mathcal{C}'$ exists, by Lemma 5.7, if the optimal path from I to F is of type $C_I^+\bar{\bar{C}}_1^-S\bar{C}_2^-\bar{C}_3^+S\bar{\bar{C}}_4^+C_F^-$. For simplicity, we define $\mathcal{C}''$ in the local frame where the line L through e is horizontal and below $\mathcal{P}$. If the distance between L and $\mathcal{C}_I^+$ is greater than 2, then $\mathcal{C}''$ is not defined. Otherwise, there exist two circles that are above L and tangent to both $\mathcal{C}_I^+$ and L. Let $\mathcal{C}_L$ be the leftmost of these two circles, and let M_L be its point of tangency with $\mathcal{C}_I^+$. Let $\mathcal{C}''$ be the first free circle after M_L along $\mathcal{C}_I^+$. Note that $\mathcal{C}'$ depends only on I and $\mathcal{P}$ and that $\mathcal{C}''$ depends only on I, e, and $\mathcal{P}$.

LEMMA 5.8. *After preprocessing $\mathcal{P}$ in $O(n\log n)$ time, for a given edge e, $\mathcal{C}'$ and $\mathcal{C}''$ can be computed in $O(\log n)$ time.*

Proof. Let Γ be the circle of radius 2 concentric with $\mathcal{C}_I^+$ (see Figure 5.9). Let $\mathcal{R}$ be the *retracted polygon* of $\mathcal{P}$ with respect to a unit circle, i.e., $\mathcal{R}$ is the set of points p such that the unit circle centered at p lies inside $\mathcal{P}$; $\mathcal{R}$ is a convex polygonal region with at most n edges, and it can be computed in linear time. We preprocess $\mathcal{R}$ in $O(n\log n)$ time using the algorithm by Cheng et al. [10] so that, given a unit-radius

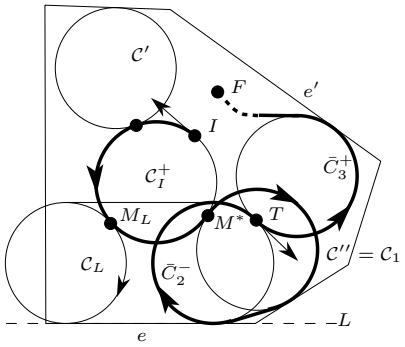


FIG. 5.10. *Circles $\mathcal{C}'$ and $\mathcal{C}''$ with a (supposedly) optimal path* Π.

circle $\mathcal{C}$ and a point $M \in \mathcal{C}$, we can compute in $O(\log n)$ time the first intersection point of $\mathcal{C}$ and $\mathcal{R}$ as we walk along $\mathcal{C}$ in the clockwise (or counterclockwise) direction.

Let I_Γ (respectively, M_Γ) be the intersection point between Γ and the ray emanating from the center of $\mathcal{C}_I^+$ and going through I (respectively, M_L), and let O' be the first intersection point between Γ and $\mathcal{R}$ starting at I_Γ and moving along Γ^+. Note that $O' = I_\Gamma$ if I_Γ lies inside $\mathcal{R}$. The center of $\mathcal{C}'$ is O'. Indeed, by definition of $\mathcal{R}$, the circle centered at O' is free, and any circle (of unit radius) centered at a point on $\Gamma^+[I_\Gamma, O')$ is not free. Since $\mathcal{C}'$ does not depend on the edge e, we can compute it in $O(n)$ time in the preprocessing stage once and for all. The center of $\mathcal{C}''$ is the first intersection point between Γ and $\mathcal{R}$ starting at M_Γ and moving along Γ^+, and it can be computed in $O(\log n)$ time. □

We now state a key lemma, which we prove in the following section.

LEMMA 5.9. *If Π is an optimal path of type $C_I^+ \bar{C}_1^- S \bar{C}_2^- \bar{C}_3^+ S \bar{C}_4^+ C_F^-$, then $\bar{C}_1$ is supported by $\mathcal{C}'$ or $\mathcal{C}''$.*

5.4.3. Proof of Lemma 5.9. Let T be the configuration on Π at the common point between $\bar{C}_2$ and $\bar{C}_3$. See Figure 5.10. As before (in section 5.2), any choice of a point $M \in \mathcal{C}_I^+$ defines one path $\Pi(M)$ of the form $C_I^+ C_1^- S C_2^-$, which begins at I and ends at T, and where $\mathcal{C}_I^+$ and $\mathcal{C}_1^-$ are tangent at M. Since the C-segments C_1^- and C_2^- have the same orientation, the path $\Pi(M)$ always exists, though it might not be free. Let $M^* \in \mathcal{C}_I^+$ be the point such that $\Pi(M^*)$ is a subpath of the optimal path Π. It follows that $\Pi(M^*)$ is an optimal path from I to T. We will show that M^* is the first free point after I or M_L.

We consider different cases based on which of the singular points exist on $\mathcal{C}_I^+$. See Figure 5.11. We first introduce some notation in order to distinguish different singular points. Let $M_1 \in \mathcal{C}_I$ be the point such that the C-segment C_1^- in $\Pi(M)$ vanishes (i.e., $\Pi(M)$ is of type $C_I^+ S C_2^-$); M_1 is only defined when $\mathcal{C}_I$ and $\mathcal{C}_2$ do not properly intersect. Assume, for simplicity, that T is the lowest point of $\mathcal{C}_2^-$, and let L_0 be the horizontal half-line lying to the right of T. Let $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_1'$ be the two circles (if they exist) tangent to $\mathcal{C}_I$ with center on the horizontal line through the center of $\mathcal{C}_2^-$, and let M_2 and M_2' be their respective common points with $\mathcal{C}_I$; assume without loss of generality that M_2 is left of M_2'. The point M_2 (respectively, M_2') is a singular point of $\Pi(M)$ (at which C_2^- vanishes) if and only if $\tilde{\mathcal{C}}_1$ (respectively, $\tilde{\mathcal{C}}_1'$) touches L_0. Otherwise, the C_2-segment of $\Pi(M_2)$ (respectively, $\Pi(M_2')$) has length π (see Figure 5.11b).

Since C_1- and C_2-segments of $\Pi(M)$ have the same orientation, the S-segment vanishes if and only if the path type $C_I^+ C_1^- S C_2^-$ degenerates to $C_I^+ C^-$. Thus, if the

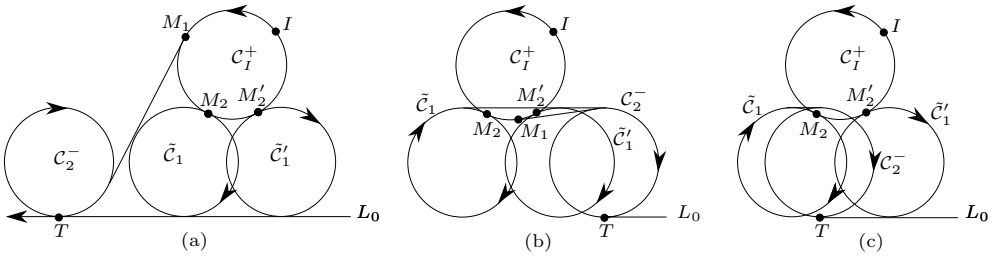


FIG. 5.11. *Singular points of* $\Pi(M)$. *In* (a), $\{I, M_1, M_2, M_2'\}$. *In* (b), $\{I, M_1\}$. *In* (c), $\{I, M_2'\}$.

S-segment vanishes, the C_1- or C_2-segment also vanishes. Therefore, in view of the discussion in section 5.2, only the following points can be singular points:

- I (C_I^+ vanishes),
- M_1, if $\mathcal{C}_I$ and $\mathcal{C}_2$ do not properly intersect (C_1^- vanishes),
- M_2, if $\tilde{\mathcal{C}}_1$ exists and touches L_0 (C_2^- vanishes), and
- M_2', if $\tilde{\mathcal{C}}_1'$ exists and touches L_0 (C_2^- vanishes).

There are three cases to consider, depending on the relative positions of $\mathcal{C}_1$ and $\mathcal{C}_2$.

(i) The distance between $\mathcal{C}_I$ and the line supporting L_0 is at most 2, and $\mathcal{C}_2$ lies to the left of $\tilde{\mathcal{C}}_1$; i.e., both $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_1'$ touch L_0; see Figure 5.11a. In this case, $\mathcal{C}_2$ does not intersect $\mathcal{C}_I$, and therefore M_1 also exists. The singular points are thus $\{I, M_1, M_2, M_2'\}$.

(ii) Either the distance between $\mathcal{C}_I$ and the line supporting L_0 is greater than 2, or neither $\tilde{\mathcal{C}}_1$ nor $\tilde{\mathcal{C}}_1'$ touches L_0; see Figure 5.11b. In both cases, $\mathcal{C}_I$ and $\mathcal{C}_2$ do not intersect, so M_1 exists. The singular points are therefore $\{I, M_1\}$.

(iii) The distance between $\mathcal{C}_I$ and the line supporting L_0 is at most 2, and $\mathcal{C}_2$ lies between $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_1'$. In this case, $\tilde{\mathcal{C}}_1$ does not touch L_0, and $\mathcal{C}_I$ intersects $\mathcal{C}_2$, so the singular points are $\{I, M_2'\}$; see Figure 5.11c.

Before proving for each case that M^* is the first free point along $\mathcal{C}_I^+$ after I or M_L, we state a few claims, which we will need for the proof.

CLAIM 5.10. *M^* is not a singular point of* $\Pi(M)$.

Proof. If M^* is a singular point, the type of $\Pi(M^*)$ degenerates, contradicting that Π is of type $C_I^+ \bar{\bar{C}}_1^- S \bar{C}_2^- \bar{C}_3^+ S \bar{\bar{C}}_4^+ C_F^-$. □

CLAIM 5.11. *M^* is the first free point along $\mathcal{C}_I^+$ after a singular point of* $\Pi(M)$.

Proof. By Lemma 5.7, M^* is free on $\mathcal{C}_I^+$. If there exists $M' \neq M^*$ free on $\mathcal{C}_I^+$ such that $\mathcal{C}_I^+(M', M^*]$ does not contain any singular point, then the path $\Pi(M')$ exists because C_1^- and C_2^- have the same orientation. $\Pi(M')$ is free because the first C_I^+-segment of $\Pi(M')$ is part of the feasible path $\Pi(M^*)$, the circle $\mathcal{C}_1^-$ is free by definition of M', and the circle $\mathcal{C}_2^-$ is free (by Lemma 5.7). Finally, $\|\Pi(M')\| < \|\Pi(M^*)\|$ by the monotonicity property (Lemma 5.5), which contradicts the optimality of $\Pi(M^*)$. □

CLAIM 5.12. *If M_2 and M_2' exist, then* (i) *the length function $\|\Pi(M)\|$ increases at $M = M_2$, and* (ii) *$M^* \in \mathcal{C}_I^+(M_2, M_2')$.*

Proof. If M_2 is not a singular point, then, by Lemma 5.5, $\|\Pi(M)\|$ increases at M_2. If M_2 is a singular point, then $\|\Pi(M)\|$ jumps by 2π at M_2 (see Figure 5.11a). As for (ii), the length of the last C-segment C_2 of $\Pi(M)$ is greater than π if and only if the center of $\mathcal{C}_1$ lies below the center of $\mathcal{C}_2$ (see Figure 5.11); that is, $M \in \mathcal{C}_I^+(M_2, M_2')$. Since C_2 is a nonterminal C-segment of the optimal path Π, $\|C_2\| > \pi$, and therefore $M^* \in \mathcal{C}_I^+(M_2, M_2')$. □

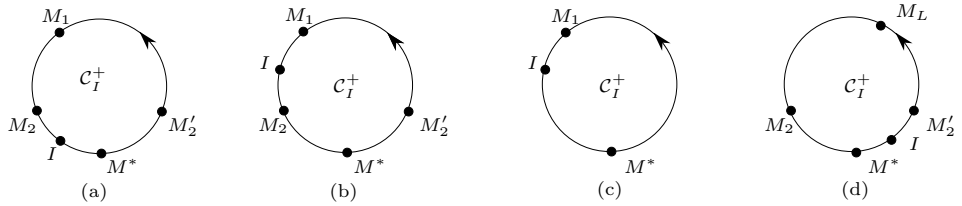


FIG. 5.12. *Some positions of the singular points* $\{I, M_1, M_2, M_2'\}$ *in* (a) *and* (b), $\{I, M_1\}$ *in* (c), *and* $\{I, M_2'\}$ *in* (d).

CLAIM 5.13. *If* M_1 *exists, then the circular arc* $\mathcal{C}_I^+[M_1, M^*]$ *contains* I *or* M_2'. *If* $I \notin \mathcal{C}_I^+[M_1, M^*]$, *then* M_2' *is a singular point.*

Proof. If $I \notin \mathcal{C}_I^+[M_1, M^*]$, then $\Pi(M_1)$ is free because the first C-segment $\mathcal{C}_I^+[I, M_1]$ of $\Pi(M_1)$ is part of $\Pi(M^*)$, and $\mathcal{C}_2^-$ is free by Lemma 5.7. Thus $\Pi(M_1, M^*]$ contains a singular point, because otherwise $\|\Pi(M_1)\| < \|\Pi(M^*)\|$ by the monotonicity property (Lemma 5.5) and Claim 5.10, and thus $\Pi(M^*)$ is not optimal, which is a contradiction. If M_2' also does not lie in $\mathcal{C}_I^+[M_1, M^*]$, then M_2 is a singular point and lies on this arc. By Claim 5.12, $\|\Pi(M_1)\| < \|\Pi(M_2)\| < \|\Pi(M^*)\|$, which is a contradiction. Hence either I or M_2' lies on $\mathcal{C}_I^+[M_1, M^*]$, and M_2' is a singular point if I does not lie in this arc. ∎

We now prove for each of the three cases stated above that M^* is the first free point after I or M_L.

Case (i). The singular points are $\{I, M_1, M_2, M_2'\}$. Since $\mathcal{C}_2$ lies to the left of $\tilde{\mathcal{C}}_1$, one can easily show that $M_1 \in \mathcal{C}_I^+[M_2', M_2]$ (see Figure 5.11a). Refer now to Figures 5.12a and b. By Claim 5.12, $M^* \in \mathcal{C}_I^+(M_2, M_2')$. It follows that $\mathcal{C}_I^+[M_1, M^*]$ does not contain M_2' and thus contains I (by Claim 5.13). Thus $\mathcal{C}_I^+(I, M^*]$ does not contain any singular point except possibly M_2. If $M_2 \notin \mathcal{C}_I^+(I, M^*]$ (Figure 5.12a), then M^* is the first free point after I because, by Claim 5.11, M^* is the first free point after a singular point. Even if $M_2 \in \mathcal{C}_I^+(I, M^*]$ (Figure 5.12b), M^* is the first free point after I. Indeed, if the first free point after I along $\mathcal{C}_I^+$ is $M' \in \mathcal{C}_I^+[I, M_2)$, then $\Pi(M')$ is free because the first arc $\mathcal{C}_I^+[I, M']$ of $\Pi(M')$ is part of Π, and the second and third C-segments of $\Pi(M')$ are free by definition of M' and by Lemma 5.7, respectively. Moreover, by Lemma 5.5 and Claim 5.12, $\|\Pi(M')\| < \|\Pi(M_2)\| < \|\Pi(M^*)\|$, which is a contradiction.

Case (ii). The singular points are $\{I, M_1\}$. See Figure 5.12c. Since M_2' is not a singular point, by Claim 5.13, $I \in \mathcal{C}_I^+[M_1, M^*]$. Consequently, $\mathcal{C}_I^+(I, M^*]$ does not contain any singular point. Therefore, by Claim 5.11, M^* is the first free point after I.

Case (iii). The singular points are $\{I, M_2'\}$. As before, if $\mathcal{C}_I^+(I, M^*]$ does not contain any singular point, M^* is the first free point after I. We thus consider the case in which $M_2' \in \mathcal{C}_I^+(I, M^*]$ (see Figure 5.12d). Since M_2 and M_2' exist, by Claim 5.12, $M^* \in \mathcal{C}_I^+(M_2, M_2')$. It thus follows that M^* is the first free point after M_2' (by Claims 5.11 and 5.12). Thus, in order to show that M^* is the first free point after M_L, it is sufficient to prove that $M_L \in \mathcal{C}_I^+[M_2', M_2]$, which is equivalent to proving that the length of the last C-segment C_2 of $\Pi(M_L)$ is at most π. This can be shown as follows.

We assume, for simplicity, that the edge e (tangent to the C-segment C_2^-) is horizontal and below $\mathcal{P}$ (see Figure 5.10); to be consistent, we no longer assume that T is the lowest point of $\mathcal{C}_2^-$. By Lemma 2.4, the arc length of $\bar{C}_2$ in Π from its point of tangency with the edge e to T must be at least π. In other words, T must be in the right half of $\bar{\mathcal{C}}_2$. On the other hand, by definition of M_L, $\mathcal{C}_L$ is the leftmost circle

of all of the unit circles tangent to L from above that intersect $\mathcal{C}_I$. Since $\mathcal{C}_2$ is tangent to L from above and properly intersects $\mathcal{C}_I$ (because M_1 is not defined in this case), the top point of $\mathcal{C}_L$ is left of the top point of $\mathcal{C}_2$. Thus, since T is on the right half of $\bar{\mathcal{C}}_2$, the arc length of C_2 in $\Pi(M_L)$ is less than π (see Figure 5.10).

5.4.4. Computing the overall path. By Lemmas 5.8 and 5.9, we can compute, in $O(\log n)$ time, two candidates for the circle supporting segment $\bar{\bar{C}}_1$. We can similarly compute two candidates for the circle supporting segment $\bar{\bar{C}}_4$. By Lemma 4.2, this gives us $O(1)$ candidate paths, for which we may check the feasibility in $O(\log n)$ time. Hence, given two edges e and e' of $\mathcal{P}$, we can compute in $O(\log n)$ time an optimal path of type $C_I\bar{\bar{C}}_1S\bar{C}_2\bar{C}_3S\bar{\bar{C}}_4C_F$, where $\bar{C}_2$ and $\bar{C}_3$ are tangent to e and e', respectively.

If the optimal path is of type (B.iii) with only one $\bar{\bar{C}}$-segment in Π_I or Π_F, we get similar results. For example, if an optimal path is of type $C_I\bar{\bar{C}}_1S\bar{C}_2\bar{C}_3SC_F$, then $\bar{\mathcal{C}}_1$ and $\bar{\mathcal{C}}_2$ are free, and $\bar{\bar{C}}_1$ is supported by $\mathcal{C}'$ or $\mathcal{C}''$ as defined above. Thus we obtain the following lemma.

LEMMA 5.14. *Let e, e' be edges of $\mathcal{P}$. In $O(\log n)$ time, we can compute an optimal path of type $\Pi_I\bar{C}_2\bar{C}_3\Pi_F$, where $\Pi_I \in \{C_I\bar{\bar{C}}S, C_IS, C_I, S\}$, $\Pi_F \in \{S\bar{\bar{C}}C_F, SC_F, C_F, S\}$, and $\bar{C}_2$ and $\bar{C}_3$ are tangent to e and e', respectively.*

5.4.5. Finding candidate pairs of edges. Now we describe how to find a suitable set of pairs of edges $\mathcal{E}$ such that, if an optimal path from I to F is of type (B.iii) (i.e., $\Pi_I\bar{C}_2\bar{C}_3\Pi_F$), then the pair of edges (e, e') tangent to $\bar{C}_2$ and $\bar{C}_3$ is in the set $\mathcal{E}$.

Agarwal, Raghavan, and Tamaki [1] showed that, if an optimal path from I to F is of type $\Pi_I\bar{C}_2\bar{C}_3\Pi_F$ such that $\bar{C}_2$ and $\bar{C}_3$ are nonterminal, then $\mathcal{C}_I$ intersects $\bar{\mathcal{C}}_3$ (the circle supporting $\bar{C}_3$), and $\mathcal{C}_F$ intersects $\bar{\mathcal{C}}_2$ (the circle supporting $\bar{C}_2$). Thus the center of $\bar{\mathcal{C}}_3$, which is at most distance 1 from the boundary of the polygon, is at most at distance 3 from I. Since the centers of $\bar{\mathcal{C}}_2$ and $\bar{\mathcal{C}}_3$ are distance 2 apart, they are each at distance less than 5 from I. Thus edges e and e' are at distance less than 6 from I. By symmetry, they are also at distance less than 6 from F. Therefore, we can consider $\mathcal{E}$ to be the set of pairs of edges of $\mathcal{P}$ that are at distance less than 6 from both I and F. Let k denote the number of edges of $\mathcal{P}$ at distance less than 6 from both I and F. Then $|\mathcal{E}| = k^2$, and $\mathcal{E}$ can be computed in $O(k^2)$ time. Lemma 5.14 then gives the following.

LEMMA 5.15. *An optimal path of type* (B.iii) *can be computed in $O(k^2 \log n)$ time.*

Putting everything together, we obtain the following.

THEOREM 5.16. *Given a convex polygon $\mathcal{P}$, an initial configuration I, and a final configuration F, an optimal path from I to F inside $\mathcal{P}$ can be computed in time $O((n + k^2)\log n)$, where k is the number of edges of $\mathcal{P}$ at distance less than* 6 *from both I and F.*

Proof. We have shown in section 4 and in Lemmas 5.2, 5.6, and 5.15 that the Dubins paths and the optimal paths of types (A.i), (A.ii), (B.i), and (B.ii) can be computed in $O(n \log n)$ time, while paths of type (B.iii) can be computed in $O(k^2 \log n)$ time. Choosing the shortest among all of those paths yields an optimal path. ∎

6. Conclusion. For an obstacle-free environment, Dubins' classification of optimal path types yields a constant time algorithm for computing optimal paths [15]. On the other hand, in the presence of general polygonal obstacles, the optimal path-planning problem is NP-hard [29]. In this paper, we have studied a very simple

environment, the inside of a convex polygon. We have given a classification of optimal path types and an $O(n^2 \log n)$ algorithm for optimal path planning. We have found that, surprisingly, the number of straight or circular segments composing optimal paths is bounded by a constant, independent of the number n of sides of the convex polygon.

The running time of the algorithm and the constant bound on the number of segments in an optimal path lead us to speculate that other simple environments may also be amenable to polynomial time algorithmic solution. However, we caution that, although the environment we have considered is simple, our algorithm results from considerable technical analysis.

Our techniques and results may provide useful tools for further study of these problems. In particular, we call attention to two properties of moderate paths that we believe are interesting and possibly useful in their own right:

(i) A feasible path entering the interior of a pocket can never escape the pocket (Lemma 2.7).

(ii) The length of a path of type $CCSC$ from X to Y is a piecewise strictly increasing function of the length of the first C-segment (Lemma 5.5).

Note that property (ii) holds regardless of the environment.

The theory of curvature-constrained path planning is relatively unexplored, so many problems remain. We mention some specific open problems arising from our work and then conclude with a more general one.

First, we ask whether our classification of optimal path types inside a convex polygon is tight; that is, does each possible type of optimal path given in our classification actually arise for some $\mathcal{P}$, I, and F? We believe that the answer is yes, although we have no formal proof of this (see Remark 3.2 for details). Also, we ask whether there is a polygon $\mathcal{P}$ in which all types arise.

Next, in our optimal path-planning algorithm, the most time-consuming part lies in the computation of the optimal paths of type (B.iii) (see Theorem 3.1). Indeed, if we eliminate type (B.iii) from consideration, the complexity of our algorithm drops to $O(n \log n)$. Furthermore, paths of type (B.iii) are rather complex and thus may be difficult to track by a mobile robot. This situation suggests two lines of investigation. First, the paths of type (B.iii) should be studied thoroughly in order to understand when they can be optimal inside a convex polygon (or amid moderate obstacles [1, 6]). We believe that optimal paths can be of type (B.iii) only if the polygon is "small." In other words, it is possible that optimal paths of type (B.iii) arise only as artifacts of tightly constricted environments (see Figure 3.1c). For example, we know (see Theorem 5.16) that, if the terminal configurations are at a distance greater than 6 from the boundary of the polygon, then an optimal path cannot be of type (B.iii). A second approach to handling paths of type (B.iii) is to simply drop them from consideration. In that case, we ask whether we can preprocess the scene such that, for any query of terminal configurations, we can compute a shortest path among the remaining types in sublinear time.

To conclude with a general problem for future research, we ask for the specification of a realistic notion of feasibility that rejects hard-to-follow paths, such as paths of type (B.iii), while admitting fast computation of optimal feasible paths.

Appendix. Closed moderate paths. The purpose of this section is to prove Proposition 2.5, which states

> *A simple moderate path* Π *such that the initial and final locations coincide (the initial and final configurations may differ) bounds a*

region that contains at least one disk of unit radius. Moreover, if the initial and final configurations coincide and if Π *is not a circle of unit radius, then the region bounded by* Π *contains at least two distinct disks of unit radius.*

Proof. We prove these results using some properties of the skeletons. Note that this is an original approach in the field of nonholonomic motion planning. We first recall a definition of skeletons for which several variants are considered in the literature; different terms in use include *medial-axis, central set*, and *cut-locus*. We use here the formulation using maximal disks [35].

Skeletons. Let $\mathcal{R}$ be an open set of $\mathbb{R}^2$ bounded by a simple closed curve. A *maximal disk* is an open disk (of positive radius) included in $\mathcal{R}$ but not included in any other disk contained in $\mathcal{R}$. The *skeleton* of $\mathcal{R}$, denoted $\mathcal{S}(\mathcal{R})$, is the locus of the centers of all the maximal disks. For any $x \in \mathcal{R}$, let $\rho(x)$ denote the intersection between the closure of the maximal disk centered at x and the boundary of $\mathcal{R}$:

$$\rho(x) = \{y \in \partial\mathcal{R} \mid \|xy\| = \min_{z \in \partial\mathcal{R}} \|xz\|\}.$$

Now let $I = F$ denote the initial and final location on the path Π, and let $\mathcal{R}$ be the open region bounded by Π. Assume that $\mathcal{R}$ is not a disk of radius at least 1, as otherwise the result is obvious.

The underlying idea of the proof is the following. The closure of skeleton $\mathcal{S}(\mathcal{R})$ has at least two distinct endpoints (i.e., nodes of degree 1 of the graph) x and x'; indeed, $\mathcal{S}(\mathcal{R})$ does not contain any cycle since Π is simple. One of these endpoints, say, x, is necessarily distinct from the terminal location I of Π. The point x cannot lie on Π because Π is C^1 continuous everywhere except (possibly) at $I \neq x$; therefore, $x \in \mathcal{R}$. Since x is an endpoint of the skeleton, $\rho(x)$ is connected (i.e., $\rho(x)$ is a point or a circular arc). It follows that the maximal disk at x is osculating[5] Π at a point $P \neq I$, and its radius is at least 1. If the initial and final orientations of Π are also equal, x and x' are both centers of maximal disks osculating Π, and thus their radii are greater than or equal to 1.

Formally, we directly show, using a result by Calabi and Riley [9], the following result.

Claim A.1. *There exists a maximal disk in* $\mathcal{R}$ *such that the contact points between its boundary and* Π *are connected.*

Proof. Let D be any maximal disk in $\mathcal{R}$, and let x be its center, and suppose, for a contradiction, that $\rho(x)$ is not connected. See Figure A.1. Then there exist four points $y_1 \neq y_2, y_1' \neq y_2'$ in $\partial D \cap \Pi$ such that the circular arcs $\partial D^+[y_1, y_2]$ and $\partial D^+[y_1', y_2']$ do not strictly contain any point of $\rho(x)$.

One of the two arcs $\Pi^+[y_1, y_2]$ and $\Pi^+[y_1', y_2']$ is C^1 continuous because these two arcs do not overlap, and Π is C^1 continuous everywhere except possibly at $I = F$. Say, without loss of generality, that $\Pi^+[y_1, y_2]$ is C^1 continuous, and denote by $\mathcal{R}'$ the open region bounded by $\Pi^+[y_1, y_2] \cup [x, y_1] \cup [x, y_2]$ (e.g., the shaded region in Figure A.1). By a result of Calabi and Riley [9, Proposition 10], either $\mathcal{R}'$ contains a skeleton point $x_0 \in \mathcal{SR}$ for which $\rho(x_0)$ is connected, or $\Pi^+(y_1, y_2)$ (without its endpoints) contains a point which is the limit of skeleton points contained in $\mathcal{R}'$.

[5]Two curves are osculating at a point P if and only if they are tangent at P and have the same signed curvature at P. Here Π is C^1 continuous, and piecewise C^2 continuous, everywhere except possibly at I. If Π is C^1 but not C^2 at P, we say that a disk is osculating Π at P if and only if they are tangent at P, the signed curvature of the disk is equal to the signed curvature (at P) of one of the two portions (C^2) of Π ending at P, and the other portion does not properly intersect the disk in a neighborhood of P.

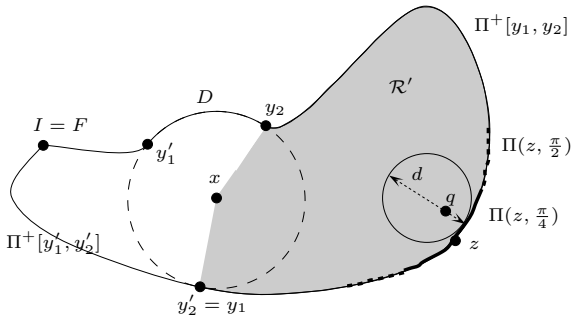


FIG. A.1. *For the proof of Proposition* 2.5.

Suppose, on the contrary, that $\Pi^+(y_1, y_2)$ contains a point z that is the limit of skeleton points $(z_i)_{i\in\mathbb{N}}$ contained in $\mathcal{R}$. Since $\Pi^+(y_1, y_2)$ is C^1, Π is C^1 at z. We get a contradiction because, as we prove below, a point z of $\partial\mathcal{R}$ is a limit of points of the skeleton only if $\partial\mathcal{R}$ is not differentiable at z. Indeed, let $\Pi(z, \frac{\pi}{2})$ be the arc of Π of length $\frac{\pi}{2}$ centered at z (if the length of Π is smaller than $\frac{\pi}{2}$, we choose a smaller value for the length of $\Pi(z, \frac{\pi}{2})$), and let $\Pi(z, \frac{\pi}{4})$ be the arc of Π, centered at z, of length half the length of $\Pi(z, \frac{\pi}{2})$, and, finally, let d be the distance between $\Pi(z, \frac{\pi}{4})$ and $\Pi \setminus \Pi(z, \frac{\pi}{2})$ (note that $d \leqslant \frac{\pi}{4} < 1$). By a result of Dubins [15, Proposition 6], $\Pi(z, \frac{\pi}{2})$ does not intersect any open disk of unit radius tangent to $\Pi(z, \frac{\pi}{4})$. It follows that Π does not intersect any open disk of radius $d/2$ tangent to $\Pi(z, \frac{\pi}{4})$. Now any point $q \in \mathcal{R}$ close enough to z belongs to a normal to $\Pi(z, \frac{\pi}{4})$ at distance $\mu < d/2$ from $\Pi(z, \frac{\pi}{4})$. Thus the disk of radius μ centered at q is not maximal because it is included in a disk of radius $d/2$ tangent to $\Pi(z, \frac{\pi}{4})$, which is included in $\mathcal{R}$. Therefore, any point $q \in \mathcal{R}$ in a small enough neighborhood of z does not belong to the skeleton. □

By Claim A.1, $\mathcal{R}$ contains a point x_0 of its skeleton such that $\rho(x_0)$ is connected. Thus $\rho(x_0)$ is a circular arc or is reduced to a point. If $\rho(x_0)$ is a circular arc, then the constraint on the average curvature of Π implies that this circular arc of Π has a radius greater than or equal to 1, and thus the radius of the maximal disk at x_0 is greater than or equal to 1. Otherwise, if $\rho(x_0)$ is reduced to a point, say, y_0, the maximal disk at x_0 osculates Π at y_0, and thus its radius is greater than or equal to 1. (Indeed, recall that the constraint on the average curvature implies that Π is piecewise C^2 continuous, and its curvature is less than or equal to 1 almost everywhere.) This concludes the proof when the initial and final orientations of Π differ.

If the initial and final configurations are equal, the path Π is C^1 continuous everywhere, and the previous arguments hold for both regions $\mathcal{R}'$ and $\mathcal{R}''$ bounded, respectively, by $\Pi^+[y_1, y_2] \cup [x, y_1] \cup [x, y_2]$ and $\Pi^+[y_1', y_2'] \cup [x, y_1'] \cup [x, y_2']$. (Note, however, that, when $\rho(x)$ is connected, $\mathcal{R}''$ is empty, but then the maximal disk centered at x is of radius at least 1.) □

Acknowledgments. This paper arose from research begun at the International Workshop on Bounded Radius of Curvature, organized by Sue Whitesides and held at the Bellairs Research Institute of McGill University on March 9–16, 1997. We would like to thank Hazel Everett, Micha Godau, Sylvain Petitjean, Kaleem Siddiqi, and Steve Wismath for many useful discussions.

REFERENCES

[1] P. K. Agarwal, P. Raghavan, and H. Tamaki, *Motion planning for a steering-constrained robot through moderate obstacles*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 343–352.

[2] H.-K. Ahn, O. Cheong, J. Matoušek, and A. Vigneron, *Reachability by paths of bounded curvature in convex polygons*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 251–259.

[3] J. Barraquand and J.-C. Latombe, *Nonholonomic multi-body mobile robots: Controllability and motion planning in the presence of obstacles*, Algorithmica, 10 (1993), pp. 121–155.

[4] J.-D. Boissonnat and X.-N. Bui, *Accessibility Region for a Car that Only Moves Forwards along Optimal Paths*, Rapport de recherche 2181, INRIA, Le Chesnay Cedex, France, 1994.

[5] J.-D. Boissonnat, A. Cérézo, and J. Leblond, *Shortest paths of bounded curvature in the plane*, Internat. J. Intell. Syst., 10 (1994), pp. 1–16.

[6] J.-D. Boissonnat and S. Lazard, *A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles*, in Proceedings of the 12th Annual ACM Symposium on Computational Geometry, ACM, New York, 1996, pp. 242–251.

[7] J.-D. Boissonnat and S. Lazard, *A Polynomial-Time Algorithm for Computing a Shortest Path of Bounded Curvature amidst Moderate Obstacles*, Rapport de recherche 2887, INRIA, Chesnay Cedex, France, 1996; Internat. J. Comput. Geom. Appl., to appear.

[8] X.-N. Bui, P. Souères, J.-D. Boissonnat, and J.-P. Laumond, *The shortest path synthesis for non-holonomic robots moving forwards*, in Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, 1994, pp. 2–7.

[9] L. Calabi and J. A. Riley, *The skeletons of stable plane sets*, Scientific Report SR3-5711, Parke Math. Labs., Carlisle, MA, 1967.

[10] S.-W. Cheng, O. Cheong, H. Everett, and R. van Oostrum, *Hierarchical vertical decomposition, ray shooting, and circular arc queries in simple polygons*, in Proceedings of the 15th Annual ACM Symposium on Computational Geometry, ACM, New York, 1999, pp. 227–236.

[11] J. Canny, B. R. Donald, J. Reif, and P. Xavier, *Kinodynamic motion planning*, J. ACM, 40 (1993), pp. 1048–1066.

[12] J. Canny, A. Rege, and J. Reif, *An exact algorithm for kinodynamic planning in the plane*, Discrete Comput. Geom., 6 (1991), pp. 461–484.

[13] G. Desaulniers, *On shortest paths for a car-like robot maneuvering around obstacles*, Robotics and Autonomous Systems, 17 (1996), pp. 139–148.

[14] B. R. Donald and P. Xavier, *A provably good approximation algorithm for optimal-time trajectory planning*, in Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989, pp. 958–963.

[15] L. E. Dubins, *On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents*, Amer. J. Math., 79 (1957), pp. 497–516.

[16] S. Fortune and G. Wilfong, *Planning constrained motion*, Ann. Math. Artificial Intelligence, 3 (1991), pp. 21–82.

[17] D. Halperin, L. Kavraki, and J.-C. Latombe, *Robotics*, in CRC Handbook of Discrete and Computational Geometry, J. Goodmand and J. O'Rourke, eds., CRC Press, Boca Raton, FL, 1997, pp. 755–778.

[18] P. Jacobs and J. Canny, *Planning smooth paths for mobile robots*, in Nonholonomic Motion Planning, Z. Li and J. Canny, eds., Kluwer Academic Publishers, Norwell, MA, 1992, pp. 271–342.

[19] P. Jacobs, J.-P. Laumond, and M. Taix, *Efficient motion planners for nonholonomic mobile robots*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, Osaka, Japan, 1991, pp. 1229–1235.

[20] J.-C. Latombe, *A fast path-planner for a car-like indoor mobile robot*, in Proceedings of the 9th National Conference on Artificial Intelligence, Los Angeles, CA, 1991, pp. 659–665.

[21] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1991.

[22] J.-P. Laumond, *Finding collision-free smooth trajectories for a non-holonomic mobile robot*, in Proceedings of the International Joint Conference on Artificial Intelligence, J. McDermott, ed., Morgan Kaufmann, Los Altos, CA, 1987, pp. 1120–1123.

[23] J.-P. Laumond, P. Jacobs, M. Taix, and R. Murray, *A motion planner for nonholonomic mobile robots*, IEEE Trans. Robotics Automation, 10 (1994), pp. 577–593.

[24] J.-P. Laumond, M. Taix, and P. Jacobs, *A motion planner for car-like robots based on global/local approach*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, Tsuchiura, Japan, 1990, pp. 765–773.

[25] B. Mirtich and J. Canny, *Using skeletons for nonholonomic path planning among obstacles*, in Proceedings of the 9th IEEE International Conference on Robotics and Automation, 1992, pp. 2533–2540.

[26] C. Ó'Dúnlaing, *Motion-planning with inertial constraints*, Algorithmica, 2 (1987), pp. 431–475.

[27] G. Pestov and V. Ionin, *On the largest possible circle imbedded in a given closed curve*, Dok. Akad. Nauk SSSR, 127 (1959), pp. 1170–1172.

[28] J. A. Reeds and L. A. Shepp, *Optimal paths for a car that goes both forwards and backwards*, Pacific J. Math., 145 (1990), pp. 367–393.

[29] J. Reif and H. Wang, *The complexity of the two-dimensional curvature-constrained shortest-path problem*, in Robotics: The Algorithmic Perspective (Houston, TX), P. K. Agarwal, L. E. Kavraki, and M. Mason, eds., A. K. Peters, Natick, MA, 1998, pp. 49–57.

[30] W. Rudin, *Real and Complex Analysis*, McGraw–Hill, New York, 1966.

[31] J. T. Schwartz and M. Sharir, *Algorithmic motion planning in robotics*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 391–430.

[32] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. Overmars, *Multi-level path planning for non-holonomic robots using semi-holonomic subsystems*, in Workshop on the Algorithmic Foundation of Robotics, J.-P. Laumond and M. Overmars, eds., A. K. Peters, Wellesley, MA, 1996, pp. 79–96.

[33] J. Sellen, *Planning paths of minimal curvature*, in Proceedings of the IEEE International Conference on Robotics and Automation, Nagoya, Japan, 1995, pp. 1976–1982.

[34] J. Sellen, *Approximation and decision algorithms for curvature constrained path planning: A state-space approach*, in Robotics: The Algorithmic Perspective, P. K. Agarwal, L. E. Kavraki, and M. Mason, eds., A. K. Peters, Natick, MA, 1998, pp. 59–68.

[35] J. Serra, *Image Analysis and Mathematical Morphology. Vol.* 2. *Theoretical Advances*, Academic Press, New York, 1988.

[36] P. Souères and J.-P. Laumond, *Shortest path synthesis for a car-like robot*, IEEE Trans. Automat. Control, 41 (1996), pp. 672–688.

[37] H. J. Sussmann, *Shortest* 3*-dimensional paths with a prescribed curvature bound*, in Proceedings of the 34th IEEE Conference on Decision and Control, IEEE Control Systems Society, Piscataway, NJ, 1995, pp. 3306–3311.

[38] H. J. Sussmann and G. Tang, *Shortest Paths for the Reeds-Shepp Car: A Worked Out Example of the Use of Geometric Techniques in Nonlinear Optimal Control*, Report SYCON-91-10, Rutgers University, New Brunswick, NJ, 1991.

[39] P. Svestka and M. H. Overmars, *Motion planning for car-like robots using a probabilistic learning approach*, J. Robot. Res., 16 (1997), pp. 119–143.

[40] M. Vendittelli, J.-P. Laumond, and C. Nissoux, *Obstacle distance for car-like robots*, IEEE Trans. Robotics Automation, 15 (1999), pp. 678–691.

[41] H. Wang and P. K. Agarwal, *Approximation algorithms for curvature-constrained shortest paths*, in Proceedings of the 7th ACM–SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 409–418.

[42] G. Wilfong, *Motion planning for an autonomous vehicle*, in Proceedings of the IEEE International Conference on Robotics and Automation, New York, NY, 1988, pp. 529–533.

[43] G. Wilfong, *Shortest paths for autonomous vehicles*, in Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989, pp. 15–20.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1852–1878

PARALLEL INTEGER SORTING IS MORE EFFICIENT THAN PARALLEL COMPARISON SORTING ON EXCLUSIVE WRITE PRAMS*

YIJIE HAN† AND XIAOJUN SHEN†

Abstract. We present a significant improvement for parallel integer sorting. On the EREW (exclusive read exclusive write) PRAM our algorithm sorts n integers in the range $\{0, 1, \ldots, m-1\}$ in time $O(\log n)$ with $O(n \sqrt{\frac{\log n}{k}})$ operations using word length $k \log(m+n)$, where $1 \le k \le \log n$. In this paper we present the following four variants of our algorithm.
(1) The first variant sorts integers in $\{0, 1, \ldots, m-1\}$ in time $O(\log n)$ and in linear space with $O(n)$ operations using word length $\log m \log n$.
(2) The second variant sorts integers in $\{0, 1, \ldots, n-1\}$ in time $O(\log n)$ and in linear space with $O(n\sqrt{\log n})$ operations using word length $\log n$.
(3) The third variant sorts integers in $\{0, 1, \ldots, m-1\}$ in time $O(\log^{3/2} n)$ and in linear space with $O(n\sqrt{\log n})$ operations using word length $\log(m+n)$.
(4) The fourth variant sorts integers in $\{0, 1, \ldots, m-1\}$ in time $O(\log n)$ and space $O(nm^{\epsilon})$ with $O(n\sqrt{\log n})$ operations using word length $\log(m+n)$.
Our algorithms can then be generalized to the situation where the word length is $k \log(m+n)$, $1 \le k \le \log n$.

Key words. algorithms, analysis of algorithms, bucket sorting, conservative algorithms, design of algorithms, integer sorting, parallel algorithms

AMS subject classifications. 68P10, 68Q22, 68Q25

PII. S0097539799352449

1. Introduction.

Sorting is a classical problem which has been studied by many researchers. For elements in an ordered set comparison sorting can be used to sort the elements. It is well known that comparison sorting has time complexity $\theta(n \log n)$. In the case where a set contains only integers both comparison sorting and integer sorting can be used to sort the elements. Since elements of a set are usually represented by binary numbers in a digital computer, integer sorting can, in many cases, replace comparison sorting. The only known time lower bound for integer sorting is the trivial linear bound of $\Omega(n)$. Radix sorting does demonstrate an $O(n)$ upper bound for sorting n integers in the range $\{0, 1, \ldots, n^t - 1\}$, where t is a constant. Researchers worked hard trying to show that, for integers in any range, integer sorting can outperform comparison sorting [4, 13, 19, 21]. Fredman and Willard [13] first showed that n integers in any range can be sorted in $O(n\sqrt{\log n})$ time, thereby demonstrating that in the sequential case, integer sorting is more efficient than comparison sorting. However, prior to this paper no deterministic parallel integer sorting algorithm outperformed the lower bound for parallel comparison sorting on any parallel computation models. (A detailed explanation is given below.) We show, for the first time, that parallel in-

*Received by the editors February 24, 1999; accepted for publication (in revised form) February 19, 2002; published electronically October 18, 2002. A preliminary version of this paper was published in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*'99), Baltimore, MD, 1999.
http://www.siam.org/journals/sicomp/31-6/35244.html
†School of Interdisciplinary Computing and Engineering, University of Missouri at Kansas City, 5100 Rockhill Road, Kansas City, MO 64110 (hanyij@umkc.edu, http://welcome.to/yijiehan, shenx@umkc.edu). The work of the second author was partially supported by UMKC faculty research grant K-2-11191.

teger sorting is more efficient than parallel comparison sorting on the exclusive write PRAMs.

The parallel computation model we use is the PRAM model [20] which is used widely by parallel algorithm designers. Usually three PRAM models are used, the EREW (exclusive read exclusive write) PRAM, the CREW (concurrent read exclusive write) PRAM, and the CRCW (concurrent read concurrent write) PRAM [20]. In a PRAM model a processor can access any memory cell. On the EREW PRAM simultaneous access to a memory cell by more than one processor is prohibited. On the CREW PRAM, processors can read a memory cell simultaneously, but simultaneous writing to the same memory cell by several processors is prohibited. On the CRCW PRAM, processors can simultaneously read or write to a memory cell. The CREW PRAM is a more powerful model than the EREW PRAM. The CRCW PRAM is the most powerful model among the three variants.

Parallel algorithms can be measured either by their time complexity and processor complexity or by their time complexity and operation complexity, which is the time processor product. A parallel algorithm with small time complexity is regarded as fast while a parallel algorithm with small operation complexity is regarded as efficient.

In order to outperform parallel comparison sorting on the exclusive write PRAM models (i.e., CREW PRAM and EREW PRAM), one has to exhibit a parallel algorithm which matches the time lower bound for parallel comparison sorting algorithms and outperforms the operation lower bound for parallel comparison sorting algorithms. Note that we cannot outperform the time lower bound (only to match it) because on the CREW and EREW PRAMs the time lower bounds for parallel comparison sorting and for parallel integer sorting are the same, namely $\Omega(\log n)$ [11]. The operation lower bound for parallel comparison sorting is $\Omega(n \log n)$ due to the time lower bound for sequential comparison sorting. We explain below that known parallel integer sorting algorithms failed to outperform the lower bound for parallel comparison sorting.

1. Parallel algorithms are known [2, 4, 12, 19, 25] to have operation complexity of $o(n \log n)$ when they are running slower than the $\theta(\log n)$ time lower bound for parallel comparison sorting. But they failed to have $o(n \log n)$ operations when achieving the time lower bound. For example, the CREW algorithm given in [2] (the best prior to this paper) has operation complexity $O(n\sqrt{\log n})$ when running at time $O(\log n \log\log n)$. But the time lower bound for comparison sorting on the CREW PRAM is $\Omega(\log n)$ [11]. It is not clear how to make the algorithm in [2] run in $O(\log n)$ time. Also the CRCW algorithm in [4, 19] has operation complexity $O(n \log\log n)$ when running at time $O(\log n)$. However, the time lower bound for comparison sorting on the CRCW PRAM using a polynomial number of processors is $\Omega(\log n/\log\log n)$ [6].

2. Parallel algorithms are known [2, 9, 29] to have operation complexity $o(n \log n)$ running at the time lower bound for parallel comparison sorting when sorting on small integers. They fail to outperform parallel comparison sorting when sorting on large integers. For example, the previous best results in [2, 12] showed that n integers in the range $\{0, 1, \ldots, 2^{O(\sqrt{\log n})}\}$ can be sorted on the EREW PRAM in $O(\log n)$ time and linear operations. However, no previous deterministic algorithms showed that n integers larger than $2^{O(\sqrt{\log n})}$ can be sorted in $O(\log n)$ time with $o(n \log n)$ operations on exclusive write PRAMs.

3. Parallel algorithms are known [4, 16] to outperform parallel comparison sorting by using a nonstandard word length (word length is the number of bits in each word).

However, they fail to outperform on a standard PRAM where word length is bounded by $O(\log(m+n))$. For example, in [4] it is shown that sorting n integers in the range $\{0, 1, \ldots, m-1\}$ can be done in $O(\log n)$ time with $O(n)$ operations on the EREW PRAM with a word length of $O((\log n)^{2+\epsilon} \log m)$. The use of extra bits in word length in parallel integer sorting is generally regarded as excess. Note that even in this case (use nonstandard word length) our results are better than all previous results.

In this paper we show for the first time that on the exclusive write PRAMs parallel integer sorting is more efficient than parallel comparison sorting. For sorting n integers in the range $\{0, 1, \ldots, m-1\}$ our algorithm runs in $O(\log n)$ time with an operation complexity of $O(n\sqrt{\frac{\log n}{k}})$ when using word length $k \log(m+n)$, where $1 \leq k \leq \log n$. When $k = 1$ our algorithm uses standard word length $\log(m+n)$ and runs in $O(\log n)$ time (which is also the lower bound for integer sorting on the CREW and EREW PRAM and which matches the time lower bound for parallel comparison sorting on the CREW and EREW PRAM) with $O(n\sqrt{\log n})$ operations (while parallel comparison sorting has a lower bound $\Omega(n \log n)$ for the operation complexity due to the sequential time complexity lower bound). This algorithm outperforms parallel comparison sorting on the CREW and EREW PRAMs.

There are many previous results on parallel integer sorting [2, 4, 9, 12, 15, 16, 19, 22, 24, 25, 26, 28, 29]. We give a brief comparison of our results with the previous results.

An important parameter in integer sorting is the word length w, which is the number of bits in a word. Much effort has been spent toward finding good integer sorting algorithms which are conservative in the sense that they do not use extra bits. According to Kirkpatrick and Reisch [21] an integer sorting algorithm sorting n integers in the range $\{0, 1, \ldots, m-1\}$ is said to be conservative if the word length is bounded by $O(\log(m+n))$. Significant progress has been made recently in this regard. Andersson et al. [4] and Han and Shen [19] showed conservative integer sorting algorithms that sort n integers in the range $\{0, 1, 2, \ldots, m-1\}$ in $O(\log n)$ time with $O(n \log\log n)$ operations on the CRCW PRAM. This also implies a conservative sequential algorithm with $O(n \log\log n)$ time. Although much progress has been made on parallel integer sorting on the CRCW PRAM [4, 9, 15, 19], which allows simultaneous read and write to shared memory cells, significant difficulties exist when parallel integer sorting algorithms are to be designed on PRAMs which do not allow simultaneous write.

Consider the problem of sorting n integers in the range $\{0, 1, \ldots, n-1\}$, which is the most important and standard case. Previous best conservative parallel algorithms running in $O(\log n)$ time on CREW and EREW PRAMs use $O(n \log n)$ operations. Rajasekaran and Sen [25], Albers and Hagerup [2], and Dessmark and Lingas [12] were able to reduce the number of operations to $o(n \log n)$ on the CREW PRAM and EREW PRAM but the running time must be over $O(\log n)$. Currently the best result due to Albers and Hagerup [2] sorts in $O(\log n \log\log n)$ time with $O(n\sqrt{\log n})$ operations on the CREW PRAM. On the EREW PRAM the algorithms in [2, 25] have $O(\log n \log\log n)$ time complexity with $O(n \log n / \log\log n)$ operations. Very recently Dessmark and Lingas presented an improved EREW algorithm [12] which needs $O(\log^{3/2} n)$ time with $O(n\sqrt{\log n})$ operations. Thus in regard to the best previous results one cannot sort better than the comparison sorting algorithm [1, 10] (which uses $O(n \log n)$ operations) if one is to sort as fast as the comparison sorting algorithm (using $O(\log n)$ time) on the CREW and EREW PRAMs.

In this paper we significantly improve on this situation (i.e., sorting n integers

in the range $\{0, 1, \ldots, n-1\}$). Our EREW PRAM algorithm sorts in $O(\log n)$ time with $O(n\sqrt{\log n})$ operations. Thus our algorithm uses the same number of operations ($O(n\sqrt{\log n})$) as the algorithm by Albers and Hagerup [2] and by Dessmark and Lingas [12] while our algorithm runs faster (in $O(\log n)$ time) than their algorithm (in $O(\log n \log\log n)$ time on the CREW PRAM and in $O(\log^{3/2} n)$ time on the EREW PRAM). Thus our EREW algorithm is faster by a factor of $\log\log n$ than the previous best CREW algorithm and is faster by a factor of $\log^{1/2} n$ than the previous best EREW algorithm.

Now consider the problem of sorting n integers in the range $\{0, 1, 2, \ldots, m-1\}$. All previous EREW and CREW conservative algorithms [2, 12, 22, 25, 29] require $O(n \log n)$ operations when m is large, even when the time complexity is allowed to be polylogarithmic of n. Actually the number of operations of best previous results is larger than $O(n \log n)$, however, we could assume that these algorithms switch to comparison sorting when m is at a certain threshold value. Our result is the first which sorts arbitrarily large integers with $o(n \log n)$ operations. Our EREW integer sorting algorithm sorts in $O(\log n)$ time with $O(n\sqrt{\log n})$ operations. This is for arbitrarily large values of m.

We also present an algorithm (Theorem 4.1) which sorts integers in $\{0, 1, \ldots, m-1\}$ with $O(\log^{3/2} n)$ time and $O(n\sqrt{\log n})$ operations and it runs in linear space. Previously, Dessmark and Lingas [12] achieved this performance only for sorting integers in the range $\{0, 1, \ldots, n^k\}$ for a constant k. This can also be compared with the sequential algorithm in [3].

We now turn to nonconservative integer sorting. Consider the problem of sorting n integers in the range $\{0, 1, 2, \ldots, m-1\}$ on a computer with word length w. Hagerup and Shen [16] showed that if $w = \Omega(n \log n \log m)$, the sorting can be done in linear space and in $O(n)$ sequential time or in $O(\log n)$ time on a EREW PRAM with $O(n)$ operations. Later, Albers and Hagerup [2] and Andersson et al. [4] improved on the word length. Albers and Hagerup [2] showed that with $w = O(\log n \log\log n \log m)$ the sorting can be done in linear space and in $O(\log^2 n)$ time with $O(n)$ operations on the EREW PRAM. The result of Andersson et al. [4] show that the sorting can be done in linear space and in $O(\log n)$ time with $O(n)$ operations on the EREW PRAM with a word length of $O((\log n)^{2+\epsilon} \log m)$. Dessmark and Lingas [12] showed that the sorting can be done on the EREW PRAM in linear space and in $O(\log n \log\log n)$ time and $O(n)$ operations with a word length of $O(\log m \log n)$. In this paper we improve on all these previous results. We show that the sorting can be done in $O(nm^{\epsilon})$ space and in $O(\log n)$ time with $O(n\sqrt{\frac{\log n}{k}})$ operations on the EREW PRAM with a word length of $O(k \log m)$, where k is a parameter satisfying $1 \le k \le \log n$. When $k = \log n$ our algorithm shows that the sorting can be done in linear space and in $O(\log n)$ time with $O(n)$ operations. It is this version of the algorithm that outperforms all previous algorithms. We note that the main focus of this paper is to present conservative EREW algorithms for integer sorting. The nonconservative algorithm we designed is to be used as a subroutine in our conservative algorithms, although our nonconservative algorithm improves on our best previous results.

Algorithms presented in this paper are deterministic algorithms. These algorithms are stable in the sense that input integers of the same value retain their original relative order in the output.

2. Nonconservative sorting. We present an EREW algorithm using word length $O(\log n \log m)$ to sort n integers in the range $\{0, 1, \ldots, m-1\}$ in $O(\log n)$

time with $O(n)$ operations. The input numbers are assumed in an array. This EREW algorithm is based on the Ajtia, Komlós, and Szemerédi (AKS) sorting network [1], Leighton's column sort [23], Albers and Hagerup's test bit technique [2], and the Benes permutation network [7, 8].

The AKS sorting network [1] has $O(\log n)$ stages. Each stage has no more than $n/2$ comparators. Each comparator has two inputs and two outputs. Each comparator can compare two input numbers and route the smaller number to the left output and the larger number to the right output. When the n input numbers are entered into the AKS sorting network and pass through $O(\log n)$ stages of the sorting network they are sorted at the output of the AKS sorting network.

We will use the principle of Leighton's column sort [23]. This principle states that if we put a set S of n numbers into $\log n$ columns with each column containing $n/\log n$ numbers, then the numbers in S will be sorted by a constant number of the following passes: Sort every column and then perform a fixed permutation among the numbers in all columns. Therefore the principle of Leighton's column sort converts the sorting on n numbers to the sorting on $n/\log n$ numbers (there are $\log n$ columns of them and each of them has to be sorted) and fixed permutations among the n numbers. A fixed permutation is a permutation known before program execution. It does not depend on the value of the input numbers. In Leighton's column sort these permutations are shuffle, unshuffle, and shift. Also note that if the principle of column sort is recursively applied we can enlarge the number of columns to n^ϵ, where $0 < \epsilon < 1$ is a constant, and the number of passes of sorting on columns and permutations is still a constant.

Albers and Hagerup's test bit technique [2] can be used for packed numbers. When there are k numbers packed into a word, we can use this technique to do pairwise comparison of the numbers in two words and extract the larger numbers into one word and smaller numbers into another word. An example will make this clear. A test bit of 0 or 1 is added between the numbers packed into a word. Suppose we pack three numbers a_1, a_2, a_3 (each containing t bits) into a word as $w_1 = 1a_1 1a_2 1a_3$ and pack another three numbers b_1, b_2, b_3 (each containing t bits) into a word as $w_2 = 0b_1 0b_2 0b_3$. That is, we add a test bit of 0 or 1 between the numbers. By setting $w_3 = w_1 - w_2$ and then applying a mask we can extract out the test bits of w_3. These test bits tell us which number is larger. By using these test bits we can subsequently extract out the larger numbers from w_1 and w_2 and put them into one word. Similarly we can also extract out the smaller numbers from w_1 and w_2 and put them into one word. Note that this operation takes constant time no matter how many numbers are packed into one word.

The Benes permutation network [7, 8] is a network with $O(\log n)$ stages. Each stage has $n/2$ switches. Each switch has two inputs and two outputs. When the switch is unset the left input goes to the left output and the right input goes to the right output. When the switch is set the left input goes to the right output and the right input goes to the left output. By setting up the switches in the Benes permutation network any permutation of the input can be realized. We will use butterfly networks (explained below) which also have $O(\log n)$ stages and each stage has $n/2$ switches. Butterfly networks can emulate the Benes network and realize any permutation.

LEMMA 2.1. *If the word length is $O(\log n \log m)$, we can pack n integers in $\{0, 1, \ldots, m-1\}$ into $n/\log n$ words in $O(\log \log n)$ time and $O(n)$ operations on the EREW PRAM.*

Proof. We first pack two integers into one word and we have $n/2$ words left. We

then pack the integers in every two words into one word. We repeat this for $\log\log n$ times until we have $\log n$ integers packed into one word. The time is $O(\log\log n)$. The operation is $n + n/2 + n/4 + n/8 + \ldots = O(n)$. □

With $\log n$ integers packed into one word we can view the n integers in $n/\log n$ words as forming $\log n$ columns. The ith column contains the ith integers in all $n/\log n$ words.

LEMMA 2.2. *The $n/\log n$ integers in each of the $\log n$ columns can be sorted in $O(\log n)$ time and $O(n)$ operations on the EREW PRAM.*

Proof. We build an AKS sorting network on these $n/\log n$ words. On the AKS sorting network we compare two words at each internal node of the network using the test bit technique. Thus each node of the AKS sorting network can be used to compare the $\log n$ integers in the word in parallel. At the output of the AKS sorting network we have sorted $\log n$ columns with the ith column containing ith integers in all $n/\log n$ words. The time is $O(\log n)$ since there are $O(\log n)$ stages in the AKS sorting network. The operation is $(n/\log n) \cdot O(\log n) = O(n)$ because we have only $n/\log n$ words and each of them goes through $O(\log n)$ stages. □

LEMMA 2.3. *A fixed permutation among n integers packed into $n/\log n$ words can be done in $O(\log\log n)$ time and $O(n)$ operations on the EREW PRAM.*

Proof. Simply disassemble the integers in the words so that each word contains one integer. Then apply the permutation. Then reassemble the integers into words. □

"$\log n$" in the paper actually stands for the smallest power of 2 no less than $\log n$. This is achieved without increasing the space to superlinear as follows. Take k to be the smallest integer which is a power of 2 and which is no less than $\log n$. Pack k integers into one word. We obtain $l = \lceil n/k \rceil$ words. We take a number l' which is a power of 2 and which is the smallest number no less than l. We assume that we have l' words. Note that here we do not require that $k = \log(l'k)$. The total number of integers is $l'k \leq 3n$.

For our purpose (because we need to sort integers in a linked list) we also need the following scheme to accomplish the permutation mentioned above. The permutation also can be done by routing the integers through a network N, which is the butterfly network in conjunction with a reverse butterfly network (see Figure 2.1). For permutations, N can be used to emulate the Benes permutation network [7, 8]. Each stage of the butterfly network emulates the processor connection along a dimension on the hypercube. (That is, at dimension j numbers at position a and $a\#j$ are the input into one switch and the output to positions a and $a\#j$ where $a\#j$ is obtained by complementing jth bit of a.) When a and $a\#j$ are in different words we switch integers between the words. (In this case every pair going into a switch is coming from a different word.) When a and $a\#j$ are in the same word (because we pack integers into words) we switch integers within words. (In this case every pair going into a switch is coming from the same word. Here we need k to be a power of 2.) Therefore each stage of the butterfly network can be done in constant time. Because the butterfly network has $O(\log n)$ stages, the permutation can be done in $O(\log n)$ time. Note that since the permutations we performed here are fixed permutations, the setting of the switches in the butterfly network can be precomputed.

THEOREM 2.4. *n integers in the range $\{0, 1, \ldots, m-1\}$ can be sorted on the EREW PRAM with word length $O(\log n \log m)$ in $O(\log n)$ time using $O(n)$ operations and $O(n)$ space.*

Proof. By Leighton's column sort we need only apply Lemmas 2.1, 2.2, and 2.3 a

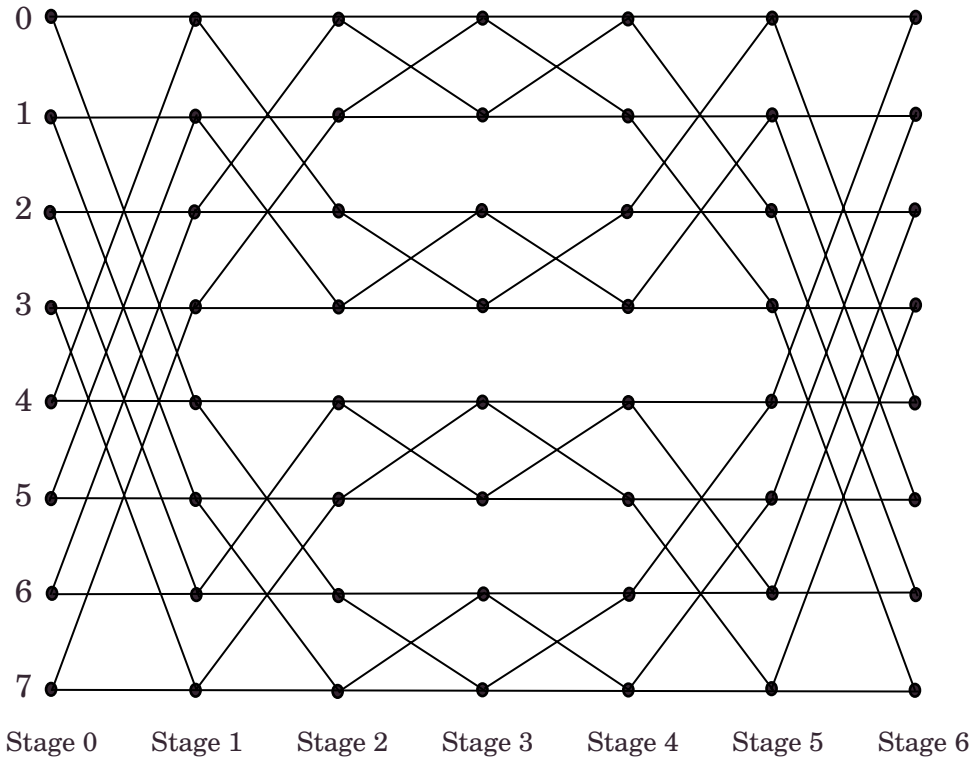


FIG. 2.1. *A permutation network.*

constant number of times. □

The principle of Theorem 2.4 can be applied to the case in which we can pack more than $\log n$ integers into one word. However, in order to apply a recursive version of Leighton's column sort [23] the number of columns cannot be greater than n^ϵ for a constant $0 \leq \epsilon < 1$. Therefore we cannot pack more than n^ϵ integers into one word and then apply the principle of Theorem 2.4. Also note that we may use more columns than the number of integers packed in one word. For example, we may use $\log^2 n$ columns in the column sort even when the number of integers packed in a word is only $\log n$. We will use this fact in the appendix.

The following corollary can now be easily shown.

COROLLARY 2.5. *n integers in the range $\{0, 1, \ldots, m-1\}$ can be sorted on the EREW PRAM with word length $O(k \log m)$, $1 \leq k \leq \log n$, in $O(\log n)$ time using $O(\frac{n \log n}{k})$ operations and $O(n)$ space.*

This corollary is easily obtained by packing k integers into one word and then by applying Lemmas 2.1, 2.2, and 2.3 and Theorem 2.4.

Also note that the sorting can be made stable by appending address bits to each integer. If $m \leq n$, we use an observation which says that n input integers $a_0, a_1, a_2, \ldots, a_{n-1}$ can be divided into n/m sets with ith set containing $a_{im}, a_{im+1}, \ldots, a_{(i+1)m-1}$ and we need only to sort each set. The results of the sorting on each set can be combined to yield the sorted sequence of input integers. This combining takes $O(\log n)$ time and linear operations. This observation is made by Rajasekaran and Sen [25]. Algorithms presented in [2] also make use of this observation. Thus the

address bits used are always being $\min\{\log n, \log m\}$. To add the address bits we can sort in two passes with each pass sorting on $\log m/2$ bits. Therefore the address bits can be added because the number of address bits now is $\min\{\log n, \log m/2\}$.

3. Sorting integers in $\{0, 1, \ldots, n-1\}$. We consider the problem of sorting n integers in the range $\{0, 1, \ldots, n-1\}$ on the EREW PRAM with word length $O(\log n)$. For our purpose we assume that $\sqrt{\log n}$ is a power of 2. The way this is done is to take the smallest integer which is a power of 2 and which is no less than $\sqrt{\log n}$. There is no danger of using superlinear space because this quantity does not determine the space usage.

If input integers with the same value are linked in a linked list in the order they appear in the input, then additional $O(\log n)$ time and $O(n)$ operations suffice for the sorting. This is because we can use linked list contraction [5] to group integers of the same value together. Because we are sorting integers from $\{0, 1, 2, \ldots, n-1\}$ we can use bucket sorting. The first integer in each linked list drops itself into a bucket. Because there are only n buckets, integers dropped into the buckets can be collected in $O(\log n)$ time and $O(n)$ operations. Here the computation is as follows:

1. The first integer (representative integer) in each linked list drops itself into a bucket. This is done for all representative integers in parallel in one step. Because different representative integers have different values the dropping operation has no conflicts among integers.

2. Do a parallel prefix computation to pack integers in the buckets into consecutive locations. This will have all integers dropped into buckets sorted.

Our goal, therefore, is to link integers of the same value into a linked list. Initially, we put all input integers into one linked list. As the computation proceeds, each linked list is split into several linked lists. When the computation ends, all integers with the same value will be linked into a linked list and integers with different values are in different linked lists.

The basic idea of the sorting algorithm is linked list splitting. Let $a_0, a_1, \ldots, a_{n-1}$ be the input integers. The algorithm has $\sqrt{\log n}$ stages. In each stage we examine $\sqrt{\log n}$ bits (we say that we reveal $\sqrt{\log n}$ bits). Initially no bits are revealed. In the first stage we reveal the most significant $\sqrt{\log n}$ bits. In the second stage we reveal the next $\sqrt{\log n}$ bits, and so on. We maintain the property that all integers are linked in a linked list if their revealed bits are the same (of the same value). If the revealed bits for two integers are different, then the two integers are in different linked lists. Initially, all integers are linked into one linked list with a_{i+1} following a_i in the linked list. After the first stage, the input linked list is split into at most $2^{\sqrt{\log n}}$ linked lists because $\sqrt{\log n}$ bits are revealed. After the second stage each linked list further splits itself into at most $2^{\sqrt{\log n}}$ linked lists, and so on.

Now we discuss how each linked list is split in each stage. A linked list is short if it contains fewer than $2^{4\sqrt{\log n}}$ elements and is long if it contains at least $2^{4\sqrt{\log n}}$ elements. We first group every consecutive S elements (integers) in the linked list into one group. For a short linked list, S is the number of total elements in the linked list. For a long linked list, S varies from group to group but is at least $2^{4\sqrt{\log n}}$ and no more than $2^{5\sqrt{\log n}}$. We can consider a grouping as contracting the S elements into one node and/or as ranking the S elements along the linked list within the group. This grouping can be done by linked list contraction algorithms [5, 17, 18]. We then sort integers in each group in parallel. Because revealed bits for the previous stages for integers in the linked list are identical and because we reveal additional $\sqrt{\log n}$ bits in this stage, we are, in fact, sorting no more than $2^{5\sqrt{\log n}}$ $\sqrt{\log n}$-bit integers in each

group. By our nonconservative sorting algorithm presented in the previous section, the sorting can be done in $O(\sqrt{\log n})$ time and $O(S)$ operations for the group (or $O(n)$ operations for all linked lists). Note that if a short linked list contains too few integers (say t integers), we cannot pack $\sqrt{\log n}$ integers into one word and then apply column sort (see the paragraph immediately below Theorem 2.4). In this situation we pack a smaller number of integers ($\log t$ integers) into a word to facilitate column sorting. For example, if a short linked list contains only $\sqrt{\log n}$ integers, then we pack $\log(\sqrt{\log n})$ integers into a word and then apply column sort.

If the linked list is short, there is only one group in the linked list. The sorting will then enable us to split the linked list into $t \le 2^{\sqrt{\log n}}$ linked lists such that each split linked list contains all integers whose revealed bits are the same, where t is the number of bit patterns for the revealed bits. Here we note that for a short linked list, t could be less than $2^{\sqrt{\log n}}$. (For example if the revealed bits for all integers are the same, t will be equal to 1.)

If the linked list is long, we will always split the linked list into exactly $2^{\sqrt{\log n}}$ linked lists no matter how many different bit patterns are revealed by the revealed bits. After sorting in each group, integers in each group are split into $2^{\sqrt{\log n}}$ linked lists. If a bit pattern among the $2^{\sqrt{\log n}}$ bit patterns does not exist in the revealed bits, we create a linked list containing only one dummy element representing this pattern. Note that no more than $2^{\sqrt{\log n}}$ dummy elements will be created for each group. For consecutive (neighboring) groups on a long linked list we then join the split linked lists in the groups such that linked lists with the same revealed bits are joined together. With the help of those dummy elements we now have split a long linked list into exactly $2^{\sqrt{\log n}}$ linked lists.

With the existence of dummy elements in the linked list, the splitting process should be modified a little bit. For a short linked list, after the grouping all dummy elements will be eliminated. For a long linked list, the dummy elements will also be eliminated after grouping, but new dummy elements could be created because we need to split each long linked list to $2^{\sqrt{\log n}}$ linked lists.

Since each group on a long linked list has at least $2^{4\sqrt{\log n}}$ elements and since each such a group creates at most $2^{\sqrt{\log n}}$ dummy elements, the total number of dummy elements created in a stage is at most $n/2^{3\sqrt{\log n}}$. Dummy elements generated in a stage are eliminated in the next stage and new dummy elements are generated for the next stage. For a total of $\sqrt{\log n}$ stages the total number of dummy elements generated is no more than $cn\sqrt{\log n}/2^{3\sqrt{\log n}}$ for a constant c. Here, constant c may be greater than 1 because as dummy elements are generated we have $n' > n$ elements now. In this situation the number of dummy elements generated in the next stage will be $n'/2^{3\sqrt{\log n'}}$.

Let us estimate the complexity. Since each stage takes $O(\sqrt{\log n})$ time and $O(n)$ operations, for a total of $\sqrt{\log n}$ stages the time complexity of the algorithm is $O(\log n)$ with $O(n\sqrt{\log n})$ operations.

Although we have left out several implementation details we hope our presentation in this section can convince most readers that our algorithm works. We therefore give the following theorem. The implementation details are very much ad hoc and several known techniques are adapted to make our implementation fit. The implementation details are described in Appendix A.

THEOREM 3.1. *n integers in the range $\{0, 1, 2, \ldots, n-1\}$ can be sorted in $O(\log n)$ time and $O(n)$ space with $O(n\sqrt{\log n})$ operations on the EREW PRAM with word length $O(\log n)$.*

4. Sorting integers in $\{0, 1, \ldots, m-1\}$. Consider the problem of sorting integers in the range $\{0, 1, \ldots, m-1\}$. All known conservative CREW and EREW parallel algorithms [2, 12, 25], even allowing for polylogarithmic running time, will eventually use $O(n \log n)$ operations when m is sufficiently large. In this section we present two conservative EREW algorithms which use only $O(n\sqrt{\log n})$ operations no matter how large m is. Our first algorithm runs in $O(\log^{3/2} n)$ time and $O(n)$ space with $O(n\sqrt{\log n})$ operations. This algorithm also serves the purpose of introducing ideas to be used in our second algorithm. The basic ideas used in both algorithms are the same. However, the second algorithm is more complicated than the first algorithm. Our second algorithm is the only algorithm in this paper which uses more than linear space. This algorithm sorts in $O(\log n)$ time and $O(nm^{\epsilon})$, $0 < \epsilon < 1$, space with $O(n\sqrt{\log n})$ operations. We note that the linked list splitting idea presented in the previous section does not apply here and therefore new ideas are needed.

First let us outline our approach. We use *bit* $[i]$ to denote bits $i \log m/\sqrt{\log n}$ through $(i+1)\log m/\sqrt{\log n} - 1$. (Bits are counted from the least significant bit starting at 0.) $[i : j]$ is used to denote bits $[i], [i+1], \ldots, [j]$ (or empty if $j < i$). We use $a^{[i]}$ to denote bits $i \log m/\sqrt{\log n}$ through $(i+1)\log m/\sqrt{\log n} - 1$ of a. $a^{[i:j]}$ is used to denote bits $a^{[i]}, a^{[i+1]}, \ldots, a^{[j]}$ (or empty if $j < i$). To sort n integers with each integer containing $\log m$ bits we could use $\sqrt{\log n}$ passes. The ith pass, $0 \le i < \sqrt{\log n}$, sorts bit $[\sqrt{\log n} - i - 1]$. Note that we are sorting from high order bits to low order bits. At the beginning of ith pass the input integers are divided into a collection C of sets such that integers in one set have the same value in bits $[\sqrt{\log n} - i : \sqrt{\log n} - 1]$. In the ith pass we can sort integers in each $s \in C$ independently and in parallel. We call the sorting problem formed by integers in an $s \in C$ an independent (sorting) problem (or an I-problem for short). The sorting in the ith pass further subdivides each $s \in C$ into several sets with each set forming an I-problem for the next pass. Note that if a set s_1 resulting from the subdivision (sorting in the ith pass) of $s \in C$ is a singleton, then the integer $a \in s_1$ needs not to be passed to the next pass because a has been distinguished from other integers and the final rank of a can be determined. When we say an I-problem p we refer to the integers passed from the previous pass to the current pass which form p. When integers in p are sorted in the current pass some of them will be passed to the next pass and these integers are no longer in p. After the current pass finishes, p refers to those integers in the singletons which remained and did not pass to the next pass. Because in a pass we sort $\log m/\sqrt{\log n}$ bits only while each word has $\log m$ bits, each pass can be computed with $O(n\sqrt{\log n})$ operations (by the corollary to Theorem 2.4). This will give us a total of $O(n \log n)$ operations for the algorithm. To reduce the number of operations, we pipeline all passes. Integers will be passed from the ith pass to the $(i+1)$st pass as soon as enough numbers of integers with the same bits $[\sqrt{\log n} - i - 1 : \sqrt{\log n} - 1]$ are accumulated instead of at the end of the ith pass. The details will be explained in the following subsection.

4.1. Sort in $O(\log^{3/2} n)$ time and $O(n\sqrt{\log n})$ operations. We give an outline which explains the essence of our ideas. Suppose we are sorting n integers. If we pass these n integers through the AKS sorting network [1], we will get $O(\log n)$ time and $O(n \log n)$ operations. This is $O(\log n)$ operations per integer. The reason each integer incurs $O(\log n)$ operations is that it must be compared across n integers. If we sort n^t, $0 < t < 1$, integers, then each integer compares across n^t integers, and therefore each integer incurs only $O(t \log n)$ operations. If we take the most significant $\log m/\sqrt{\log n}$ bits out of $\log m$ bits from each integer to form a small integer, we can pack $\sqrt{\log n}$ small integers into one word and therefore we have only $n/\sqrt{\log n}$

words to handle. Sorting these words through the AKS sorting network takes only $O(n\sqrt{\log n})$ operations. However, this sorting may not accomplish the sorting for the input integers. The problems occur when two integers have the same $[\sqrt{\log n}-1]$ bits, i.e., small integers obtained from them are equal. We treat this problem in this way. We use several levels for sorting. We do not sort all small integers at once. At the level 0 we divide them into groups with each group containing $2^{\sqrt{\log n}}$ integers. In the first stage we sort integers in each group. This entails constant operations for each small integer because each small integer has only $\sqrt{\log n}$ bits. In each next stage we merge $2^{\sqrt{\log n}}$ groups into one group. If the sorting/merging reveals that several small integers are equal, then we remove all these small integers and replace them with a dummy. This dummy has the same value as the small integers removed. Removed integers will participate in sorting at level 1 where their $[\sqrt{\log n}-2]$ bits are sorted. Since removed small integers no longer participate in sorting in level 0, each of them incurred a few operations. The operations incurred by each integer at levels greater than 0 depend on when it is passed from level 0. If it is passed down at early stages, it incurs few operations at level 0, but it will incur more operations at higher numbered levels. If it is passed down at later stages, it incurs more operations at level 0, but it will incur less operations at higher numbered levels. This is because if the integer is passed down at later stages, there are only a few integers with the same bit value at bits $[\sqrt{\log n}-1]$ as itself, and therefore the I-problem formed at higher numbered levels is of smaller size, and therefore it incurs fewer operations at higher numbered levels.

In our algorithm the computation is organized into $\sqrt{\log n}$ levels. Each level represents a pass explained in the paragraph before this subsection. There are $\sqrt{\log n}$ stages in each level and stage i_1 at level l_1 is executed concurrently with stage i_2 at level $l_2 > l_1$, where $i_1 - i_2 = l_2 - l_1$. Each stage takes $O(\log n)$ time and $O(n)$ operations. Because there are a total of $2\sqrt{\log n}-1$ stages our algorithm takes $O(\log^{3/2} n)$ time and $O(n\sqrt{\log n})$ operations. The computation at level i, $0 \le i < \sqrt{\log n}$, is to work on bits $[\sqrt{\log n}-i-1]$. We use array $I[0:n-1]$ to represent the n input integer and use $I[i:j]$ to denote $I[i], I[i+1], \ldots, I[j]$. Although the computation at each level is similar, describing the computation at an arbitrary level would be too complicated. Instead we first give pseudo codes outlining the overall structure of the algorithm and then describe the computation at levels 0 and 1 and then generalize it to arbitrary levels.

```
Sort1(A)
{
a.    for i = 0 to n − 1 do in parallel
b.        { A[0][i] = A[i]; /* Put integers at level 0 */

c.    for k = 0 to 2√(log n) − 2 do /*stage k */
d.    {
e.        if k < √(log n), then for l = 0 to k do in parallel B(A, k, l); /* level l */
f.        else for l = k − √(log n) + 1 to √(log n) − 1 do in parallel B(A, k, l);
g.    }
}

B(A, k, l)
{
    1.    for j = 0 to n/2^((k−l+1)√(log n)) and i = i(j) = j2^((k−l+1)√(log n)) to
          (j + 1)2^((k−l+1)√(log n)) − 1 do in parallel

    2.        /* section j, here i(j) indicates that this i is coming from section j. */
```

```
3.      {
4.          if Meet[j][(A[l][i(j)])^[√log n − l, √log n − 1]][(A[l][i(j)])^[√log n − l − 1]], then
5.              /* Integers in the same I-problem (indicated by [(A[l][i(j)])^[√log n − l, √log n − 1]]
                at level l in section j meet if they have the same (A[l][i(j)])^[√log n − l − 1] value. Meet is true
                if at lease two integers meet.*/
6.          {
7.              if l < √log n − 1 then
8.              {
9.                  move A[l][i(j)] to A[l + 1][i(j)]; /* move to next level*/
10.             make one integer among the meet become a dummy with value A[l][i(j)],
                then delete A[l][i(j)];
11.             }
12.         }
13.     }
}
```

The above procedure outlines the overall process of sorting. The Meet operation in procedure B is actually done by sorting and merging. The above procedures give the precise relationship among stage(k), sections(j), levels(l), I-problem (bits $[\sqrt{\log n} - l, \sqrt{\log n} - 1]$), and the value being sorted on (bits $[\sqrt{\log n} - l - 1]$).

The computation at different levels is illustrated in Figure 4.1 and the Meet operation at different levels is illustrated in Figure 4.2.

The computation at level 0 is to sort the n input integers by their most significant $\log m/\sqrt{\log n}$ bits. Each stage at level 0 is to merge $2^{\sqrt{\log n}}$ sorted sequences. (This is indicated by the Meet operation in line 4 of procedure B.) That is, the sorting at level 0 is guided by a complete $2^{\sqrt{\log n}}$-ary tree. Each level of the tree represents the $2^{\sqrt{\log n}}$-way merge in a stage. After stage s and before stage $s + 1$ there are $n/2^{(s+1)\sqrt{\log n}}$ sorted sequences (which are the sections in the procedure B). Suppose integer $a^{[\sqrt{\log n}-1]}$ is in the sorted sequence S. a will remain in level 0 of the algorithm as long as there are fewer than $2^{\sqrt{\log n}}$ integers $b^{[\sqrt{\log n}-1]}$ in S such that $a^{[\sqrt{\log n}-1]} =$

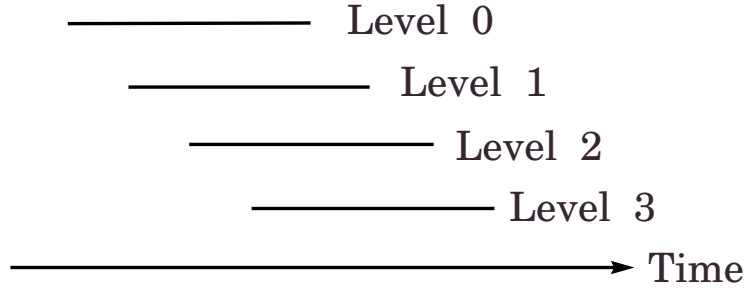


FIG. 4.1. *Computation at different levels is pipelined.*

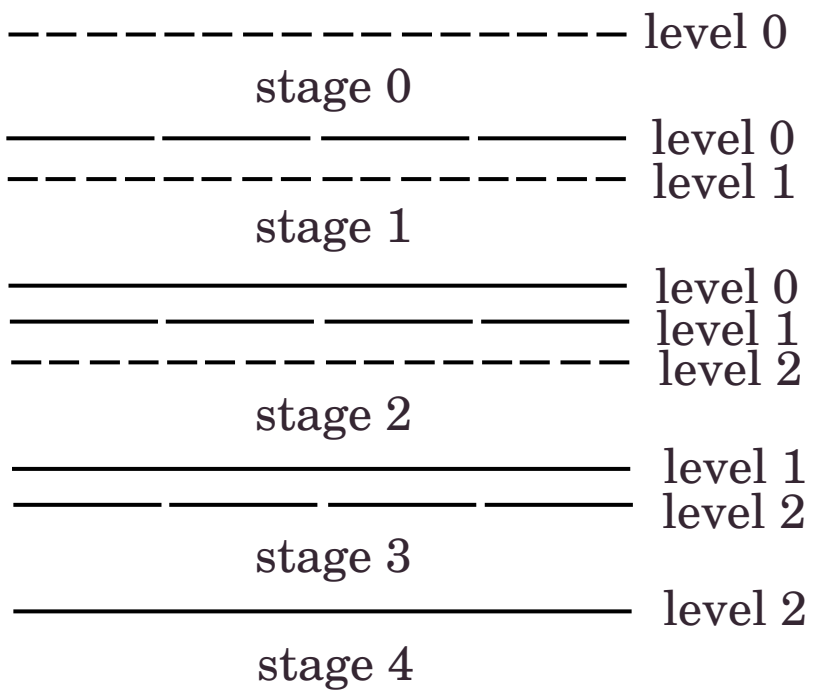


FIG. 4.2. *The merging (Meet) process at different stages.*

$b^{[\sqrt{\log n}-1]}$ (which is what we mean by Meet in procedure B). (Note that all these integers are now consecutive in memory.) Once this condition is not satisfied a will be moved to level 1. Thus one function of level 0 is to group integers $a^{[\sqrt{\log n}-1]}$ and once there are enough integers of the same value grouped together they are sent to level 1. When enough integers of the same value $a^{[\sqrt{\log n}-1]}$ are grouped together in a sorted sequence S and are sent to level 1, we create a dummy with value $a^{[\sqrt{\log n}-1]}$ and place this dummy in S in level 0 to replace the integers sent to level 1. If in a subsequent merge some integers of the same value $a^{[\sqrt{\log n}-1]}$ are grouped together with the dummy (again Meet in procedure B), all these integers (no matter how many) are sent to level 1 and we need only one dummy to represent these integers at level 0. Of course when dummies with the same value $a^{[\sqrt{\log n}-1]}$ are grouped together by the merge only one dummy needs to remain while others can be discarded. After the sorting (i.e., all $\sqrt{\log n}$ stages) in level 0 finishes, integers remain in level 0 are those that do not have $2^{\sqrt{\log n}}$ or more input integers with the same $a^{[\sqrt{\log n}-1]}$ value. For integers with the same $a^{[\sqrt{\log n}-1]}$ value in level 0 (there are less than $2^{\sqrt{\log n}}$ of them) we sort them by their whole integer value (not just the most $\log m/\sqrt{\log n}$ bits) by comparison sorting [1, 10]. Because each such comparison sorting is on no more than $2^{\sqrt{\log n}}$ elements, the number of operations will be bounded by $O(n\sqrt{\log n})$. After this comparison sorting all integers and dummies at level 0 are sorted. Level 0 has divided integers passed to level 1 into I-problems. Integers a with the same $a^{[\sqrt{\log n}-1]}$ value which are passed to level 1 are in one such I-problem. Now we need to sort integers in each I-problem independently and in parallel.

Now consider the computation at level 1. We consider only one I-problem. The

problem is to sort all integer a's by $a^{[\sqrt{\log n}-2]}$ value, where the value $a^{[\sqrt{\log n}-1]}$ for all a's are identical. The sorting at level 1 also has $\sqrt{\log n}$ stages and is also guided by a conceptual $2^{\sqrt{\log n}}$-ary tree. However, many of the leaves may be empty because no integer is passed from level 0. Stage s at level 0 and stage $s-1$ at level 1 (and stage $s-i$ at level i) are executed concurrently. Immediately after stage 0 at level 0 all integers a with the same $a^{[\sqrt{\log n}-1]}$ values in $I[i2^{\sqrt{\log n}} : (i+1)2^{\sqrt{\log n}}-1]$ are grouped together, where $0 \le i < n/2^{\sqrt{\log n}}$. If there are integers a in $I[i2^{\sqrt{\log n}} : (i+1)2^{\sqrt{\log n}}-1]$ which have the same $a^{[\sqrt{\log n}-1]}$ value and are passed down to level 1 in stage 0 of level 0, these integers are merged (sorted) by the $2^{\sqrt{\log n}}$-way merge on the $a^{[\sqrt{\log n}-2]}$ value in stage 0 at level 1. In stage 1 at level 0 all integers a with the same $a^{[\sqrt{\log n}-1]}$ values in $I[i2^{2\sqrt{\log n}} : (i+1)2^{2\sqrt{\log n}}-1]$ are grouped together, where $0 \le i < n/2^{2\sqrt{\log n}}$. Consider integers a with the same $a^{[\sqrt{\log n}-1]}$ value in $I[i2^{2\sqrt{\log n}} : (i+1)2^{2\sqrt{\log n}}-1]$. These integers are grouped into a collection C of at most $2^{\sqrt{\log n}}$ groups (which are sections in procedure B) in stage 0 of level 0 (group j coming from $I[j2^{\sqrt{\log n}} : (j+1)2^{\sqrt{\log n}}-1]$, $i2^{\sqrt{\log n}} \le j \le (i+1)2^{\sqrt{\log n}}-1$). These $2^{\sqrt{\log n}}$ groups are further grouped into one group G in stage 1 of level 0. If some groups in C are passed down to level 1 at stage 0 of level 0, these passed down groups are sorted in stage 0 at level 1 (which execute in parallel with stage 1 of level 0). Note the relation between sections and levels in procedure B. If there is at least one group passed down to level 1, there will be a dummy at level 0 and therefore all integers in G will be passed down at stage 1 of level 0. By using the dummies at level 0 we will be able to build a linked list to link integers passed down at stage 0 with integers passed down at stage 1. And by executing linked list ranking [5] we can then move all integers in G into consecutive memory locations. Here list ranking takes $O(|G|)$ operations and $O(\log n)$ time. Note that linked list linking and ranking here also maintain the stable property for sorting. Our intention is to do a $2^{\sqrt{\log n}}$-way merge (one way for integers in a group in C) at stage 1 of level 1. However, integers in G which are passed down at stage 1 of level 0 (denote this set by G') are not sorted by $a^{[\sqrt{\log n}-2]}$ and therefore they cannot participate in the $2^{\sqrt{\log n}}$-way merge directly. What we do is to first sort integers in G' by bit $[\sqrt{\log n}-2]$ and then perform the merge. Because G' contains less than $2^{2\sqrt{\log n}}$ integers and because sorting is performed on integers each having $\log m/\sqrt{\log n}$ bits, the sorting can be done in $O(\sqrt{\log n})$ time and linear operations by Theorem 2.4.

Thus at level 1 we are forming sorted sequences (sections in procedure B and sorted by bits $[\sqrt{\log n}-2 : \sqrt{\log n}-1]$) and repeatedly merge the sorted sequences (to form larger sections). Suppose integer $a^{[\sqrt{\log n}-2:\sqrt{\log n}-1]}$ is in the sorted sequence S. a will remain in level 1 as long as there are less than $2^{\sqrt{\log n}}$ integers $b^{[\sqrt{\log n}-2:\sqrt{\log n}-1]}$ in S such that $a^{[\sqrt{\log n}-2:\sqrt{\log n}-1]} = b^{[\sqrt{\log n}-2:\sqrt{\log n}-1]}$. (Note that all these integers are now consecutive in memory.) Once this condition is not satisfied a will be moved to level 2 and we will create a dummy with value $a^{[\sqrt{\log n}-2:\sqrt{\log n}-1]}$ at level 1 to replace the integers moved to level 2. As we did in level 0, for integers a stayed in level 1 and never passed to level 2, there are less than $2^{\sqrt{\log n}}$ integers with the same $a^{[\sqrt{\log n}-2:\sqrt{\log n}-1]}$ values and therefore we can sort them by their whole integer value using parallel comparison sorting after the $(\sqrt{\log n}-1)$st stage at level 1.

The relation of level 2 to level 1 is the same as that of level 1 to level 0. In general, integers passed to level i are divided to belong to I-problems with each problem containing integer a's with the same $a^{[\sqrt{\log n}-i:\sqrt{\log n}-1]}$ value. In each such problem at level i, integers are either sorted at level i (by a repeated $2^{\sqrt{\log n}}$-way

merge) or passed down to level $i+1$. Integers passed to level $i+1$ are divided at level i into I-problems such that all integer a's with the same $a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ value are in one I-problem.

There are a total of $2\sqrt{\log n}-1$ stages executed in our algorithm. After these stages and after we use parallel comparison sorting to sort integer a's with the same $a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ value at level i, integers at all levels are sorted. We can then build a linked list. For integers and dummies in each I-problem we simply let each element point to the next element. We then "insert" integers sorted in each I-problem p at level i into the position of the corresponding dummy at level $i-1$ by using a pointer from the dummy to point to the first integer in p and using another pointer from the last integer in p to point to the successor of the dummy. We therefore build a linked list for all the integers and these integers are in their sorted order in the linked list. After a linked list ranking [5] we have all the integers sorted.

At the end of each stage of our algorithm we use linked list ranking [5] and standard parallel prefix computation [20] to move integers and dummies belonging to each I-problem in consecutive memory locations so that the next stage can proceed. For example, integers in an I-problem p at level i need to be packed to consecutive memory locations because some integers in p are passed to level $i+1$. When some integers and dummies in p are grouped into one group by the merging at level i because they have the same $a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ value, we build a linked list to link the integers at level i with the integers already at level $i+1$. (They are represented by the dummies at level i.) We use linked list ranking and prefix computation to move these integers in one group into consecutive memory locations. Because linked list ranking and prefix computation can be done in $O(\log n)$ time and $O(n)$ operations they are within the time and the number of operations allocated to each stage. Note here that we generate one dummy for at least $2^{\sqrt{\log n}}$ integers in each stage. Thus for all stages the total number of dummies generated is bounded by $2n\sqrt{\log n}/2^{\sqrt{\log n}}$.

Now we discuss the $x \le 2^{\sqrt{\log n}}$-way merge performed on a collection C of x sorted sequences in each stage at each level. The integers to be merged are in consecutive memory locations and processors can be easily allocated to them. The integers we are considering here have only $\log m/\sqrt{\log n}$ bits while each word has $\log m$ bits. We have to accomplish the merge in $O(\log n)$ time and a linear number of operations. If the total number of integers to be merged together is $N < 2^{2\sqrt{\log n}}$ we simply sort them by using Theorem 2.4. Otherwise, $N \ge 2^{2\sqrt{\log n}}$ and we sample every $2^{\sqrt{\log n}}$-th integer from each of the $x \le 2^{\sqrt{\log n}}$ sorted sequences (to be merged). If a sequence has no more than $2^{\sqrt{\log n}}$ integers, we sample its first and last integers. The total number of sampled integers is no more than $2N/2^{\sqrt{\log n}}$. We sort all sampled integers into one sequence S using parallel comparison sorting [1, 10]. We make x copies of S. We then merge one copy of S with one sequence in C. Suppose s_1, s_2, $s_1 \le s_2$, are two consecutive integers in S. Then there are no more than $2^{\sqrt{\log n}}$ integers in each sequence in C which are $\le s_2$ and $\ge s_1$. (For equal integers their order is determined first by the sequence they are in and then by the position they are in the sequence.) These integers form a merging subproblem. Because S is merged with each sequence in C, the original merging problem is now transformed into $|S|+1$ ($|S|$ is the number of integers in S) merging subproblems; each of them is to merge x subsequences with each subsequence containing at most $2^{\sqrt{\log n}}$ integers (they come from a sequence in C). For each merging subproblem we use Theorem 2.4 to sort all integers in the subproblem.

It can now be checked that the $2^{\sqrt{\log n}}$-way merge in each stage at each level

takes $O(\log n)$ time and linear operations. At the end of each stage we use linked list ranking and parallel prefix computation to move integers belonging to each I-problem into consecutive memory locations. This computation takes $O(\log n)$ time and $O(n)$ operations. Therefore each stage takes $O(\log n)$ time and $O(n)$ operations.

THEOREM 4.1. *n integers in the range $\{0, 1, \ldots, m-1\}$ can be sorted in $O(\log^{3/2} n)$ time and $O(n)$ space with $O(n\sqrt{\log n})$ operations and $O(\log(m+n))$ word length on the EREW PRAM.*

4.2. Sort in $O(\log n)$ time and $O(n\sqrt{\log n})$ operations. Our second algorithm will run in $O(\log n)$ time with $O(n\sqrt{\log n})$ operations. This algorithm is more complicated. To achieve $O(\log n)$ time we have to allocate only $O(\sqrt{\log n})$ time to each stage. An immediate problem is the following. Consider an I-problem p at level 1. Integers are passed to p at different stages. Suppose several stages have passed and each sorted sequence in p is pretty long. Now in the current stage a set S of integers are passed down from level 0 to p. Although the number of integers in S are few, to merge integers in S with the sorted sequences in p takes a long time because a sorted sequence in p contains too many integers. The problem here is that not all integers in p are passed down in one stage—some are passed down earlier while others are passed down later. If, for example, all integers in p are passed from level 0 at stage 0 (of level 0), then we can merge the integers in p. To avoid the problem that integers are passed down at different stages, we modify our first algorithm (given in Theorem 4.1) as follows.

We append $\log n$ bits to each input integer to indicate the position of each integer in the input. Note that we can assume $\log m > 2 \log n$. To put $\log n$ bits into a integer we could sort in two passes with each pass sorting $\log m/2$ bits. Then in each pass we can put $\log n$ bits into each integer. The process of our sorting is not stable. Adding the $\log n$ address bits stabilizes the sorting. At each level our algorithm sorts $\log m/\sqrt{\log n}$ bits. In the process of our algorithm execution, an integer a in a sorted sequence at level i will stay in level i as long as the number of integers b in the sorted sequence with $b^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]} = a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ is less than $2^{4\sqrt{\log n}}$. Once this condition is not satisfied, a will be passed to level $i+1$. We keep a dummy at level i to replace the integers passed down to level $i+1$. If there are integers c with $c^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]} = a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ left at level i, they keep grouping as integers at level i are merged. If dummies and integers of the same $[\sqrt{\log n} - i - 1 : \sqrt{\log n} - 1]$ bit values are grouped together, we view dummies as smaller than integers and therefore we group dummies with dummies and integers with integers if they have the same value in bits $[\sqrt{\log n}-i-1 : \sqrt{\log n}-1]$. Note that here we do not pass integers to level $i+1$ when integers are grouped with dummies. Instead we keep grouping more integers together. Once the new group contains $2^{4\sqrt{\log n}}$ or more integers with the same $c^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ value, all integers in the group will be passed to level $i+1$. Integers which remain at a level to the end of the last stage will have less than $2^{4\sqrt{\log n}}$ integers with the same revealed bit values. We can then sort them by their whole integer value by using comparison sorting.

We define a grouped subproblem of sorting (G-problem for short). The n input integers are initially in a G-problem p. As computation proceeds, some integers in p will be passed to level 1 and will no longer be in p. Some dummies will be created and added to p. *All integers at level $i+1$ which are passed from a G-problem at level i at stage t of level i form a G-problem.*

Consider a G-problem at level i. Integers in different G-problems are sorted independently even if they have the same value in bits $[\sqrt{\log n} - i - 1 : \sqrt{\log n} - 1]$.

Integers in a G-problem may have different values in bits $[\sqrt{\log n}-i-1:\sqrt{\log n}-1]$. Thus a G-problem is further divided into I-problems such that each I-problem contains all the integers in the G-problem which have the same bits $[\sqrt{\log n}-i-1:\sqrt{\log n}-1]$. Note that here the definition of an I-problem is slightly different than that which we defined before because now we require integers in an I-problem to be those within the same G-problem. All I-problems in a G-problem can be solved independently.

For the G-problem p at level 0 we execute $(1/2)\log\log n-1$ stages. In the ith stage we sort all $2^{2^{i+2}\sqrt{\log n}}$ integers by their bit $[\sqrt{\log n}-1]$. That is, in the ith stage every array $I[j2^{2^{i+2}\sqrt{\log n}},(j+1)2^{2^{i+2}\sqrt{\log n}}-1]$, $0\le j<n/2^{2^{i+2}\sqrt{\log n}}$, is sorted. Integers may be passed to level 1 at different stages and form different G-problems at level 1. Because there are $(1/2)\log\log n-1$ stages at level 0, only $(1/2)\log\log n-1$ G-problems are created at level 1. Consider integers passed to level 1 at the ith stage which form a G-problem q at level 1. Integers in q with different bit $[\sqrt{\log n}-1]$ are in different I-problems. Consider an I-problem r in q. Because integers are passed at the ith stage there are at most $S=n/2^{(2^{(i+1)}-4)\sqrt{\log n}}$ integers in r for $i\ge 1$ and there are at most $S=n$ integers in r for $i=0$. We can store the integers in r in an array A of size S as follows. For stage 0, integers in r which are passed from $I[j2^{4\sqrt{\log n}},(j+1)2^{4\sqrt{\log n}}-1]$, $0\le j<n/2^{4\sqrt{\log n}}$, at level 0 are stored in $A[j2^{4\sqrt{\log n}},(j+1)2^{4\sqrt{\log n}}-1]$, $0\le j<n/2^{4\sqrt{\log n}}$. For stage $i>0$, integers in r which are passed from $I[j2^{2^{i+2}\sqrt{\log n}},(j+1)2^{2^{i+2}\sqrt{\log n}}-1]$, $0\le j<n/2^{2^{i+2}\sqrt{\log n}}$, at level 0 are stored in

$$A\left[j\frac{2^{2^{i+2}\sqrt{\log n}}}{2^{(2^{(i+1)}-4)\sqrt{\log n}}},(j+1)\frac{2^{2^{i+2}\sqrt{\log n}}}{2^{(2^{(i+1)}-4)\sqrt{\log n}}}-1\right],\qquad 0\le j<n/2^{2^{i+2}\sqrt{\log n}}.$$

Note that although all integers in p can be stored in A there may be many blank cells in A with no integers stored in them. Also integers passed from $I[j2^{2^{i+2}\sqrt{\log n}},(j+1)2^{2^{i+2}\sqrt{\log n}}-1]$ are now in consecutive positions in A and there are at least $2^{4\sqrt{\log n}}$ of them. Integers in each I-problem in q can be stored in an array of size S. We say that the G-problem q has size S.

For each I-problem r in a G-problem of size R we store integers in r in an array A of size R and execute s stages, where s is the minimum integer satisfying $2^{2^{s+1}\sqrt{\log n}}\ge R$. In the ith stage, $0\le i<s$, we sort integers in $A[j2^{2^{i+2}\sqrt{\log n}},(j+1)2^{2^{i+2}\sqrt{\log n}}-1]$, $0\le j<R/2^{2^{i+2}\sqrt{\log n}}$. As we said before, integers are grouped by the sorting and if the number of integers of the same revealed bits is at least $2^{4\sqrt{\log n}}$, they are sent to the next level. (They form an I-problem in a new G-problem in the next level.) We shall use the algorithm of Theorem 2.4 to do the sorting (the details are to be explained below) and therefore the time expended in the ith stage is $c2^i\sqrt{\log n}$ for a constant c.

Our sorting algorithm can be summarized in the following coding:

$Sort2(A,l,g)$ /*Sort on a set of integers at level l in the g-th G-problem*/
{
  for ($i=1$; $i\le s$; $i++$) /* $s=\min\{t|2^{2^{t+2}\sqrt{\log n}}\ge|A|\}$*/
  {
    Divide integers in A to $|A|/2^{2^{i+2}\sqrt{\log n}}$ groups and sort every group in parallel on $[\sqrt{\log n}-l-1,\sqrt{\log n}-1]$ bits;
    if sets S of integers with the same value on bits $[\sqrt{\log n}-l-1,\sqrt{\log n}-1]$ are detected, do in parallel
    {

if $|S| \geq 2^{4\sqrt{\log n}}$, { remove S from A, replace it with a dummy and call $Sort2(S, l+1, g \cdot \log\log n + i - 1)$; }
}
}
}

Note that in $Sort2$ when several sets S's with the same value in bits $[\sqrt{\log n} - l - 1, \sqrt{\log n} - 1]$ are detected in different groups of A and all these sets have size no less than $2^{4\sqrt{\log n}}$, these sets will be moved into one I-problem of a G-problem. The fact that they are moved into the same I-problem is affected by placing them in the same array.

We say that a G-problem (I-problem) p is solved if integers in each I problem at all levels generated from p are sorted. The time complexity for solving a G-problem p can now be expressed as follows. Assume that the size of p is S, p is at level l, and the time complexity of solving p is $f(S, l)$, and let $t = \min\{i | 2^{2^{i+2}\sqrt{\log n}} \geq S\}$. We have

$$f(S,l) = \max\{\max_{i=1}^{t} c2^i\sqrt{\log n} + f(S/2^{(2^{(i+1)}-4)\sqrt{\log n}}, l+1), \\ c\sqrt{\log n} + f(S, l+1)\},$$
$$f(S, \sqrt{\log n}) = 0.$$

Thus the time complexity for solving the G-problem at level 0 is $f(n, 0) = O(\log n)$.

Now let us consider the operation complexity for solving a G-problem. Suppose the integers in an I-problem p is stored in array A. Suppose at the current stage (stage i) p has S integers and dummies. (Note that many integers may have already passed to the next level at earlier stages.) S may be much smaller than the size of A. However, integers are not scattered around in A. Instead, they are stored as segments with each segment containing at least $2^{4\sqrt{\log n}}$ integers stored in consecutive memory locations if the segment does not contain a dummy. Thus in the current stage (stage i) we can pack the integers in $A[j2^{2^{i+2}\sqrt{\log n}}, (j+1)2^{2^{i+2}\sqrt{\log n}} - 1]$, $j = 0, 1, 2, \ldots$, into consecutive memory locations using $O(2^i\sqrt{\log n})$ time and $O(S + d2^i\sqrt{\log n})$ operations, where d is the number of dummies. Since the number of dummies is a fraction of the total number of integers, the total number of operations is linear. After we packed integers we use the corollary to Theorem 2.4 to sort them. The sorting takes $O(2^i\sqrt{\log n})$ time and $O(S2^i)$ operations. This is to say that if an integer a stayed in p until stage i it incurs $O(2^i)$ operations. However, since a is passed to the I-problem q at the next level at stage i, the size of q is the size of p divided by $2^{(2^{(i+1)}-4)\sqrt{\log n}}$ if $i > 0$ and is at most the same as that of p if $i = 0$. Assume that the size of p is S, p is at level l, and the number of operations incurred by an integer a in p is $g(S, l)$. Then

$$g(S,l) = \max\{\max_{i=1}^{t} c2^i + g(S/2^{(2^{(i+1)}-4)\sqrt{\log n}}, l+1), \\ c + g(S, l+1)\},$$
$$g(S, \sqrt{\log n}) = 0.$$

Thus the number of operations incurred by each integer in the G-problem at level 0 is $g(n, 0) = O(\sqrt{\log n})$. Thus the operation complexity for solving the G-problem at level 0 is $O(n\sqrt{\log n})$.

Below we will discuss how to have all the input integers sorted after we solve the G-problem at level 0.

How many G-problems will be created in the execution of our algorithm? Consider a G-problem p at level l with size S. p executes $\log \frac{\log S}{\sqrt{\log n}} - 1$ stages which generates $\log \frac{\log S}{\sqrt{\log n}} - 1$ G-problems at the next level. The ith G-problem, $i > 0$, generated has size $S/2^{(2^{i+1}-4)\sqrt{\log n}}$. Let the number of G-problems at all levels which are generated from p be $h(S, l)$. We have

$$h(S,l) = h(S,l+1) + 1 + \sum_{i=1}^{\log((\log S)/(\sqrt{\log n}))-2} (h(S/2^{(2^{i+1}-4)\sqrt{\log n}}, l+1) + 1),$$

$$h(S, \sqrt{\log n} - 1) = 1.$$

$h(n, 0)$ is the total number of G-problems generated in our algorithm. It is not difficult to see that $h(n, 0) > 2^{\sqrt{\log n}}$. To evaluate the above formula we enlarge it and obtain the following formula:

$$h(S,l) \le 4 \sum_{i=0}^{(\log S)/(\sqrt{\log n})} h(S/2^{i\sqrt{\log n}}, l+1),$$

$$h(S, \sqrt{\log n} - 1) = 1,$$

which can be rewritten as

$$C(j,l) \le 4 \sum_{k=0}^{j} C(k, l+1),$$

$$C(j, \sqrt{\log n} - 1) = 1,$$

where $C(j, l) = h(2^{j\sqrt{\log n}}, l)$.

The total number of groups generated by our algorithm is $h(n, 0)$, which is

$$\begin{aligned} h(n,0) &= C(\sqrt{\log n}, 0) \\ &= 4 \sum_{k_1=0}^{\sqrt{\log n}} C(k_1, 1) \\ &= 4^{\sqrt{\log n}} \sum_{k_1=0}^{\sqrt{\log n}} \sum_{k_2=0}^{k_1} \sum_{k_3=0}^{k_2} \cdots \sum_{k_{\sqrt{\log n}}=0}^{k_{\sqrt{\log n}-1}} 1 \\ &\le 2^{\delta\sqrt{\log n}}, \quad \text{where } \delta < 3.5. \end{aligned}$$

Therefore, there are no more than $2^{\delta\sqrt{\log n}}$ G-problems generated.

We attach a tag to each integer and dummy to indicate which G-problem it is in. Although the tag can be implemented by an $O(\sqrt{\log n})$-bit integer because there are only $2^{\delta\sqrt{\log n}}$ G-problems, to facilitate the computation of the tag when an integer is passed from a G-problem at level i to another G-problem at level $i + 1$, we use an $O(\sqrt{\log n} \log\log\log n)$-bit integer for a tag. If an integer a is in the G-problem p at level i and its tag is t, and it is passed at stage s of level i to another G-problem p_1 at level $i + 1$, we form the new tag for a by appending $O(\log\log\log n)$ bits indicating s

to t. Dummies created to represent integers passed to the next level inherit the new tag of the integers after they are passed. Thus all integers in a G-problem have the same tag. Dummies in a G-problem may have different tags because they represent integers passed to next level at different stages.

Now consider an I-problem p in a G-problem at level i. As the sorting in p proceeds, integers of the same value in bits $[\sqrt{\log n}-i-1 : \sqrt{\log n}-1]$ are grouped in each sorted sequence in p. If the number of integers of the same value $a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ is no less than $2^{4\sqrt{\log n}}$, these integers will be passed down to a G-problem at the next level. A dummy will be created in p to represent these integers. The dummy has the tag which is the same as the new tag those integers obtained after they passed to the next level. Integers with value $a^{[\sqrt{\log n}-i-1:\sqrt{\log n}-1]}$ left in p will keep grouping as the sorting proceeds. When a new group contains at least $2^{4\sqrt{\log n}}$ integers they will be passed to another G-problem and another dummy will be created for them. All integers in the G-problem which are passed down at a stage form a new G-problem. When integers and dummies of the same value in bits $[\sqrt{\log n}-i-1 : \sqrt{\log n}-1]$ are grouped together in p, we assume that dummies are smaller than integers and this allows integers to be grouped with integers to form new groups with more than $2^{4\sqrt{\log n}}$ integers. When dummies with the same bit values and the same tag are grouped together (they represent integers passed to the same I-problem in a G-problem), all of them but one can be removed. But dummies with different bit values or different tags cannot cancel each other.

Now suppose that integers and dummies in each I-problems are sorted and occupy consecutive memory locations. We need to put all integers in a G-problem together to form a sorted sequence. If we accomplish this, we will have only $2^{\delta\sqrt{\log n}}$ sorted sequences left because there are only $2^{\delta\sqrt{\log n}}$ G-problems. We first reduce the number of bits used for a tag to $O(\sqrt{\log n})$ (remember we were using $O(\sqrt{\log n}\log\log\log n)$ bits). This is accomplished easily by sorting all input integers in the input array by their tags. Here we are sorting on $O(\sqrt{\log n}\log\log\log n)$ bits and it can be done by Theorem 3.1. Thereafter we reduce the value for a tag to be within $\{0, 1, \ldots, 2^{\delta\sqrt{\log n}}-1\}$ by eliminating unused values. Thus the number of bits used for a tag becomes $\delta\sqrt{\log n}$. Then we sort integer and dummies in an I-problem p at level i by their tag value. Note that integers in p have the same tag value while dummies may have different tags. Note also that integers and dummies in p were sorted by bits $[\sqrt{\log n}-i-1 : \sqrt{\log n}-1]$. By sorting on their tags we arrange dummies with the same tag value into consecutive memory locations while dummies with the same tag value are still in sorted order by their value of bits $[\sqrt{\log n}-i-1 : \sqrt{\log n}-1]$. We then add a pointer for the first and last integers and each dummy in p to point to the dummy at level $i-1$, which represents the integers in p when they are passed from level $i-1$ to level i. We now view that all integers and dummies being stored in a big array A (of course many cells A are blanks). We allocate $2^{\delta\sqrt{\log n}}$ arrays $A_0, A_1, \ldots, A_{2^{\delta\sqrt{\log n}}-1}$; each of them has the same size as A. Array A_i is used to store integers in ith G-problem (integers tagged with i). For each integer a in A, if a is in $A[k]$ and tagged with i, we now move a to $A_i[k]$ in constant time. If a is the first or last integer in an I-problem (therefore, a has a pointer pointing to $A[j]$, which is a dummy at the lower numbered level), a moves this pointer to A_i and points to $A_i[j]$. For each dummy d in $A[k]$ we make $2^{\delta\sqrt{\log n}}$ copies of it and put one copy in the kth cell of each A_i, $0 \le i < 2^{\delta\sqrt{\log n}}$. If the pointer of d (which points to a dummy at the previous level) points to $A[j]$, we copy the pointer to each A_i and make it point to $A_i[j]$, $0 \le i < 2^{4\sqrt{\log n}}$. Note that because each dummy is created at a level for

at least $2^{4\sqrt{\log n}}$ integers, we can allocate $2^{\delta\sqrt{\log n}}$ processors for each dummy. Up to now we have separated integers in different G-problems into different arrays. The integers and dummies in each array A_i together with their pointers form a tree. By using pointer jumping along the Euler tour of the tree [27] we obtain a sorted list of integers in each G-problem. Consider the pointers allocated to each I-problem. If an I-problem does not contain a dummy (i.e., no integers in the I-problem are passed to the next level), then the I-problem has at least $2^{4\sqrt{\log n}}$ integers. We need to use two pointers, one at the beginning of the integers and one at the end of the integers, for linking this I-problem with other I-problems in the same G-problem. If the I-problem has a dummy, we can always allocate three pointers per dummy. Then one pointer is used for the dummy and the other two pointers can be used for the integers in the I-problem. Suppose there are D dummies then the number of pointers used for all I-problems is $O(D + n/2^{4\sqrt{\log n}})$ and the time used is $O(\log n)$.

Now we have a collection C of $\leq 2^{\delta\sqrt{\log n}}$ sorted sequences of integers. We need to merge all sequences in C into one sorted sequence. We first sample every $2^{\delta\sqrt{\log n}}$th integer from each sequence $s \in C$. If a sequence has no more than $2^{\delta\sqrt{\log n}}$ integers, we sample its first and last integer. We obtain a collection C' of $|C|$ sampled sequences. There are no more than $2n/2^{\delta\sqrt{\log n}}$ sampled integers. We merge every pair of sampled sequences by first making $2^{\delta\sqrt{\log n}}$ copies of each sampled sequence and merge the corresponding pair. If integer a is in a sampled sequence s which is merged with every other sampled sequence, then a knows its rank in every sampled sequence. By summing these ranks a knows its rank in all sampled integers. Therefore we have sorted sampled integers into one sequence q. We then make $2^{\delta\sqrt{\log n}}$ copies of q and merge q with each sequence in C. This merge divides the original merge problem (of merging sequence in C) into $|q| + 1$ submerge problems with each submerge problem merging $|C|$ subsequences of no more than $2^{\delta\sqrt{\log n}}$ integers (one subsequence coming from one sequence in C). Since the total number of integers in each submerge problem is no more than $|C|2^{\delta\sqrt{\log n}} \leq 2^{7\sqrt{\log n}}$ we can use comparison sorting [1, 10] to sort integers in each submerge problem.

Let us estimate the space complexity. Each I-problem formed at stage 0 or 1 at each level requires $O(n)$ space. The space needed for I-problems formed in later stages is geometrically decreasing. Thus $O(mn\sqrt{\log n})$ space is used for solving all G-problems. We then allocate $2^{\delta\sqrt{\log n}}$ arrays of A_i's. Thus the total space used is $O(mn\sqrt{\log n}2^{\delta\sqrt{\log n}})$. However, space can be reduced to $O(nm^{\epsilon})$ by using radix sorting.

THEOREM 4.2. *n integers in the range $\{0, 1, \ldots, m-1\}$ ($m \geq n$) can be sorted in $O(\log n)$ time and $O(nm^{\epsilon})$ space with $O(n\sqrt{\log n})$ operations and $O(\log(m+n))$ word length on the EREW PRAM.*

To sort n integers in the range $\{0, 1, 2, \ldots, m-1\}$ with word length $k \log m$ bits, we modify our algorithm to sort $O(\log m/\sqrt{\log n/k})$ bits in each level, and in the ith stage we sort every $2^{2^{i+2}\sqrt{k\log n}}$ integers. This will give $O(\log n)$ time and $O(n\sqrt{\log n/k})$ operations.

THEOREM 4.3. *n integers in the range $\{0, 1, \ldots, m-1\}$ can be sorted in $O(\log n)$ time with $O(n\sqrt{\log n/k})$ operations and $O(k\log(m+n))$ word length on the EREW PRAM.*

5. Conclusions. We presented EREW integer sorting algorithms which outperform parallel comparison sorting. There are several open questions. An immediate one is to further improve operation complexity. Another open problem is to reduce

the space complexity of the algorithm in Theorem 4.2 to linear. Applications of our algorithm to other problems will also be interesting where the advantage of parallel integer sorting over parallel comparison sorting can be made use of.

Appendix A. Implementation of the algorithm in section 3. The subtlety of our algorithm is in how to do grouping, where to place dummy elements, and how to maintain the space complexity within $O(n)$. Grouping should be done with linked list contraction. However, we cannot apply any existing linked list contraction algorithms directly to obtain $O(\sqrt{\log n})$ time and $O(n)$ operations for a stage because we need an algorithm to do partial linked list contraction. The dummy elements generated need to be placed within $O(n)$ space so that processors can be allocated to them. For the space complexity consideration, after grouping we need sort integers within each group and this may seem to require that we place the integers in a group in consecutive memory locations. If we allocate $O(n)$ memory for placing all integers such that all integers in a group occupy consecutive memory locations, then we would need $O(\log n)$ time while we can expend only $O(\sqrt{\log n})$ time in each stage. What we could do instead is to allocate a two dimensional array with $2^{5\sqrt{\log n}}$ rows and n columns. We place the linked lists in the first row. For each group, we could put the integers in the group in the jth column of the array if the first integer in the group is in column j. This scheme facilitates sorting. The only shortcoming of the scheme is that it uses more than $O(n)$ space. We give schemes from which all the problems mentioned above can be resolved.

For implementation purposes we reveal $\sqrt{\log n}$ bits in each stage except the last stage which reveals $10\sqrt{\log n}$ bits. A linked list is very short if it contains no more than $2^{2\sqrt{\log n}}$ integers, is short if it contains less than $2^{6\sqrt{\log n}}$ integers, and is long if it contains at least $2^{6\sqrt{\log n}}$ integers. A group on a short linked list contains all integers in the list. A group on a long linked list contains at least $2^{6\sqrt{\log n}}$ but fewer than $2^{7\sqrt{\log n}}$ integers.

A.1. Blocked linked list. We modify our linked list construction. Instead of linking elements (integers) from memory location to memory location, we require that every $2^{2\sqrt{\log n}}$ elements in a linked list occupy consecutive memory locations and that the first element among these $2^{2\sqrt{\log n}}$ elements is at a memory cell j where $j \bmod 2^{2\sqrt{\log n}} = 0$. We call such $2^{2\sqrt{\log n}}$ elements a block. Thus if we walk down the linked list, we visit $2^{2\sqrt{\log n}}$ consecutive memory locations, then follow the pointer to another memory location, then visit another $2^{2\sqrt{\log n}}$ consecutive memory locations, and so on. We call such a linked list a blocked linked list. For all the linked lists split we maintain this property (except for the linked lists split at the end of the last stage). This property facilitates linked list contraction. The condition on memory cell $j \bmod 2^{2\sqrt{\log n}} = 0$ ensures that processors can be allocated to the elements in the linked lists. Because we use $n/\sqrt{\log n}$ processors, one processor is allocated for $\sqrt{\log n}$ elements or integers.

A.2. Linked list contraction. Now consider grouping. Because linked lists are blocked, the linked list contraction for the bottom $2^{2\sqrt{\log n}}$ elements are automatically done. That is, for a linked list l_1 of length S, we can view it as being already contracted to a linked list l_2 of length $S/2^{2\sqrt{\log n}}$. For the further contraction of l_2, we can allocate one processor for each node in l_2.

Therefore we are now facing the following linked list contraction problem: Contract a linked list in $O(\sqrt{\log n})$ time and $O(n\sqrt{\log n})$ operations (note that here we can assign one processor to each node in the linked list) such that every S element on

the linked list is contracted to a single node, where $2^{c\sqrt{\log n}} \leq S \leq 2^{(c+1)\sqrt{\log n}}$ and c is a constant.

This is a very special linked list contraction problem and no known linked list contraction algorithm can solve it directly. We use the following scheme to solve this problem.

We will apply pointer jumping [30] to contract the linked list. For a linked list of n nodes pointer jumping takes $O(\log n)$ time and $O(n \log n)$ operations to contract the whole list. Since we are going to contract every S element on the linked list we should break the linked list into pieces such that each piece has S nodes of the linked list and then we apply pointer jumping on each piece. Then the pointer jumping will take $O(\log S) = O(\sqrt{\log n})$ time and $O(n \log S) = O(n\sqrt{\log n})$ operations. This scheme is a perfect one except that we do not have a scheme to break the linked list into pieces of S nodes each.

The current best parallel algorithm [17, 18, 14] for breaking the linked list into pieces (symmetry breaking algorithm) can break the linked list into pieces in $O(\log d)$ time and uses linear operations such that each piece has at least two nodes and at most $\log^{(d)} n$ nodes, where d is a constant. Here we are guaranteed that the linked list will be broken up into pieces. But we are not guaranteed that each piece contains between $2^{c\sqrt{\log n}}$ and $2^{(c+1)\sqrt{\log n}}$ nodes.

Should each broken piece contain about the same number of nodes, say T nodes, then we could apply pointer jumping on each piece for $O(\log T)$ time to contract each piece into a single node. The original linked list L_1 of n nodes is thus being contracted into a linked list L_2 of n/T nodes. Now we could apply symmetry breaking on L_2 and then pointer jumping on the pieces of the linked list after symmetry breaking. We could repeat this symmetry breaking and pointer jumping process for $\log S / \log T$ times and would have finished linked list contraction in $O(\log S)$ time and $O(n \log S)$ operations.

The problem now is that each broken piece of the linked list can contain as few as two nodes and as many as $\log^{(c)} n$ nodes. Thus, the shortest piece takes constant time to contract and the longest piece takes $O(\log^{(c+1)} n)$ time to contract when pointer jumping is applied. If we let processors working on the shortest piece wait until after they finish pointer jumping for the processors working on the longest piece, then the progress of the algorithm will not be fast enough for us to get the $O(\sqrt{\log n})$ time for linked list contraction.

Thus the strategy we use is that when the processors working on a short piece P finish pointer jumping they check whether the neighboring pieces (the previous piece and the next piece) have finished contraction (pointer jumping). If both of its neighboring pieces have not finished contraction, P will wait. If one of P's neighbors N has already finished contraction, then the node P contracts into and the node N contracts into are linked on a linked list and then symmetry breaking and pointer jumping can applied to this linked list.

Since no two consecutive nodes can be in a wait state at the same time, every three nodes are contracted into at most two nodes in a step. Thus the whole contraction process takes $O(\sqrt{\log n})$ time and $O(n\sqrt{\log n})$ operations.

A.3. Coping with dummy elements. We first show how we handle short linked lists. The number of bits we revealed for each integer is $\leq \log n - 10\sqrt{\log n}$ before the last stage. Thus we can have at most $n/2^{10\sqrt{\log n}}$ short linked lists before the last stage. Each short linked list can be indexed by an integer range from 0 to $n/2^{10\sqrt{\log n}}$. We can allocate an array A of size n to store only short linked lists.

$A[i2^{7\sqrt{\log n}} \dots (i-1)2^{7\sqrt{\log n}} - 1]$ is reserved for the short linked list indexed i. Once we have a short linked list we sort it with parallel comparison sorting on the whole integer (not just the revealed bits). This entails $O(\sqrt{\log n})$ time for each short list and $O(\sqrt{\log n})$ operations for each integer, and therefore it will not destroy the complexity analysis for the whole algorithm. Thereafter we consider only long linked lists.

Initially we put all n input integers into a linked list, and therefore we start with a long linked list. To facilitate the later processing we add n dummy elements to the initial linked list. We put input integers and dummies alternately in the initial linked list as $a_0, d, a_1, d, a_2, d, \dots, d, a_{n-2}, d, a_{n-1}$, where d is a dummy. Therefore for every t consecutive integers on the linked list we have t dummies. These dummies will later be used for keeping the blocking property of the linked list and for representing missing patterns in a group.

Initially we have one dummy for every two consecutive elements on the linked list. We use $1/2$ to represent this ratio. After several stages this ratio will become smaller. We assume that at the current stage the ratio is $1/b$, i.e., there is a dummy in every b elements on the linked list.

After sorting integers in a group, integers with the same revealed bits (bit pattern) are consecutive on the linked list. However, the number of integers with the same revealed bits may not be a multiple of $2^{2\sqrt{\log n}}$. To maintain the blocking property of the linked list, we make use of dummy elements so that the number of integers and dummies within each group with the same revealed bit pattern become a multiple of $2^{2\sqrt{\log n}}$. For a group G of S integers in a long linked list, we split it into $2^{\sqrt{\log n}}$ linked lists. Let a linked list L split from G have T integers. We put $\lceil T/(2b) \rceil + 2 \cdot 2^{2\sqrt{\log n}} - (T + \lceil T/(2b) \rceil) \% 2^{2\sqrt{\log n}}$ dummies into L, where % is the modulo operation. Thus the total number of integers and dummies in L is a multiple of $2^{2\sqrt{\log n}}$. Summing over all split linked lists, the total dummies we used is $< S/(2b) + 3 \cdot 2^{3\sqrt{\log n}}$. Thus if $S/b \geq S/(2b) + 3 \cdot 2^{3\sqrt{\log n}}$, i.e., $S/(2b) \geq 3 \cdot 2^{3\sqrt{\log n}}$, then we have enough dummies. The ratio of the dummies to the elements in the split linked list is now $1/(2b)$. Thus through one stage the ratio is reduced by at most half. Since there are $\sqrt{\log n}$ stages the smallest ratio we have is $1/2^{\sqrt{\log n}}$. Because $S \geq 2^{6\sqrt{\log n}}$ we can hold $S/(2b) \geq 3 \cdot 2^{3\sqrt{\log n}}$.

A.4. Sorting each group in linear space. We now show how to sort each group in linear space. For a short linked list we reveal all remaining bits of the integers on the list and then sort these integers using comparison sorting [1, 10]. Since there are at most $n/2^{10\sqrt{\log n}}$ very short linked lists (at the beginning of the last stage), the total number of operations involved in sorting short linked lists is $O((n/2^{10\sqrt{\log n}}) \cdot 2^{7\sqrt{\log n}} \cdot \sqrt{\log n}) \leq O(n\sqrt{\log n}/2^{3\sqrt{\log n}})$.

The sorting of integers in a group on a long linked list is done directly on integers on the linked list. For the purpose of sorting we may assume that the number of integers in a group is a power of 2. Otherwise we simply add some dummy elements to make it a power of 2 and store these dummy elements along elements of the linked list so that each memory location on the linked list stores at most two elements. (This can be realized by an array of two rows.) Note that we use a sorting network to accomplish the sorting. In order to sort integers on a linked list, an integer on the linked list has to know the address of the integers that it will compare itself with at each level of the sorting network. Because the AKS sorting network has $O(\sqrt{\log n})$ levels (because we are sorting at most $2^{7\sqrt{\log n}}$ integers in the group), and because in our sorting algorithm we pack $O(\sqrt{\log n})$ integers into one word, each word on the

linked list has to know $O(\sqrt{\log n})$ addresses (one address for each level). Let a_i be the word at memory address $d(a_i)$ (memory address is an $O(\log n)$ bit integer). If a_i needs to be compared with a_j at the first level of the sorting network, we need route address $d(a_i)$ to $d(a_j)$ and address $d(a_j)$ to $d(a_i)$. What we need here is a permutation of the memory addresses, and the permutation is a fixed one (known in advance). Thus we can route the memory address through the permutation network given in section 2. Note that a butterfly network has such a "regular" structure that each node (word) on the linked list can find the address of the nodes it needs to switch with at all levels of the network through pointer jumping [30]. The regularity we are talking about here is that each node needs to know the nodes which are 2^i, $i = 0, 1, 2, \ldots$, positions away from it and this can be done by pointer jumping. The AKS sorting network, on the other hand, does not have this property. Pointer jumping allows nodes on a linked list to meet with nodes which are 2^i positions (distance) away along the linked list, $i = 0, 1, 2, \ldots$. Let S be the number of integers in the group. Then there are $S/\sqrt{\log n}$ words. Each word has an address to be permuted and the permutation takes $O(\sqrt{\log n})$ time. Thus one permutation uses $O(S)$ operations for the group ($O(n)$ operations for all linked lists). However, we need to do $O(\sqrt{\log n})$ permutations for $O(\sqrt{\log n})$ addresses because the AKS sorting network has that many levels. And we have to do all these permutations in $O(S)$ operations for the group ($O(n)$ operations for all linked lists). In order to solve this problem we use a modified version of our nonconservative sorting algorithm. In each node of the sorting network we compare $\sqrt{\log n}$ words with another $\sqrt{\log n}$ words in parallel instead of comparing just two words. That is, we use $\sqrt{\log n} \cdot \sqrt{\log n} = \log n$ columns in the column sort. By the blocking property of the linked list these $\sqrt{\log n}$ words occupy consecutive memory addresses. Thus for each $\sqrt{\log n}$ words we need permute only the address of the first word. When the permutation is done, the addresses of the other words can be figured out. Therefore the number of operations for each permutation is reduced to $O(S/\sqrt{\log n})$ ($O(n/\sqrt{\log n})$ for all linked lists). For all $O(\sqrt{\log n})$ permutations the total number of operations is $O(S)$ ($O(n)$ for all linked lists). In order to keep $O(\sqrt{\log n})$ time for all these $O(\sqrt{\log n})$ permutations, we do them in parallel. For the $\sqrt{\log n}$ words, we let the ith word participate in the ith permutation.

The permutations performed among integers after each sort on columns can be done by routing the integers through the permutation networks given in section 2. Again, since the butterfly network has such a regular connection structure, a word can find the word it will switch with through pointer jumping. We therefore showed that sorting can be done for integers on the linked list.

After sorting we need to move integers to the sorted position. The problem here is that when we are sorting we are moving $\sqrt{\log n}$ bits for each integer through the AKS sorting network. In order for each integer to know the address after sorting the address which is a $\log n$ bit integer should be known to the integer. Note that such an address has $\log n$ bits and we cannot move it through the sorting network (otherwise it will incur $O(n\sqrt{\log n})$ operations *in one stage*). We cannot move integers through the sorting network either because each integer has about $\log n$ bits (it has fewer bits in later stages, though) instead of the $\sqrt{\log n}$ revealed bits.

Since each block has $2^{2\sqrt{\log n}}$ integers we modify the sorting within each group first to sort integers in each block and then to sort the whole group. Sorting integers in each block can be done by Theorem 2.4 and the relative address of the sorted position of an integer a has only $2\sqrt{\log n}$ bits. Therefore the *relative address* can be transferred back to a through the AKS sorting network (as we did in the above

paragraph). Then a can use this address to index into the sorted position. After each block is sorted the integers with the same revealed bits in a block are in consecutive memory locations. Because each block has $2^{2\sqrt{\log n}}$ integers and there are only $2^{\sqrt{\log n}}$ revealed bit patterns, for each bit pattern p we could use a representative integer a_p and let a_p find sorted locations (after the sort for the group) for all integers in the block with bit pattern p. Since the total number of representatives is a fraction of the total number of integers, the representatives can find the sorted positions by routing them through the sorting network.

Before the beginning of the last stage (which reveals $10\sqrt{\log n}$ bits) we use linked list ranking [5] to move all integers in a linked list into consecutive memory locations. Therefore in the last stage the integers to be sorted are based on arrays instead of a linked list.

REFERENCES

[1] M. Ajtia, J. Komlós, and E. Szemerédi, *Sorting in $c \log n$ parallel steps*, Combinatorica, 3 (1983), pp. 1–19.

[2] S. Albers and T. Hagerup, *Improved parallel integer sorting without concurrent writing*, Inform. and Comput., 136 (1997), pp. 25–51.

[3] A. Andersson, *Fast deterministic sorting and searching in linear space*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 135–141.

[4] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman, *Sorting in linear time?* in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 427–436.

[5] R. Anderson and G. Miller, *Deterministic parallel list ranking*, Algorithmica, 6 (1991), pp. 859–868.

[6] P. Beame and J. Hastad, *Optimal bounds for decision problems on the CRCW PRAM*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, 1987, pp. 83–93.

[7] V. E. Benes, *On rearrangeable three-stage connecting networks*, Bell Syst. Tech. J., 41 (1962), pp. 1481–1492.

[8] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[9] P. C. P. Bhatt, K. Diks, T. Hagerup, V. C. Prasad, T. Radzik, and S. Saxena, *Improved deterministic parallel integer sorting*, Inform. and Comput., 94 (1991), pp. 29–47.

[10] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.

[11] S. Cook, C. Dwork, and R. Reischuk, *Upper and lower time bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput., 15 (1986), pp. 87–97.

[12] A. Dessmark and A. Lingas, *Improved bounds for integer sorting in the EREW PRAM model*, J. Parallel Distrib. Comput., 48 (1998), pp. 64–70.

[13] M. L. Fredman and D. E. Willard, *Surpassing the information theoretic bound with fusion trees*, J. Comput. System Sci., 47 (1994), pp. 424–436.

[14] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon, *Parallel symmetry-breaking in sparse graphs*, SIAM J. Discrete Math., 1 (1988), pp. 434–446.

[15] T. Hagerup, *Towards optimal parallel bucket sorting*, Inform. and Comput., 75 (1987), pp. 39–51.

[16] T. Hagerup and H. Shen, *Improved nonconservative sequential and parallel integer sorting*, Inform. Process. Lett., 36 (1990), pp. 57–63.

[17] Y. Han, *Matching partition a linked list and its optimization*, in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'89), Santa Fe, NM, 1989, pp. 246–253.

[18] Y. Han, *An optimal linked list prefix algorithm on a local memory computer*, in Proceedings of the 17th Annual ACM Conference on Computer Science (CSC'89), Louisville, KY, 1989, pp. 278–286.

[19] Y. Han and X. Shen, *Conservative algorithms for parallel and sequential integer sorting*, in Proceedings of the International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 959, Springer-Verlag, Berlin, 1995, pp. 324–333.

[20] J. JÁJÁ, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
[21] D. KIRKPATRICK AND S. REISCH, *Upper bounds for sorting integers on random access machines*, Theoret. Comput. Sci., 28 (1984), pp. 263–276.
[22] C. P. KRUSKAL, L. RUDOLPH, AND M. SNIR, *A complexity theory of efficient parallel algorithms*, Theoret. Comput. Sci., 71, pp. 95–132.
[23] T. LEIGHTON, *Tight bounds on the complexity of parallel sorting*, IEEE Trans. Comput., 34 (1985), pp. 344–354.
[24] S. RAJASEKARAN AND J. H. REIF, *Optimal and sublogarithmic time randomized parallel sorting algorithms*, SIAM J. Comput., 18 (1989), pp. 594–607.
[25] S. RAJASEKARAN AND S. SEN, *On parallel integer sorting*, Acta Inform., 29 (1992), pp. 1–15.
[26] R. RAMAN, *The power of collision: Randomized parallel algorithms for chaining and integer sorting*, in Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 472, Springer-Verlag, Berlin, pp. 161–175.
[27] R. E. TARJAN AND U. VISHKIN, *Finding biconnected components and computing tree functions in logarithmic parallel time*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 12–20.
[28] R. VAIDYANATHAN, C. R. P. HARTMANN, AND P. K. VARSHNEY, *Towards optimal parallel radix sorting*, in Proceedings of the 7th International Parallel Processing Symposium, Newport Beach, CA, 1993, IEEE, pp. 193–197.
[29] R. A. WAGNER AND Y. HAN, *Parallel algorithms for bucket sorting and the data dependent prefix problem*, in Proceedings of the International Conference on Parallel Processing, University Park, PA, 1986, IEEE, pp. 924–930.
[30] J. C. WYLLIE, *The Complexity of Parallel Computation*, Technical Report 79-387, Department of Computer Science, Cornell University, Ithaca, NY, 1979.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1879–1895

A RANDOMIZED TIME-WORK OPTIMAL PARALLEL ALGORITHM FOR FINDING A MINIMUM SPANNING FOREST*

SETH PETTIE† AND VIJAYA RAMACHANDRAN†

Abstract. We present a randomized algorithm to find a minimum spanning forest (MSF) in an undirected graph. With high probability, the algorithm runs in logarithmic time and linear work on an exclusive read exclusive write (EREW) PRAM. This result is optimal w.r.t. both work and parallel time, and is the first provably optimal parallel algorithm for this problem under both measures. We also give a simple, general processor allocation scheme for tree-like computations.

Key words. parallel algorithm, minimum spanning tree, optimal algorithm, EREW PRAM

AMS subject classifications. 05C85, 68R10, 68Q85

PII. S0097539700371065

1. Introduction. We present a randomized parallel algorithm to find a minimum spanning forest (MSF) in an edge-weighted, undirected graph. On an exclusive read exclusive write (EREW) PRAM [KR90] our algorithm runs in expected logarithmic time and linear work in the size of the input; these bounds also hold with high probability in the size of the input. This result is optimal w.r.t. both work and parallel time and is the first provably optimal parallel algorithm for this problem under both measures.

Here is a brief summary of related results. Following the linear-time sequential MSF algorithm of Karger, Klein, and Tarjan [KKT95] (and building on it) came linear-work parallel MSF algorithms for the concurrent read concurrent write (CRCW) PRAM [CKT94, CKT96] and the EREW PRAM [PR97]. The best CRCW PRAM algorithm known to date [CKT96] runs in logarithmic time and linear work, but the time bound is not known to be optimal. The best EREW PRAM algorithm known prior to our work is the result of Poon and Ramachandran which runs in $O(\log n \log\log n 2^{\log^* n})$ time and linear work. All of these algorithms are randomized. Recently Chong, Han, and Lam [CHL01] presented a deterministic EREW PRAM algorithm for MSF, which runs in logarithmic time with a linear number of processors, and hence with work $O((m+n)\log n)$, where n and m are the number of vertices and edges in the input graph. It was observed by Poon and Ramachandran [PR98] that the algorithm in [PR97] could be sped up to run in $O(\log n \cdot 2^{\log^* n})$ time and linear work by using the algorithm in [CHL01] as a subroutine (and by modifying the "contract" subroutine in [PR97]).

In this paper we improve on the running time of the algorithm in [PR97, PR98] to $O(\log n)$, which is the best possible, and we improve on the algorithm in [CKT96] by achieving the logarithmic time bound on the less powerful EREW PRAM.

*Received by the editors April 19, 2000; accepted for publication (in revised form) March 20, 2002; published electronically October 18, 2002. This work was supported by Texas Advanced Research Program grant 003658-0029-1999. A preliminary version of this paper appeared in *Randomization, Approximation, and Combinatorial Optimization (Berkeley, CA,* 1999*)*, Lecture Notes in Comput. Sci. 1671, Springer-Verlag, Berlin, 1999, pp. 233–244.

http://www.siam.org/journals/sicomp/31-6/37106.html

†Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (seth@cs.utexas.edu, vlr@cs.utexas.edu). The research of the first author was supported by an MCD graduate fellowship. The research of the second author was supported by NSF grant CCR-9988160.

Our algorithm has a simple 2-phase structure. It makes subroutine calls to the Chong–Han–Lam (CHL) algorithm [CHL01], which is fairly complex. But outside of these subroutine calls (which are made to the simplest version of the algorithm in [CHL01]), the steps in our algorithm are quite straightforward.

In addition to being the first time-work optimal parallel algorithm for MSF, our algorithm can be used as a simpler alternative to several other parallel algorithms:

1. For the CRCW PRAM we can replace the calls to the CHL algorithm by the (work inefficient) scheme used in [CKT96]. The resulting algorithm runs in logarithmic time and linear work but is considerably simpler than the MSF algorithm in [CKT96] and also is about twice as fast.
2. As modified for the CRCW PRAM, our algorithm is simpler than the linear-work logarithmic-time CRCW algorithm for connected components given in [Gaz91].
3. Our algorithm improves on the EREW connectivity and spanning tree algorithms in [HZ96, HZ01] since we compute a *minimum* spanning tree within the same time and work bounds. Our algorithm also is simpler than the algorithms in [HZ96, HZ01].

In the following we use the notation $S + T$ to denote the union of sets S and T, and we use $S + e$ to denote the set formed by adding the element e to the set S. We say that a result holds *with high probability (or w.h.p.) in* n if the probability that it fails to hold is less than $1/n^c$ for any constant $c > 0$.

The rest of this paper is organized as follows. In section 2 we give a high-level description of our algorithm, which works in two phases. In section 3 we provide the details of phase 1, whose main purpose is to reduce the number of vertices in the graph by at least a $(\log \log n)^2$ factor. Phase 2, given in section 4, finds the MSF of the reduced-vertex graph using a recursion-free version of the [PR97] algorithm. In sections 5 and 6 we prove the algorithm runs in logarithmic time and linear work with high probability. Section 7 gives a simple processor allocation scheme for "tree-structured" computations (a class containing our MSF algorithm) which is space-optimal. In section 8 we discuss the adaptability of our algorithm to realistic parallel models like the bulk-synchronous parallel (BSP) [Val90] and queuing shared memory models (QSM) [GMR99]. Our conclusions are given in section 9.

2. The high-level algorithm. Our algorithm is divided into two phases along the lines of the CRCW PRAM algorithm of [CKT96]. In phase 1, the algorithm reduces the number of vertices in the graph from n to n/k_0 vertices, where n is the number of vertices in the input graph, and[1] $k_0 = (\log^{(2)} n)^2$. To perform this reduction the algorithm uses the familiar recursion tree of depth $\log^* n$ [CKT94, CKT96, PR97], which gives rise to $O(2^{\log^* n})$ recursive calls; however, the time needed per invocation in our algorithm is well below $O(\log n / 2^{\log^* n})$. Thus the total time for phase 1 is $O(\log n)$. We accomplish this by requiring phase 1 to find only a subset of the MSF. By contracting this subset of the MSF we obtain a graph with $O(n/k_0)$ vertices. Phase 2 then uses an algorithm similar to the one in [PR97], but needs no recursion due to the reduced number of vertices in the graph. Thus phase 2 is able to find the MSF of the contracted graph in $O(\log n)$ time and linear work.

We assume that edge weights are distinct. As always, distinctness can be forced by ordering the vertices, then ordering identically weighted edges by their end points.

[1] We use $\log^{(r)} n$ to denote the log function iterated r times and $\log^* n$ to denote the minimum r s.t. $\log^{(r)} n \leq 1$.

The high-level algorithm is given in Figure 2.1.

High-Level(G)

(Phase 1)
$G_t :=$ For all $v \in G$, retain the lightest k_0 edges in edge-list(v)
$M :=$ Find-k-Min$(G_t, \log^* n)$
$G' :=$ Contract all edges in G appearing in M

(Phase 2)
$G_s :=$ Sample edges of G' with prob. $1/\sqrt{k_0} = 1/\log^{(2)} n$
$F_s :=$ Find-MSF(G_s)
$G_f :=$ Filter(G', F_s)
$F :=$ Find-MSF(G_f)
Return$(M + F)$

FIG. 2.1. *The high-level algorithm.*

THEOREM 2.1. *With high probability, High-Level*(G) *returns the MSF of* G *in* $O(\log n)$ *time using* $(m+n)/\log n$ *processors.*

In the following sections we describe and analyze the algorithms for phase 1 and phase 2 and then present the proof of the main theorem for the expected running time. We then obtain a high probability bound for the running time and work. When analyzing the performance of the algorithms in phase 1 and phase 2, we use a time-work framework, assuming perfect processor allocation. This can be achieved with high probability (to within a constant factor), using the load balancing scheme in [HZ96], which requires *superlinear* space. A linear-space load balancing scheme is claimed in [HZ01], though it is difficult to extricate the load balancing computation from the connectivity computation from [HZ01]. In section 7 we give a simple load balancing scheme based on [HZ96], which uses linear space. Its description is abstract and fully self-contained.

3. Phase 1. In phase 1, our goal is to contract the input graph G into a graph with $O(n/k_0)$ vertices. We do this by identifying certain edges in the MSF of G and contracting the connected components formed by these edges. The challenge here is to identify these edges in logarithmic time and linear work.

Phase 1 achieves the desired reduction in the number of vertices by constructing a *k-Min forest* (for $k = k_0$), defined below. This is similar to the algorithm in [CKT96]. However, our algorithm is considerably simpler. We show that a k-Min forest satisfies certain properties, and we exploit these properties to design a procedure *Borůvka-A*, which keeps the sizes of the trees contracted in the various stages of phase 1 very small so that the total time needed for contracting and processing edges in these trees is $o(\log n/2^{\log^* n})$. Phase 1 also needs a *Filter* subroutine, which removes "k-Min heavy" edges. For this, we show that we can use an MSF verification algorithm on the small trees we construct to perform this step. The overall algorithm for phase 1, called Find-k-Min, uses these two subroutines to achieve the stated reduction in the number of vertices within the desired time and work bounds.

3.1. The *k*-Min forest. Phase 1 uses the familiar "sample, contract, and discard edges" framework of earlier randomized algorithms for the MSF problem [KKT95, CKT94, CKT96, PR97]. However, instead of computing an MSF, we will construct the k_0-*Min tree* [CKT96] of each vertex (recall that $k_0 = (\log^{(2)} n)^2$). Contracting the edges in these k_0-Min trees will produce a graph with $O(n/k_0)$ vertices.

To understand what a k-Min tree is, consider the Dijkstra–Jarnik–Prim minimum spanning tree algorithm, given in Figure 3.1. This simple algorithm was discovered

independently by Dijkstra [Dij59], Jarnik [Jar30], and Prim [Prim57], though it is commonly known as Prim's algorithm. The edge set k-Min(v) consists of the first k edges chosen by this algorithm, when started at vertex v. A forest F is a *k-Min forest* of G if $F \subseteq \text{MSF}(G)$ and for all $v \in G$, k-Min$(v) \subseteq F$.

Dijkstra–Jarnik–Prim(G)
    $S := \{v\}$ *(choose an arbitrary starting vertex v)*
    $T := \emptyset$
    Repeat until T contains the MSF of G
        Choose minimum weight edge (a, b) s.t. $a \in S$, $b \notin S$
        $T := T + (a, b)$
        $S := S + b$

FIG. 3.1. *The Dijkstra–Jarnik–Prim algorithm.*

We define $P_T(x, y)$ to be the set of edges on the path from x to y in tree T, and $maxweight\{A\}$ be the maximum weight in a set of edges A.

For any forest F in G, define an edge (a, b) in G to be *F-heavy* if $weight(a, b) > maxweight\{P_F(a, b)\}$, and to be *$F$-light* otherwise. If a and b are not in the same tree in F, then (a, b) is F-light.

This notion of being F-heavy or F-light can be generalized as follows. Suppose F is a k-Min forest for some graph. We will say an edge (a, b) (not necessarily in the graph) is *k-Min-light* if F is not a k-Min forest for the graph $F+(a, b)$, and *k-Min-heavy* otherwise. An equivalent definition for k-Min-lightness and heaviness (which is more suited to proofs) is this: Let M be the k-Min tree of v. We define $weight_v^k(w)$ to be $maxweight\{P_M(v, w)\}$ if w appears in M, otherwise $weight_v^k(w) = maxweight\{M\}$. An edge (a, b) is then k-Min-light iff $weight(a, b) \leq \max\{weight_a^k(b), weight_b^k(a)\}$. Notice that $weight_v^k(w)$ implicitly depends on the underlying graph, which if not clear from the context will be stated explicitly.

FACT 3.1. *The two definitions given for "k-Min-light" (and "k-Min-heavy") are equivalent.*

We claimed earlier that k-Min-lightness is a generalization of F-lightness. To see this, set $k = n - 1$ and observe that an edge is k-Min-light iff it is F-light.

CLAIM 3.1. *If an edge (u, v) is k_1-Min-heavy w.r.t. G_1, it also is k_2-Min-heavy w.r.t. G_2, where $k_2 \leq k_1$, $V(G_1) = V(G_2)$, and $E(G_1) \subseteq E(G_2)$.*

Proof. This follows from two observations: $weight_v^k(w)$ is nondecreasing in k, and $weight_v^k(w)$ is nonincreasing as new edges are added to the underlyinggraph. □

In other words, whenever an edge is found to be k-Min-heavy for $k \geq k_0$ and w.r.t. some subset of the original graph, this is a certificate that the edge is k_0-Min-heavy in the original graph.

CLAIM 3.2. *For any k, $weight_v^k(w) \leq maxweight\{P_{MSF}(v, w)\}$.*

Proof. There are two cases: when w falls inside the k-Min tree of v and when it falls outside. If w lies inside k-Min(v), then $weight_v^k(w)$ must be the same as $maxweight\{P_{MSF}(v, w)\}$ since k-Min$(v) \subseteq MSF$. Now suppose that w falls outside k-Min(v) and $weight_v^k(w) > maxweight\{P_{MSF}(v, w)\}$. There must be a path from v to w in the MSF consisting of edges lighter than $maxweight\{k\text{-Min}(v)\}$. However, at each step in the Dijkstra–Jarnik–Prim algorithm, at least one edge in $P_{MSF}(v, w)$ is eligible to be chosen in that step. Since $w \notin k$-Min(v), the edge with weight $maxweight\{k\text{-Min}(v)\}$ is never chosen, a contradiction. □

LEMMA 3.1. *Let H be a graph formed by sampling each edge in graph G with probability p. The expected number of edges in G that are k-Min-light w.r.t. H for any k is less than n/p.*

Proof. We show that any edge that is k-Min-light in G also is F-light where F is the MSF of H. The lemma then follows from the sampling lemma of [KKT95] which states that the expected number of F-light edges in G is less than n/p. Let us look at any k-Min-light edge (v, w). By Claim 3.2, $weight_v^k(w) \leq maxweight\{P_{MSF}(v, w)\}$, the measure used to determine F-lightness. Thus the criterion for k-Min-lightness, $\max\{weight_v^k(w), weight_w^k(v)\}$, must also be no more than $maxweight\{P_{MSF}(v, w)\}$. Restating this, if (v, w) is k-Min-light, it must be F-light as well. □

We will use the above property of a k-Min forest to develop a procedure Find-k-Min(G, l). It takes as input the graph G and a suitable positive integer l, and returns a k_0-Min forest of G. For $l = \log^* n$, it runs in logarithmic time and linear work. In the next few sections we describe some basic steps and procedures used in Find-k-Min, and then present and analyze this main procedure of phase 1.

Since phase 1 is concerned only with the k_0-Min tree of each vertex, it suffices to retain only the lightest k_0 edges incident on each vertex. Hence, as stated in the first step of phase 1 in algorithm High-Level from section 2, we will discard all but the lightest k_0 edges incident on each vertex since we will not need them until phase 2. This step can be performed in logarithmic time and linear work by a simple randomized algorithm that selects a sample of size $\sqrt{|L|}$ from each adjacency list L, sorts this sample, and then uses this sorted list to narrow the search for the k_0th smallest element to a list of size $O(|L|^{3/4})$.

3.2. Borůvka-A steps. In a basic Borůvka step [Bor26], each vertex chooses its minimum weight incident edge, inducing a number of disjoint trees. All such trees are then contracted into single vertices, and the useless edges are discarded. We will call edges connecting two vertices in the same tree *internal* and all others *external*. All internal edges are useless, and if multiple external edges join the same two trees, all but the lightest are useless.

Our algorithm for phase 1 uses a *modified Borůvka step* in order to reduce the time bound to $o(\log n)$ per step. All vertices are classified as being either *live* or *dead*; only live vertices participate in modified Borůvka steps. After such a step, vertex v's *parent pointer* is $p(v) = w$, where (v, w) is the edge of minimum weight incident on v. In addition, each vertex has a *threshold* which keeps the weight of the lightest discarded edge adjacent to v. The algorithm discards edges known not to be in the k_0-Min tree of any vertex. The threshold variable guards against vertices choosing edges which may not be in the MSF. A dead vertex v has the useful property (shown below) that for any edge e in k_0-Min(v), weight$(e) \leq$ weight$(v, p(v))$, thus dead vertices *need not participate* in any more Borůvka steps.

It is well known that a Borůvka step generates a forest of *pseudo-trees*, where each pseudo-tree is a tree together with one extra edge that forms a cycle of length 2. In our algorithm we will assume that a Borůvka step also removes one of the edges in the cycle so that it generates a collection of rooted trees.

The following three claims refer to any tree resulting from a (modified) Borůvka step. Their proofs are straightforward and are omitted.

CLAIM 3.3. *The sequence of edge weights encountered on a path from v to root(v) is monotonically decreasing.*

CLAIM 3.4. *If depth$(v) = d$, then d-Min(v) consists of the edges in the path from v to root(v). Furthermore, the weight of $(v, p(v))$ is greater than any other edge in*

d-Min(v).

CLAIM 3.5. *If the minimum-weight incident edge of u is (u,v), k-Min$(u) \subseteq (k$-Min$(v) + (u,v))$.*

Claim 3.6 may not be as obvious. A similar claim was proved in [CHL01].

CLAIM 3.6. *Let T be a tree induced by a Borůvka step, and let T' be a subtree of T. If e is the minimum weight incident edge on T, then the minimum weight incident edge on T' is either e or an edge of T.*

Proof. Suppose, on the contrary, that the minimum weight incident edge on T' is $e' \notin T$, and let v and v' be the end points of e and e', which are inside T. Consider the paths P (P') from v (v') to the root of T. By Claim 3.3, the edge weights encountered on P and P' are monotonically decreasing. There are two cases. If T' contains some, but not all of P', then e' must lie along P', a contradiction. If T' contains all of P', but only some of P, then some edge $e'' \in P$ is adjacent to T'. Then $w(e') < w(e'') < w(e)$, also a contradiction. □

The procedure Borůvka-A(H, l, F), given in Figure 3.2, returns a contracted version of H with the number of live vertices reduced by a factor of l. Edges designated as parent pointers, which are guaranteed to be in the MSF of H, are returned in F. Initially $F = \emptyset$.

Borůvka-A(H, l, F)
Repeat $\log l$ times: *($\log l$ modified Borůvka steps)*
$F' := \emptyset$
For each live vertex v
Choose min. weight edge (v, w)
(1) If weight$(v, w) >$ threshold(v), v becomes dead, stop else
p$(v) := w$
$F' := F' + (v,$ p$(v))$
Each tree T induced by edges of F' is one of two types:
If root of T is dead, then
(2) Every vertex in T becomes dead *(Claim 3.5)*
If T contains only live vertices,
(3) If depth$(v) \geq k_0$, v becomes dead *(Claim 3.4)*
Contract the subtree of T made up of live vertices.
The resulting vertex is live, has no parent pointer, and
keeps the smallest threshold of its constituent vertices.
$F := F + F'$

FIG. 3.2. *The Borůvka-A procedure.*

LEMMA 3.2. *If Borůvka-A designates a vertex as dead, its k_0-Min tree has already been found.*

Proof. Vertices make the transition from live to dead only at the lines indicated by a number. By our assumption that we discard only edges that cannot be in the k_0-Min tree of any vertex, if the lightest edge adjacent to any vertex has been discarded, we know its k_0-Min tree has already been found. This covers line (1). The correctness of line (2) follows from Claim 3.5. Since $(v, p(v))$ is the lightest incident edge on v, k_0-Min$(v) \subseteq k_0$-Min$(p(v)) + (v, p(v))$. If $p(v)$ is dead, then v can also be called dead. Since the root of a tree is dead, vertices at depth one are dead, implying vertices at depth two are dead, and so on. The validity of line (3) follows directly from Claim

3.4. If a vertex finds itself at depth $\geq k_0$, its k_0-Min tree lies along the path from the vertex to its root. □

LEMMA 3.3. *After a call to Borůvka-A(H, k_0+1, F), the k_0-Min tree of each vertex is a subset of F.*

Proof. By Lemma 3.2, dead vertices already satisfy the lemma. After a single modified Borůvka step, the set of parent pointers associated with live vertices induce a number of trees. Let $T(v)$ be the tree containing v. We assume inductively that after $\lceil \log i \rceil$ modified Borůvka steps, the $(i-1)$-Min tree of each vertex in the original graph has been found (this is clearly true for $i = 1$). For any live vertex v let (x, y) be the minimum weight edge s.t. $x \in T(v)$, $y \notin T(v)$. By the inductive hypothesis, the $(i-1)$-Min trees of v and y are subsets of $T(v)$ and $T(y)$, respectively. By Claim 3.6, (x, y) is the first external edge of $T(v)$ chosen by the Dijkstra–Jarnik–Prim algorithm, starting at v. As every edge in $(i-1)$-Min(y) is lighter than (x, y), $(2(i-1)+1)$-Min(v) is a subset of $T(v) \cup \{(x, y)\} \cup T(y)$. Since edge (x, y) is chosen in the $(\lceil \log i \rceil + 1)$st modified Borůvka step, $(2i-1)$-Min(v) is a subset of $T(v)$ after $\lceil \log i \rceil + 1 = \lceil \log 2i \rceil$ modified Borůvka steps. Thus after $\log(k_0+1)$ steps, the k_0-Min tree of each vertex has been found. □

LEMMA 3.4. *After b modified Borůvka steps, the length of any edge list is bounded by ${k_0}^{{k_0}^b}$.*

Proof. This is true for $b = 0$. Assuming the lemma holds for $b-1$ modified Borůvka steps, the length of any edge list after that many steps is $\leq {k_0}^{{k_0}^{b-1}}$. Since we contract only trees of height $< k_0$, the length of any edge list after b steps is $< ({k_0}^{{k_0}^{b-1}})^{k_0} = {k_0}^{{k_0}^b}$. □

It is shown in the next section that our algorithm deals only with graphs that are the result of $O(\log k_0)$ modified Borůvka steps. Hence the maximum length edge list is ${k_0}^{{k_0}^{O(\log k_0)}}$.

The costliest step in Borůvka-A is calculating the depth of each vertex. After the minimum weight edge selection process, the root of each induced tree will broadcast its depth to all depth 1 vertices, which in turn broadcast to depth 2 vertices, etc. Once a vertex knows it is at depth $k_0 - 1$, it may stop, letting all its descendents infer that they are at depth $\geq k_0$. Interleaved with each round of broadcasting is a processor allocation step. We account for this cost separately in section 7.

LEMMA 3.5. *Let G_1 have m_1 edges. Then a call to Borůvka-A(G_1, l, F) can be executed in time $O({k_0}^{O(\log k_0)} + \log l \cdot \log n \cdot (m_1/m))$ with $(m+n)/\log n$ processors.*

Proof. Let G_1 be the result of b modified Borůvka steps. By Lemma 3.4, the maximum degree of any vertex after the ith modified Borůvka step in the current call to Borůvka-A is ${k_0}^{{k_0}^{b+i}}$. Let us now look at the required time of the ith modified Borůvka step. Selecting the minimum cost incident edge takes time $O(\log {k_0}^{{k_0}^{b+i}})$, while the time to determine the depth of each vertex is $O(k_0 \cdot \log {k_0}^{{k_0}^{b+i}})$. Summing over the $\log l$ modified Borůvka steps, the total time is bounded by $\sum_i^{\log l} {k_0}^{O(b+i)} = {k_0}^{O(b+\log l)}$. As noted above, the algorithm performs $O(\log k_0)$ modified Borůvka steps on any graph, hence the time is ${k_0}^{O(\log k_0)}$.

The work performed in each modified Borůvka step is linear in the number of edges. Summing over $\log l$ such steps and dividing by the number of processors, we arrive at the second term in the stated running time. □

3.3. Filtering edges via the Filter forest. We will maintain, concurrent with the operation of Borůvka-A, a structure called the *Filter forest*. This collec-

tion of rooted trees records which vertices merged together and the edge weights involved. (This structure appeared first in [K97].) If v is a vertex of the original graph or a new vertex resulting from contracting a set of edges, there is a corresponding vertex $\phi(v)$ in the Filter forest. During a Borůvka step, if a vertex v becomes dead, a new vertex x is added to the Filter forest, as well as a directed edge $(\phi(v), x)$ having the same weight as $(v, p(v))$. If live vertices $v_1, v_2, \ldots, v_j$ are contracted into a live vertex v, a vertex $\phi(v)$ is added to the Filter forest in addition to edges $(\phi(v_1), \phi(v)), (\phi(v_2), \phi(v)), \ldots, (\phi(v_j), \phi(v))$, having the weights of edges $(v_1, \mathrm{p}(v_1))$, $(v_2, \mathrm{p}(v_2)), \ldots,$ $(v_j, \mathrm{p}(v_j))$, respectively. We make the simple observation that the edge weights on the path from $\phi(u)$ to $root(\phi(u))$ are exactly the edge weights of the edges chosen by u (or its representative) in previous Borůvka steps.

It is shown in [K97] that the heaviest weight in the path from u to v in the MSF is the same as the heaviest weight in the path from $\phi(u)$ to $\phi(v)$ in the Filter forest (if there is such a path). We extend this scheme to handle k-Min-lightness.

Let $P_f(y, z)$ be the path from y to z in the Filter forest. If $\phi(u)$ and $\phi(v)$ are in the same Filter tree, then let

$$w_u(v) = w_v(u) = maxweight\{P_f(\phi(u), \phi(v))\}.$$

If $\phi(u)$ and $\phi(v)$ are not in the same Filter tree, then let

$$w_u(v) = maxweight\{P_f(\phi(u), \mathrm{root}(\phi(u)))\},$$
$$w_v(u) = maxweight\{P_f(\phi(v), \mathrm{root}(\phi(v)))\}.$$

In a call to Filter(H, F) (from the procedure Find-k-Min, section 3.4), we examine each edge $e = (u, v)$ in $H - F$ and remove or *filter* e from H if $weight(e) > \max\{w_u(v),\ w_v(u)\}$. Notice that if $w_u(v) = weight_u^{k_0}(v)$ for all v, then we will filter out edges precisely when they are k_0-Min-heavy. We show below that using $w_u(v)$ in lieu of $weight_u^{k_0}(v)$ causes no problems: we retain all k_0-Min-light edges without retaining too many edges in total.

To implement the Filter procedure we use a slight modification to the $O(\log n)$-time, $O(m)$-work MSF verification algorithm of [KPRS97]. If $e = (u, v)$ is the edge being tested and $\phi(u), \phi(v)$ are not in the same Filter tree, we test the pairs $(\phi(u), root(\phi(u)))$ and $(\phi(v), root(\phi(v)))$ instead and delete e if both of these pairs are identified to be deleted. This computation actually takes time $O(\log r)$ where r is the size of the largest tree formed.

Lemmas 3.6 and 3.7, proved below, establish the correctness of the filtering procedure.

LEMMA 3.6. *Suppose b modified Borůvka steps were applied to a graph; then for any vertex u and some $k \geq \min\{k_0, 2^b - 1\}$,*

$$maxweight\{P_f(\phi(u), root(\phi(u)))\} = maxweight\{k\text{-}Min(v)\}.$$

Before proving this we first prove a necessary technical lemma.

LEMMA 3.7. *Let T be a tree of MSF edges after an arbitrary number of Borůvka steps and let $T' = T \cup \{(v, w)\}$, where (v, w), $v \in T$, $w \notin T$ is the edge chosen by T in the next Borůvka step. For any $u \in T$, the maximum weight edge in $P_{T'}(u, w)$ was chosen by the tree containing u in some Borůvka step.*

Proof. Let T be formed after b Borůvka steps. Suppose, without loss of generality, that the lemma is falsified for the first time after the bth Borůvka step. That is, the

heaviest edge in $P_{T'}(u,w)$, say f, was chosen in the bth step. Let $g \neq f$ be the edge chosen by u or u's representative tree in this step. If f lies between g and the root of T, then by Claim 3.3 it is lighter than g, and similarly, if it lies between vertex v and the root of T, then it is lighter than (v,w). Both cases are contradictions. □

We are now ready to prove Lemma 3.6.

Proof. Let $e(u,b)$ be the maximum weight edge chosen by u's tree in the first b Borůvka steps. Assume inductively that $weight(e(u,b-1)) = maxweight\{k(u,b-1)\text{-Min}(u)\}$, where $k(u,b-1) \geq 2^{b-1}-1$ if u is live, $k(u,\cdot) \geq k_0$ if u is dead, and $k(u,b-1)$-Min(u) is contained in a tree of MSF edges after $b-1$ Borůvka steps. If u is dead, it already satisfies the inductive claim for b Borůvka steps, so assume u is alive. Let (z_1,z_2) be the edge chosen by the tree containing u in the bth Borůvka step and let P be the MSF path connecting $k(u,b-1)$-Min(u) to z_1—see Figure 3.3 for a schematic diagram. We have that $weight(z_1,z_2) > weight(e(z_2,b-1))$, because (z_1,z_2) was not already chosen by z_2 in the first $b-1$ steps, and $maxweight\{e(u,b-1)+P+(z_1,z_2)\} = weight(e(u,b))$. This is true because $weight(e(u,b)) = maxweight\{e(u,b-1),(z_1,z_2)\} > maxweight\{P\}$, where the equality is by definition and the inequality is by Lemma 3.7. Let D be the subgraph

$$D = k(u,b-1)\text{-Min}(u) + P + (z_1,z_2) + k(z_2,b-1)\text{-Min}(z_2)$$

and $k(u,b)$ be the smallest number such that $k(u,b)$-Min$(u) \supseteq D$. It follows that $e(u,b)$ is the heaviest edge in $k(u,b)$-Min(u) because when the Dijkstra–Jarnik–Prim algorithm is started from u, until all edges from D are chosen, there is *some* eligible edge from D weighing no more than the edge $e(u,b)$.

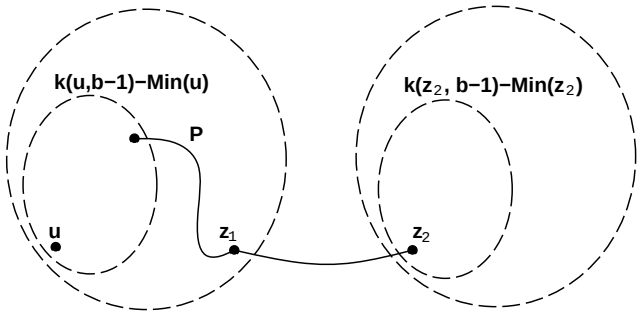


FIG. 3.3. *The larger ovals represent the trees of MSF edges after $b-1$ Borůvka steps containing u and z_2, respectively. The smaller ovals are the $k(u,b-1)$-Min(u) tree and the $k(z_2,b-1)$-Min(z_2) tree.*

If u and z_2 remain live, then $k(u,b) \geq 2\cdot(2^{b-1}-1)+1 \geq 2^b-1$. On the other hand, if u becomes dead after the bth Borůvka step, then (z_1,z_2) is the heaviest edge at the end of a chain C of length at least k_0 and $k(u,b) \geq 2^{b-1}-1+|C| \geq k_0$. In either case our inductive claim is proved for b Borůvka steps. □

LEMMA 3.8. *Suppose the Filter procedure is called only on graphs after performing at least* $\log(k_0+1)$ *Borůvka steps. Then no k_0-Min-light edges are filtered, and all unfiltered edges are k-Min-light for some $k \geq k_0$.*

Proof. Consider an edge (u,v) examined by the Filter procedure. Note that if $\phi(u)$ is in the same Filter tree as $\phi(v)$, by King's observation [K97], $w_u(v) = w_v(u) = maxweight\{P_{MSF}(u,v)\}$. If $weight(u,v)$ is greater than $w_u(v)$, then by Claim 3.1 (u,v) is k_0-Min-heavy and may be safely filtered. On the other hand, if $weight(u,v)$

is less than $w_u(v)$, then (u, v) is k-Min-light for $k = n$. We therefore focus on the case when $\phi(u)$ and $\phi(v)$ are in different Filter trees.

By Lemma 3.6, for some k_1, k_2 we have that $w_u(v) = maxweight\{k_1\text{-Min}(u)\}$ and $w_v(u) = maxweight\{k_2\text{-Min}(v)\}$. Since $maxweight\{k\text{-Min}(u)\}$ is a nondecreasing function of k, if (u, v) is not filtered out, then by Claim 3.1 it must be k_3-Min-light where $k_3 = \max\{k_1, k_2\}$. On the other hand, if (u, v) is filtered out, then it must be k_4-Min-heavy where $k_4 = \min\{k_1, k_2\}$. Because the Filter procedure is applied only after performing at least $\log(k_0 + 1)$ Borůvka steps, by Lemma 3.6 $k_3, k_4 \geq k_0$. □

Remark. Filter is responsible for updating the threshold variables—see section 3.2. When an edge (u, v) is discarded, threshold(u) is updated to reflect the weight of the lightest discarded edge incident to u; threshold(v) is updated similarly.

3.4. Finding a *k*-Min forest. We are now ready to present the main procedure of phase 1, Find-k-Min, which is given in Figure 3.4. (Recall that the initial call, given in section 2, is Find-k-Min$(G_t, \log^* n)$, where G_t is the graph obtained from G by removing all but the k_0 lightest edges on each adjacency list.)

Find-k-Min(H, i)
- $H_c :=$ Borůvka-A$(H, (\log^{(i-1)} n)^4, F)$
- if $i = 3$, return(F)
- $H_s :=$ sample edges of H_c with prob. $1/(\log^{(i-1)} n)^2$
- $F_s :=$ Find-k-Min$(H_s, i - 1)$
- $H_f :=$ Filter(H_c, F_s)
- $F' :=$ Find-k-Min$(H_f, i - 1)$
- Return$(F + F')$

FIG. 3.4. *The Find-k-Min procedure.*

H is a graph with some vertices possibly marked as dead; i is a parameter that indicates the level of recursion (which determines the number of Borůvka steps to be performed and the sampling probability). Lemmas 3.9 and 3.10 establish the correctness of this procedure. The performance of Find-k-Min is analyzed in section 3.5.

LEMMA 3.9. *Let H' be a graph formed by sampling each edge in H with probability p, and let F be a k_0-Min forest of H' (derived by at least $\log(k_0 + 1)$ Borůvka steps). The call to Filter(H, F) returns a graph containing a k_0-Min forest of H, whose expected number of edges is no more than n/p.*

Proof. By Claim 3.1, any edge in the k_0-Min forest of H is k_0-Min-light w.r.t. H'. By Lemma 3.8, no edges k_0-Min-light w.r.t. H' are filtered; this establishes the first part of the lemma. By the second part of Lemma 3.8, all edges not filtered are k-Min-light w.r.t. H' for some k. According to Lemma 3.1, the number of edges in H that are k-Min-light w.r.t. H' for any k is no more than n/p. This establishes the rest of the lemma. □

LEMMA 3.10. *The call Find-k-Min$(G_t, \log^* n)$ returns a set of edges that includes the k_0-Min tree of each vertex in G_t.*

Proof. The proof is by induction on i. For $i = 3$ (the base case) Find-k-Min$(H, 3)$ returns F, which by Lemma 3.3 contains the k_0-Min tree of each vertex. Now assume inductively that Find-k-Min$(H, i - 1)$ returns the k_0-Min tree of H. Consider the call Find-k-Min(H, i). By the induction assumption the call to Find-k-Min$(H_s, i - 1)$ returns the k_0-Min tree of each vertex in H_s. By Lemma 3.9, the call to Filter(H_c, F_s) returns in H_f a set of edges that contains the k_0-Min trees of all vertices in H_c.

Finally, by the inductive assumption, the set of edges returned by the call to Find-k-Min$(H_f, i-1)$ contains the k_0-Min trees of all vertices in H_f. Since F' contains the $(\log^{(i-1)} n)$-Min tree of each vertex in H, and Find-k-Min(H, i) returns $F + F'$, it returns the edges in the k_0-Min tree of each vertex in H. □

3.5. Performance of find-k-Min. In this section we bound the time and work required by the Find-k-Min procedure.

CLAIM 3.7. *The following invariants are maintained at each call to Find-k-Min. The number of live vertices in* $H \le n/(\log^{(i)} n)^4$*, and the expected number of edges in* $H \le m/(\log^{(i)} n)^2$*, where* m *and* n *are the number of edges and vertices in the original graph.*

Proof. These hold for the initial call, when $i = \log^* n$. By Lemma 3.3, the contracted graph H_c has $\le n/(\log^{(i-1)} n)^4$ live vertices. Since H_s is derived by sampling edges with probability $1/(\log^{(i-1)} n)^2$, the expected number of edges in H_s is $\le m/(\log^{(i-1)} n)^2$, maintaining the invariants for the first recursive call.

By Lemma 3.1, the expected number of edges in $H_f \le \frac{n(\log^{(i-1)} n)^2}{(\log^{(i-1)} n)^4} = \frac{n}{(\log^{(i-1)} n)^2}$. Since H_f has the same number of vertices as H_c, both invariants are maintained for the second recursive call. □

LEMMA 3.11. *Find-k-Min*$(G_t, \log^* n)$ *runs in expected* $O(\log n)$ *time and* $O(m)$ *work.*

Proof. Since recursive calls to Find-k-Min proceed in a sequential fashion, the total running time is the sum of the local computation performed in each invocation. Aside from randomly sampling the graph, the local computation consists of calls to Filter and Borůvka-A.

In a given invocation of Find-k-Min, the number of Borůvka steps performed on graph H is the sum of all Borůvka steps performed in all ancestral invocations of Find-k-Min, i.e., $\sum_{i=3}^{\log^* n} O(\log^{(i)} n)$, which is $O(\log^{(3)} n)$. From our bound on the maximum length of edge lists (Lemma 3.4), we can infer that the size of any tree in the Filter forest is ${k_0}^{k_0^{O(\log^{(3)} n)}}$, thus the time needed for each modified Borůvka step and each Filter step is ${k_0}^{O(\log^{(3)} n)}$. Summing over all such steps, the total time required is $o(\log n)$.

The work required by the Filter procedure and *each* Borůvka step is linear in the number of edges. By Claim 3.7, the expected number of edges in an invocation at level i is $O(m/(\log^{(i)} n)^2)$. Since there are $O(\log^{(i)} n)$ Borůvka steps performed in this invocation, the work required is $O(m/\log^{(i)} n)$. There are $2^{\log^* n - i}$ invocations with depth parameter i; therefore the total work is given by $\sum_{i=3}^{\log^* n} 2^{\log^* n - i} O(m/\log^{(i)} n)$, which is $O(m)$. □

4. Phase 2. Recall the phase 2 portion of our overall algorithm High-Level:

(the number of vertices in G_s is $\le n/k_0$)
$G_s :=$ Sample edges of G' with prob. $1/\sqrt{k_0} = 1/\log^{(2)} n$
$F_s :=$ Find-MSF(G_s)
$G_f :=$ Filter(G', F_s)
$F :=$ Find-MSF(G_f)

The procedure Filter(G, F) [KPRS97] returns the F-light edges of G. The procedure Find-MSF(G_1), described below, finds the MSF of G_1 in $O(\frac{m_1}{m} \log n \log^{(2)} n)$ time, where m_1 is the number of edges in G_1.

The graphs G_s and G_f each have expected $m/\sqrt{k_0} = m/\log^{(2)} n$ edges since G_s is derived by sampling each edge with probability $1/\sqrt{k_0}$, and, by the sampling lemma of [KKT95], the expected number of edges in G_f is $(m/k_0)/(1/\sqrt{k_0}) = m/\sqrt{k_0}$. Because we call Find-MSF on graphs having expected size $O(m/\log^{(2)} n)$, each call takes $O(\log n)$ time.

4.1. The Find-MSF procedure. The procedure Find-MSF(H), given in Figure 4.1, is similar to previous randomized parallel algorithms except it uses no recursion. Instead, a separate *BaseCase* algorithm is used in place of recursive calls. We also use slightly different Borůvka steps in order to reduce the work. These modifications are inspired by [PR97] and [PR98].

As its BaseCase, we use the simplest version of the algorithm of Chong, Han, and Lam [CHL01], which takes time $O(\log n)$ using $(m+n)\log n$ processors. By guaranteeing that it is called only on graphs of expected size $O(m/\log^2 n)$, the running time remains $O(\log n)$ with $(m+n)/\log n$ processors. An adaptation of our algorithm to the CRCW PRAM leads to one roughly twice as fast as [CKT96]. Because of a more efficient phase 1 we can afford to make only four BaseCase calls in phase 2, rather than eight calls as in [CKT96].

Find-MSF(H)
  $H_c :=$ Borůvka-B($H, \log^4 n, F$)
  $H_s :=$ Sample edges of H_c with prob. $p = 1/\log^2 n$
  $F_s :=$ BaseCase(H_s)
  $H_f :=$ Filter(H_c, F_s)
  $F' :=$ BaseCase(H_f)
  Return($F + F'$)

FIG. 4.1. *The Find-MSF procedure.*

After the call to Borůvka-B, the graph H_c has $< m/\log^4 n$ vertices. Since H_s is derived by sampling the edges of H_c with probability $1/\log^2 n$, the expected number of edges to the first BaseCase call is $O(m/\log^2 n)$. By the sampling lemma of [KKT95], the expected number of edges to the second BaseCase call is $< (m/\log^4 n)/(1/\log^2 n)$, thus the total time spent in these subcalls is $O(\log n)$. Assuming the size of H conforms to its expectation of $O(m/\log^{(2)} n)$, the calls to Filter and Borůvka-B also take $O(\log n)$ time, as described below.

The Borůvka-B(H, l, F) procedure, shown in Figure 4.2, returns a contracted version of H with $O(m/l)$ vertices. It uses a simple growth control schedule, designating vertices as *inactive* if their degree exceeds l. We can determine if a vertex is inactive by performing list ranking on its edge list for $\log l$ time steps. If the computation has not stopped after this much time, then its edge list has length $> l$.

The last step takes $O(\log n)$ time; all other steps take $O(\log l)$ time, as they deal with edge lists of length $O(l)$. Consequently, the total running time is $O(\log n + \log^2 l)$. For each iteration of the main loop, the work is linear in the number of edges. Assuming the graph conforms to its expected size of $O(m/\log^{(2)} n)$, the total work is linear. The edge-plugging technique as well as the idea of a growth control schedule were introduced by Johnson and Metaxas [JM92].

Borůvka-B(G, l, F)
Repeat $\log l$ times
For each vertex, let it be *inactive* if its edge list
has more than l edges, and *active* otherwise.
For each *active* vertex v
choose min. weight incident edge e
$F := F + e$
Using the edge-plugging technique, build a
single edge list for each induced tree ($O(1)$ time)
Contract all trees of inactive vertices

FIG. 4.2. *The Borůvka-B procedure.*

5. Proof of Theorem 2.1. The set of edges M returned by Find-k-Min is a subset of the MSF of G. By contracting the edges of M to produce G', the MSF of G is given by the edges of M together with the MSF of G'. The call to Filter produces graph G_f by removing from G' edges known not to be in the MSF. Thus the MSF of G_f is the same as the MSF of G'. Assuming the correctness of Find-MSF, the set of edges F constitutes the MSF of G_f, and thus $M + F$ is the MSF of G.

Earlier we have shown that each step of High-Level requires $O(\log n)$ time and work linear in the number of edges. In the next two sections we show that w.h.p, the number of edges encountered in all graphs during the algorithm is linear in the size of the original graph.

6. High probability bounds. Consider a single invocation of Find-k-Min(H, i), where H has m' edges and n' vertices. We want to place likely bounds on the number of edges in each recursive call to Find-k-Min, in terms of m' and i.

For the first recursive call, the edges of H are sampled independently with probability $1/(\log^{(i-1)} n)^2$. Call the sampled graph H_1. By applying a Chernoff bound [AS00], the probability that the size of H_1 is less than twice its expectation is $1 - \exp(-\Omega(m'/(\log^{(i-1)} n)^2))$.

Before analyzing the second recursive call, we recall the sampling lemma of [KKT95] which states that the number of F-light edges is dominated by the negative binomial distribution with parameters n' and p, where p is the sampling probability, and F is the MSF of H_1. As we saw in the proof of Lemma 3.1, every k-Min-light edge must also be F-light. Using this observation, we will analyze the size of the second recursive call in terms of F-light edges and conclude that any bounds we attain apply equally to k-Min-light edges.

We now bound the likelihood that more than twice the expected number of edges are F-light. This is the probability that in a sequence of more than $2n'/p$ flips of a coin, with probability p of heads, the coin comes up heads less than n' times (since each edge selected by a coin toss of heads goes into the MSF of the sampled graph). By applying a Chernoff bound, this is $\exp(-\Omega(n'))$. In this particular instance of Find-k-Min, $n' \leq m/(\log^{(i-1)} n)^4$ and $p = 1/(\log^{(i-1)} n)^2$, so the probability that fewer than $2m/(\log^{(i-1)} n)^2$ edges are F-light is $1 - \exp(-\Omega(m/(\log^{(i-1)} n)^4))$.

Given a single invocation of Find-k-Min(H, i), we can bound the probability that H has more than $2^{\log^* n - i}\, m/(\log^{(i)} n)^2$ edges by $\exp(-\Omega(m/(\log^{(i)} n)^4))$. This follows from applying the argument used above to each invocation of Find-k-Min from the initial call to the current call at depth $\log^* n - i$. Summing over all recursive calls

to Find-k-Min, the total number of edges (and thus the total work) is bounded by $\sum_{i=3}^{\log^* n} 2^{2\log^* n - 2i}\, m/(\log^{(i)} n)^2 = O(m)$ with probability $1-\exp(-\Omega(m/(\log^{(3)} n)^4))$.

The probability that phase 2 uses $O(m)$ work is $1-\exp(-\Omega(m/\log^2 n))$. We omit the analysis as it is similar to the analysis for phase 1.

The probability that our bounds on the time and total work performed by the algorithm fail to hold is exponentially small in the input size. However, this assumes perfect processor allocation. In the next section we show that the probability that work fails to be distributed evenly among the processors is less than $1/m^{\omega(1)}$. Thus the overall probability of failure is very small, and the algorithm runs in logarithmic time and linear work w.h.p.

7. Processor allocation. As stated in section 2, the processor allocation needed for our algorithm can be performed by a fairly simple scheme given in [HZ96] that takes logarithmic time and linear work overall but uses *superlinear* space. An algorithm claimed in [HZ01] uses linear space; however, it is not given a clear description in [HZ01] and, more seriously, it makes heavy use of a nontrivial linked-list based sorting algorithm of Goodrich and Kosaraju [GK96]. In this section we give a self-contained description of a processor allocation scheme for "tree structured" computations which does not use any sorting subroutine.

Let M be a set of m processes which perform some computation. So long as the computation is *tree structured*, in the sense given below, its exact nature is unimportant. At any point in the computation there is a set $D \subseteq M$ of *dead processes* and a stack $\mathcal{S} = (S_0, S_1, \ldots, S_d)$, where $S_0 = M$ and $S_{j+1} \subseteq S_j$. In the ith round of computation, the stack is potentially changed and some set $R_i \subseteq M$ of the processes compute for t_i time steps. Round i follows these steps:

1. Either (a) $\mathcal{S}$ is unchanged, $R_i := S_d - D$, or
   (b) $\mathcal{S} := (S_0, \ldots, S_d, S_{d+1})$, $S_{d+1} \subseteq S_d$, $R_i := S_{d+1} - D$, or
   (c) $\mathcal{S} := (S_0, \ldots, S_{d-1})$, $R_i := S_{d-1} - D$.
2. The R_i do something for $t_i \geq 1$ steps.
3. $D := D + \{\text{some subset of } R_i\}$.

This is a rather technical characterization of a class of algorithms. Informally, any recursive algorithm fits into this scheme if the active processes in one recursive call are a subset of the active processes from its parent call.

Let $p \leq m$ be the number of EREW processors available. Ideally, we would like to simulate round i in $O(\lceil R_i/p \rceil)$ time (i.e., with zero overhead). Like [HZ96] our overhead is nonconstant but usually negligible.

THEOREM 7.1. *For some tree computation, let r be the total number of rounds, $T = \sum_i t_i$ be the total time for all rounds, $W = \sum_i t_i \cdot |R_i|$ be the total work for all rounds, d_{max} be the maximum depth of the stack, and $q = \Omega(\log(mr))$ be a parameter. Then with probability $1 - e^{-\Omega(q)}$ the computation of m processes can be simulated with p EREW processors in $O(T + W/p + r\log q + \log p)$ time. The space required is $O(m + p \cdot d_{max})$. It is assumed there is some (easily computable) bound r_i such that $r_i \geq |R_i|$ and $r_i = O(|R_i|)$.*

In our MSF algorithm the number of rounds $r = O(2^{\log^* n} k_0 \log k_0) = O(\log^3 \log n)$, the time $T = O(\log n)$, work $W = O(m)$, and $d_{max} = O(\log^3 \log n)$. Plugging these values into Theorem 7.1, our MSF algorithm can be simulated in $O(m/p + \log n + \log q \log^3 \log n)$ time with probability at least $1 - e^{-\Omega(q)}$, using space $O(m + p \log^3 \log n)$. Since $p < m/\log n$, the space is linear in m. We could set $q =$

$\Theta(\log(mr)) = \Theta(\log n)$ and achieve a polynomially small error probability or set q as high as $2^{\log n/\log^3 \log n}$ for a much smaller error probability. Also, the "dead processes" correspond to those edges known to be k_0-Min-heavy (in phase 1) or not in the MSF at all (in phase 2).

As in [HZ96] we organize the processes into *blocks* of size $b = qm/p$ (q processors per block) as follows. We imagine placing the processes deterministically into an $m/b \times b$ array, then performing a random rotation on each column. The processes that end up in the same row are in the same block. Computing this initial allocation is easily done in $O(m/p + \log p)$ time. Since processors from different blocks do not communicate we will isolate our discussion to a single arbitrary block. Let B denote the set of processes in this block; initially $|B| = b$.

We maintain the invariant that the block is represented as a linked list $L = L_d, L_{d-1}, \ldots, L_0$, where $L_d = S_d - D$, and, in general, $L_j = S_j - S_{j+1} - D$. That is, $L = B - D$: no dead processes appear in this list, and L_j lists those processes that do not appear higher up in the stack. We also maintain that for all j, L_j has been fairly allocated. What this means is that the ℓth processor ($0 \le \ell < q$) assigned to this block "owns" a sublist $L_{\ell,j}$ of L_j extending from element $\ell\lceil \frac{|L_j|}{q} \rceil$ to element $(\ell+1)\lceil \frac{|L_j|}{q} \rceil - 1$ (if they exist). We assume processor ℓ has a pointer to $L_{\ell,j}$. (These pointers contribute the pd_{max} term to the space in Theorem 7.1. The other space requirements are linear in m.)

Suppose in round i, step 1 is of type (a)—the stack is not altered. Then $R_i \cap L = L_d$ and we already have a fair allocation of L_d. Provided $|L_d|$ is about the same in this block as in any other, step 2 can be simulated optimally in $O(t_i \cdot \lceil |L_d|/q \rceil)$ time. This will be discussed later. To restore our invariants after step 3 we simply need to splice out newly dead processes from L_d and compute a fair allocation for the new list. Let L_d and L'_d be the list before and after step 3. Processor ℓ will find all $L'_{w,d}$ which lie in $L_{\ell,d}$, sending a pointer of $L'_{w,d}$ to processor w. For this task processor ℓ must know $|L'_d|$ and the number of elements from L'_d which lie before $L_{\ell,d}$, both of which can be computed in $O(|L_d|/q + \log q)$ time with a prefix-sums computation. Finally we compute L'_d by splicing out all dead elements, also in $O(|L_d|/q + \log q)$ time.[2] The other two cases for step 1, (b) and (c), involve either splitting L_d into two lists or combining L_d and L_{d-1} into one list, followed by a step to compute a fair allocation for the new list(s). We omit a discussion of these two cases; the techniques used are the same as in step 3.

In implementing step 2 we use the assumption that there is a known upper bound $r_i \ge |R_i|$ on the number of processes taking part in the ith round. (In our MSF algorithm, for instance, this upper bound would hold w.h.p.) We argue that with a certain probability (that depends on q) for every round i, every processor is given no more than $(1+\epsilon)(1+r_i/p)$ active processes. Each of the t_i time steps in step 2 is then easily simulated in $(1+\epsilon)(1+r_i/p)$ time. Consider the $m/b \times b$ array used in the initial allocation, and an arbitrary block and round. Let X_k be 1 if the process initially placed in the kth column is active in the round, and 0 otherwise. Because the rotations on different columns were *independent*, so too are the X_k's. Let $X = \sum_{k=1}^{q} X_k$ be the number of active processes appearing in the block; clearly $E(X) = |R_i|b/m \le r_i b/m$. Since each processor can be thought to have a "dummy" process associated with it which is active in every round, assume, without loss of generality, that $E(X) \ge q$. Noting that X is the sum of independent Bernoulli trials,

[2] The prefix-sums and splicing can, of course, be performed in one pass.

we can bound the probability that X deviates too far from its expectation using a Chernoff bound [AS00]. For $0 < \epsilon < 1$, $\Pr[X > (1+\epsilon)E(X)] < e^{-\Omega(\epsilon^2 E(X))}$, and for constant ϵ, the probability that *any* block in any round gets more than $1+\epsilon$ times its expectation is $< \frac{mr}{b} e^{-\Omega(q)} = e^{-\Omega(q)}$ since $q = \Omega(\log(mr))$. The analysis of our scheme is very similar to that of [HZ96] but considerably more efficient in terms of time. In [HZ96] ϵ is increased in order to reduce the probability of failure. In our scheme we would set ϵ to be a small constant and increase q (number of processors per block) as necessary. It is crucial to keep ϵ small because in either scheme nearly all processors spend an $\epsilon/(1+\epsilon)$ fraction of their time doing nothing! On the other hand, the q parameter can usually be increased dramatically with negligible effects on the overall running time. Hence our scheme achieves a low failure probability without excessive processor idling.

We remark that the space claimed in Theorem 7.1 can be reduced to $O(m)$ at the expense of a slightly more complicated scheme. The idea is to compute fair allocations only when necessary. Very frequently, a previously computed fair allocation is "fair enough." For instance, in step 1(b) L_d is split into two lists, L'_d and L'_{d+1}. If L'_{d+1} contains most of the elements from L_d, we might as well use the fair allocation of L_d instead of computing new ones for L'_{d+1} and L'_d.

8. Adaptations to other practical parallel models. Our results imply good MSF algorithms for the QSM [GMR99] and BSP [Val90] models, which are more realistic models of parallel computation than the PRAM models. Theorem 8.1 given below follows directly from results mapping EREWcomputations on to QSM given in [GMR99]. Theorem 8.2 follows from the QSM to BSP emulation given in [GMR99] in conjunction with the observation that the slowdown in that emulation due to hashing does not occur for our algorithm since the assignment of vertices and edges to processors made by our processor allocation scheme achieves the same effect.

THEOREM 8.1. *An MSF of an edge-weighted graph on n nodes and m edges can be found in $O(g \log n)$ time and $O(g(m+n))$ work w.h.p. using $O(m+n)$ space on the QSM with a simple processor allocation scheme, where g is the gap parameter of the QSM.*

THEOREM 8.2. *An MSF of an edge-weighted graph on n nodes and m edges can be found on the BSP in $O((L+g)\log n)$ time w.h.p. using $(m+n)/\log n$ processors and $O(m+n)$ space with a simple processor allocation scheme, where g and L are the gap and periodicity parameters of the BSP.*

9. Conclusion. We have presented a randomized algorithm for MSF on the EREW PRAM which is provably optimal both in time and work. Our algorithm works within the stated bounds with high probability in the input size and has good performance in other popular parallel models.

One drawback to our algorithm is that it uses a linear number of random bits. A recent MSF algorithm [PR02a] for the EREW PRAM performs linear work but uses only a polylogarithmic number of random bits; however, the time required is suboptimal ($O(\log^2 n \log^* n)$). Unlike the algorithm presented here, the [PR02a] algorithm is not a parallelization of [KKT95] and does not use the sampling lemma from [KKT95].

An open question is how to obtain a *deterministic* time-work optimal MSF algorithm. Pettie and Ramachandran [PR02b] have given a provably optimal *sequential* MSF algorithm; however, its exact complexity (and therefore the complexity of MSF) is still unknown. Parallelizing this optimal sequential algorithm seems very difficult.

REFERENCES

[AS00] N. Alon and J. Spencer, *The Probabilistic Method*, 2nd ed., Wiley-Interscience, New York, 2000.

[Bor26] O. Borůvka, *O jistém problému minimaálním*, Moravské Přírodovědecké Společnosti, 3 (1926), pp. 37–58 (in Czech).

[CHL01] K. W. Chong, Y. Han, and T. W. Lam, *Concurrent threads and optimal parallel minimum spanning trees algorithm*, J. ACM, 48 (2001), pp. 297–323.

[CKT94] R. Cole, P. N. Klein, and R. E. Tarjan, *A linear-work parallel algorithm for finding minimum spanning trees*, in Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures (SPAA'94), Cape May, NJ, ACM, pp. 11–15.

[CKT96] R. Cole, P. N. Klein, and R. E. Tarjan, *Finding minimum spanning trees in logarithmic time and linear work using random sampling*, in Proceedings of the 8th Annual Symposium on Parallel Algorithms and Architectures (SPAA'96), Padua, Italy, ACM, pp. 243–250.

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[Gaz91] H. Gazit, *An optimal randomized parallel algorithm for finding connected components in a graph*, SIAM J. Comput., 20 (1991), pp. 1046–1067.

[GMR99] P. B. Gibbons, Y. Matias, and V. Ramachandran, *Can a shared-memory model serve as a bridging model for parallel computation?*, Theory Comput. Syst., 32 (1999), pp. 327–359.

[GK96] M. T. Goodrich and S. R. Kosaraju, *Sorting on a parallel pointer machine with applications to set expression evaluation*, J. ACM, 43 (1996), pp. 331–361.

[HZ96] S. Halperin and U. Zwick, *An optimal randomized logarithmic time connectivity algorithm for the EREW PRAM*, J. Comput. System Sci., 53 (1996), pp. 395–416.

[HZ01] S. Halperin and U. Zwick, *Optimal randomized EREW PRAM algorithms for finding spanning forests and for other basic graph connectivity problems*, J. Algorithms, 39 (2001), pp. 1–46.

[Jar30] V. Jarník, *O jistém problému minimaálním*, Moravské Přírodovědecké Společnosti, 6 (1930), pp. 57–63 (in Czech).

[JM97] D. B. Johnson and P. Metaxas, *Connected components in $O(\log^{3/2} n)$ parallel time for CREW PRAM*, J. Comput. System Sci., 54 (1997), pp. 227–242.

[K97] V. King, *A simpler minimum spanning tree verification algorithm*, Algorithmica, 18 (1997), pp. 263–270.

[KKT95] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, J. ACM, 42 (1995), pp. 321–328.

[KPRS97] V. King, C. K. Poon, V. Ramachandran, and S. Sinha, *An optimal EREW PRAM algorithm for minimum spanning tree verification*, Inform. Process. Lett., 62 (1997), pp. 153–159.

[KR90] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, Elsevier Science, Amsterdam, The Netherlands, 1990, pp. 869–941.

[PR97] C. K. Poon and V. Ramachandran, *A randomized linear work EREW PRAM algorithm to find a minimum spanning forest*, in Algorithms and Computation (Singapore, 1997), Lecture Notes in Comput. Sci. 1350, Springer-Verlag, Berlin, 1997, pp. 212–222.

[PR98] C. K. Poon and V. Ramachandran, *private communication*, 1998.

[PR02a] S. Pettie and V. Ramachandran, *Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, PA, 2002, pp. 713–722.

[PR02b] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, J. ACM, 49 (2002), pp. 16–34.

[Prim57] R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Technical J., 36 (1957), pp. 1389–1401.

[Val90] L. G. Valiant, *A bridging model for parallel computation*, Comm. ACM, 33 (1990), pp. 103–111.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1896–1908

LOWER BOUNDS FOR LUCAS CHAINS*

MARTIN KUTZ†

Abstract. Lucas chains are a special type of addition chains satisfying an extra condition: for the representation $a_k = a_j + a_i$ of each element a_k in the chain, the difference $a_j - a_i$ must also be contained in the chain. In analogy to the relation between addition chains and exponentiation, Lucas chains yield computation sequences for Lucas functions, a special kind of linear recurrences.

We show that the great majority of natural numbers n does not have Lucas chains shorter than $(1-\epsilon)\log_\phi n$ for any $\epsilon > 0$, where ϕ is the golden ratio.

Peter L. Montgomery was the first to consider Lucas chains, in the early eighties. He discovered a decomposition theorem for Lucas chains and a lower bound on their length in terms of Fibonacci numbers. His work was not published. Therefore several of Montgomery's original ideas are represented in this paper.

Key words. Lucas chain, addition chain, Lucas function, lower bound, Fibonacci number, golden ratio, smooth number

AMS subject classifications. 11Y55, 11Y16, 11B39, 11N25, 68Q25

PII. S0097539700379255

1. Introduction.

An increasing sequence $1 = a_0 < a_1 < \cdots < a_r = n$ of integers is called an *addition chain* for n if for each index $k \geq 1$ there exist $i \leq j < k$ so that

$$a_k = a_i + a_j. \tag{1}$$

This notion is motivated by the problem of computing x^n from x with few multiplications, so one is primarily interested in chains of small length r for given n. Since their first appearance in [12], addition chains have been intensively studied. See, for example, Schönhage's lower bound in [13] or Bergeron, Berstel, and Brlek's paper [1] on advanced methods for the construction of short addition chains. We refer to Section 4.6.3 of Knuth's classic [5] for a broader survey.

In this paper, we investigate *Lucas chains*, a variant of addition chains introduced by Peter L. Montgomery [9]. Those are chains for which the indices i, j in (1) can be chosen such that either $i = j$ or the difference $a_j - a_i$ is also part of the chain. The term "Lucas chain" is due to the observation that such chains yield computation sequences for Lucas functions, a special kind of linear recurrences.

Montgomery's paper [9], written in 1983, has never been published; for several years no further research was done on Lucas chains. This changed in 1993 when Smith and Lennon introduced the public-key crypto system *LUC* [14], which is based on Lucas functions. Yen and Laih [16] proposed Lucas chains as a means of evaluating the one-way functions of that crypto system; they used the term "Luc chains," though. Then in 1996 in his Ph.D. thesis on crypto systems [2], Bleichenbacher used results from [9] to actually compute short Lucas chains.

*Received by the editors October 9, 2000; accepted for publication (in revised form) June 30, 2002; published electronically October 31, 2002. This work was completed while the author was member of the European graduate school "Combinatorics, Geometry, and Computation" supported by the Deutsche Forschungsgemeinschaft, grant GRK 588/1. Most of this work is part of the author's diploma thesis [6].

http://www.siam.org/journals/sicomp/31-6/37925.html

†Mathematisches Institut II, Freie Universität Berlin, Arnimallee 3, 14195 Berlin, Germany (kutz@math.fu-berlin.de).

Besides some elaborate techniques for the construction of short Lucas chains, Montgomery [9] proved lower bounds on the length of Lucas chains for given integers. We will show similar results, reusing several of his ideas. From those bounds we will derive the more general statement that the majority of natural numbers n does not have Lucas chains shorter than $(1-\epsilon)\log_\phi n$ for any $\epsilon > 0$, where ϕ is the golden ratio. Two important prerequisites for this bound are a decomposition theorem stating that any Lucas chain can be uniquely factored into a product of so-called *simple* chains, and a lower bound on the length of these chains in terms of Fibonacci numbers.

The results of this paper are from the author's diploma thesis [6], written in ignorance of Montgomery's work. The author is grateful for Montgomery's kind permission to represent several ideas from his original work.

2. From Lucas functions to Lucas chains. Let P and Q be elements from a commutative ring with identity. The *Lucas functions* $V_n(P,Q)$ are defined recursively by [7]:

$$V_0(P,Q) = 2, \quad V_1(P,Q) = P, \quad V_{n+2}(P,Q) = P \cdot V_{n+1}(P,Q) - Q \cdot V_n(P,Q).$$

If α and β are the roots of the polynomial $X^2 - PX + Q$, then

$$P = \alpha + \beta, \quad Q = \alpha\beta, \quad \text{and} \quad V_n(P,Q) = \alpha^n + \beta^n. \tag{2}$$

In the following we will omit the arguments P, Q and simply write V_n.

We ask for a method to compute V_n for some $n \geq 0$ from a given pair P, Q. Looking at the identities

$$\begin{aligned} V_{m+n} &= (\alpha^m + \beta^m)(\alpha^n + \beta^n) - \alpha^n\beta^m - \alpha^m\beta^n \\ &= (\alpha^m + \beta^m)(\alpha^n + \beta^n) - \alpha^m\beta^m(\alpha^{n-m} + \beta^{n-m}) \\ &= V_m \cdot V_n - Q^m \cdot V_{n-m} \end{aligned} \tag{3}$$

for $0 \leq m \leq n$, we see that we can compute V_{m+n} from V_m, V_n, V_{n-m}, and a certain power of Q. This gives rise to the following definition.

DEFINITION 1. *A* Lucas chain *for an integer $n \geq 1$ is an increasing sequence*

$$1 = a_0 < a_1 < a_2 < \cdots < a_r = n$$

of integers such that for every $k \in \{1, \ldots, r\}$,

(L)	*there exist indices i, j with $0 \leq i \leq j < k$ such that $a_k = a_j + a_i$ and $a_j - a_i \in \{0, a_0, a_1, \ldots, a_{k-1}\}$.*

We call r the length *of the chain.*

Example 1. The sequence $(1,2,3,5)$ is a Lucas chain for 5 whereas $(1,2,4,5)$ is not—both are addition chains, though. In the latter sequence, 5 can only be represented as $4+1$ but $4-1=3$ is not part of the sequence.

Example 2. $(1,2,4,8,\ldots,2^l)$ is a Lucas chain of length l for 2^l. For every k, (L) is satisfied with $i = j = k-1$.

Example 3. Let the *Fibonacci numbers* F_n be recursively defined by

$$F_0 = 0, \quad F_1 = 1, \qquad \text{and} \qquad F_n = F_{n-1} + F_{n-2}, \quad n \geq 2.$$

For any $l \geq 0$ the sequence $\mathcal{F}_l = (F_2, F_3, F_4, F_5, \ldots, F_{l+2})$ is a Lucas chain of length l for F_{l+2}. To show (L), let $j = k-1$ and $i = k-2$; then $F_j - F_i = F_{k-3}$. We call $\mathcal{F}_l$ the *lth Fibonacci chain.*

A Lucas chain for n directly yields a computation of V_n. First, we successively compute Q^{a_k} for $k = 1, \ldots, r-1$ with $r-1$ multiplications. Second, we repeatedly use (3) to obtain V_{a_k} for $k = 1, \ldots, r$, which takes two multiplications in each step. Thus we can compute V_n from P and Q with $3r-1$ multiplications altogether.

3. Applications. As stated in the introduction, Lucas chains turned out to be useful in public-key cryptography. Instead of using powers $X^n \bmod N$ with some large integer N for the one-way function as in the RSA scheme [11], the LUC crypto system [14] uses the Lucas function $V_n \bmod N$ for encryption and decryption. In this application, the parameter Q is always chosen to be 1 so that powers of Q need not be computed and (3) simplifies to $V_{m+n} = V_m \cdot V_n - V_{n-m}$ (see also [16]). Hence in this special case, a Lucas chain of length r for n yields a computation of V_n with exactly r multiplications in $\mathbb{Z}/N\mathbb{Z}$.

The LUC crypto system stimulated research on Lucas chains [2, 16], but the use of Lucas functions for public-key cryptography had been considered before: Müller and Nöbauer [10] proposed the *Dickson polynomials* $g_n(x, a)$, given by

$$g_n(x,a) = \sum_{0 \le j \le n/2} \frac{n}{n-j} \binom{n-j}{j} (-a)^j x^{n-2j} \qquad \text{for} \quad n \ge 1$$

and $g_0(x, a) = 2$ with some a from a commutative ring with identity, as one-way functions. Waring's formula tells us that [8, p. 355]

$$g_n(P,Q) = \alpha^n + \beta^n = V_n(P,Q)$$

with α, β as in (2). Hence, from this point of view, the LUC system uses the Dickson polynomials $g_n(x, 1)$ for encryption. Müller and Nöbauer argued that it might be difficult to efficiently compute Dickson polynomials for large n, but von zur Gathen [15] used the equation $g_{n+2} = xg_{n+1} - ag_n$ and a matrix representation of this recurrence relation to show that g_n can be computed in $\mathcal{O}(\log n)$ ring operations. Lucas chains can thus be seen as a tool to reduce the multiplicative constant in this asymptotic expression.

Montgomery [9] already pointed out another application of Lucas chains: For every $n \in \mathbb{N}$, the *nth Chebyshev polynomial* T_n is defined as the unique polynomial satisfying

$$T_n(\cos z) = \cos(nz).$$

If we let $\alpha = e^{iz}$ and $\beta = e^{-iz}$, we get $\cos z = (\alpha^n + \beta^n)/2$ and $P = 2\cos z$, $Q = 1$ by (2). Writing $x = \cos z$ then yields [8, p. 355]

$$2T_n(x) = 2\cos(nz) = g_n(2x, 1) = V_n(2x, 1).$$

In other words, these polynomials are just a special case of the Dickson polynomials over the complex numbers. Again we see that we can compute $T_n(x)$ from x with r multiplications if we have a Lucas chain of length r for n.

4. Basic structural properties. For technical reasons we extend the usual notion of Lucas chains to what we call "prechains." They do not alter the capabilities of the original chains but simplify our arguments and proofs in this section.

DEFINITION 2. *A strictly monotonically increasing sequence $\chi = (a_0, a_1, \ldots, a_r)$ of positive integers is called a* Lucas prechain *if property* (L) *holds for every $k \in$*

$\{1,\dots,r\}$. *We define* $\operatorname{len}(\chi) := r$ *to be the* length *and* $\operatorname{val}(\chi) := a_r/a_0$ *to be the* value *of the prechain. A prechain of length zero is called* trivial.

Obviously, every Lucas chain for some n is a Lucas prechain with value n. Conversely, it is also easy to see that a prechain is nothing but a scaled Lucas chain. Let $\alpha\chi$ denote the sequence $(\alpha a_0,\dots,\alpha a_r)$, $\alpha \in \mathbb{Q}_+$. Property (L) is obviously invariant under such scalar multiplications; i.e., a sequence χ of integers is a Lucas prechain iff $c\chi$ is, $c \in \mathbb{N}_+$. Also note that the first element of a prechain divides all others. This follows by induction since (L) implies that a_0 divides a_k if it divides a_i and a_j. Thus we see that values of Lucas prechains are always integers, and scalar multiplication with $1/a_0$ makes any prechain into a chain without changing length or value.

Example 4. The sequence $(6, 12, 18, 30, 48)$ is a Lucas prechain of length 4 for 8. Multiplication with 1/6 yields the chain $(1, 2, 3, 5, 8)$.

4.1. Multiplying chains. It is a well-known fact that two addition chains for a and b can be combined to yield an addition chain for ab [5, Sect. 4.6.3]. As shown in the proof of Theorem 2 of [9], the same method applies to Lucas chains. We restate it here in terms of prechains.

DEFINITION 3. *Let* $\chi_1 = (a_0,\dots,a_r)$ *and* $\chi_2 = (b_0,\dots,b_s)$ *be Lucas prechains with* $a_r = b_0$. *Then their* composition $\chi_1 \circ \chi_2$ *is defined as the sequence* $(a_0,\dots,a_{r-1}, b_0,\dots,b_s)$.

We see that the composition of two Lucas prechains is also a Lucas prechain because no element is removed from the original sequences. This concept is easily adapted to chains. We simply have to scale the second chain appropriately.

DEFINITION 4. *Let* χ_1 *and* χ_2 *be Lucas chains. Then their* product $\chi_1 * \chi_2$ *is the Lucas chain* $\chi_1 \circ \operatorname{val}(\chi_1)\chi_2$.

Both operations, $\circ$ and $*$, clearly are associative. The following equations are immediate from the definitions:

$$\operatorname{len}(\chi_1 * \chi_2) = \operatorname{len}(\chi_1) + \operatorname{len}(\chi_2), \tag{4}$$

$$\operatorname{val}(\chi_1 * \chi_2) = \operatorname{val}(\chi_1) \cdot \operatorname{val}(\chi_2). \tag{5}$$

In order to obtain a chain for a composite number $n = ab$, we can thus multiply chains χ_1 and χ_2 for a and b, respectively. As with addition chains, we call this technique the *factor method* [5, p. 463].

Example 5. The product chain $(1, 2, 3) * (1, 2, 3, 5, 7) * (1, 2, 4) = (1, 2, 3, 6, 9, 15, 21, 42, 84)$ is a Lucas chain of length $2 + 4 + 2 = 8$ for $3 \cdot 7 \cdot 4 = 84$.

4.2. Decomposing chains. We shall now identify those chains that can be written as products of smaller chains. The following proposition is essentially equivalent to Theorem 5 and Corollary 6 of [9]. But because of its fundamental importance for the understanding of the structure of Lucas chains, we shall prove it here again, in terms of prechains. It states that for Lucas chains, decomposability is a completely local concept.

PROPOSITION 1. *Let* $\chi = (a_0,\dots,a_r)$ *be a Lucas prechain and let* $0 \le m < r$. *Then the following two statements are equivalent:*

(i) $a_{m+1} = 2a_m$,

(ii) $\chi^{(m)} := (a_m, a_{m+1},\dots,a_r)$ *is a Lucas prechain.*

Proof. Assume that (i) holds. We show by induction on $k = m+1,\dots,r$ that a_m divides a_k and that any pair of indices i, j satisfying (L) fulfills

$$m \le i \le j \qquad \text{and} \qquad a_j - a_i \in \{0, a_m,\dots,a_{k-1}\}.$$

For $k = m+1$, property (L) is satisfied with $i = j = m$ only. So let $k > m+1$ and assume that $a_m | a_l$ for $m \le l < k$. Let i, j as in (L). We have the implications

$$\begin{aligned} k > m+1 \;&\Rightarrow\; a_k > a_{m+1} = 2a_m \;\Rightarrow\; a_j > a_m \\ &\Rightarrow\; a_j \ge a_{m+1} = 2a_m \;\Rightarrow\; j \ge m+1, \end{aligned}$$

hence $a_m | a_j$ by induction. Let us now consider the two cases $a_i \ge a_j/2$ and $a_i < a_j/2$. In the first case we get

$$a_i \ge a_j/2 \;\Rightarrow\; a_i \ge a_m \;\Rightarrow\; i \ge m \;\Rightarrow\; a_m | a_i.$$

Otherwise

$$a_i < a_j/2 \;\Rightarrow\; a_j - a_i > a_j/2 \ge a_m \;\Rightarrow\; a_m | (a_j - a_i);$$

combined with $a_m | a_j$ this yields $a_m | a_i$ and thus $i \ge m$. In both cases a_m divides $(a_j + a_i) = a_k$ as was to be shown.

The implication (ii) $\Rightarrow$ (i) is immediate from (L). □

Such a pair (a_m, a_{m+1}) of consecutive elements with $a_{m+1} = 2a_m$ is called a *doubling step* [5, p. 467] of the prechain χ. Note that the positions of doubling steps in a prechain are obviously invariant under scaling, i.e., (a, b) is a doubling step of χ iff (ca, cb) is a doubling step of $c\chi$. It is now very easy to identify those chains that are not representable as products.

DEFINITION 5. *We call a Lucas prechain* simple *if it contains exactly one doubling step—its first two elements.*

The Fibonacci chains $\mathcal{F}_l$ from Example 3 are simple for every $l \ge 1$. Note that by definition, trivial prechains are not simple.

The term *simple* is due to Bleichenbacher [2, Chap. 5]. He observed that Lucas chains that cannot be written as nontrivial products are simple. For our lower bounds in section 6, we need to make this notion a little more precise.

PROPOSITION 2. *Let $\chi = (a_0, \ldots, a_r)$ be a nontrivial Lucas prechain and let $(a_{r_\mu}, a_{r_\mu+1})$, $1 \le \mu \le d$, be all its doubling steps in increasing order, i.e., $1 = a_{r_1} < a_{r_2} < \cdots < a_{r_d}$. Additionally let $r_{d+1} := r$. Then*

$$\chi_\mu := (a_{r_\mu}, a_{r_\mu+1}, \ldots, a_{r_{\mu+1}})$$

is a simple Lucas prechain for every $\mu \in \{1, \ldots, d\}$. We have $\chi = \chi_1 \circ \ldots \circ \chi_d$ and this decomposition into simple prechains is unique.

Proof. By Proposition 1, every χ_μ is a Lucas prechain because they all start with a doubling step. They are also simple because none of them contains more doubling steps. The equation $\chi = \chi_1 \circ \cdots \circ \chi_d$ is immediate from the definition of the χ_μ. For uniqueness, just observe that by Proposition 1 every prechain χ'_μ of a decomposition into simple prechains has to start with a doubling step of χ. To be simple it must not contain any other of χ's doubling steps. And it is also of strictly positive length since trivial prechains are not simple. □

We can directly restate this result for chains. Defining the empty product to be the trivial chain, we get the following theorem, which very much resembles the proof of Theorem 7 in [9].

THEOREM 1. *Every Lucas chain χ has a unique decomposition $\chi = \chi_1 * \cdots * \chi_d$ into simple chains. This induces a factorization*

$$\operatorname{val}(\chi) = \prod_{\mu=1}^{d} \operatorname{val}(\chi_\mu)$$

of its value.

Proof. The statement about the simple chain decomposition is just a reformulation of Proposition 2. The product formula for the values directly follows from equation (5). □

Consequently, the structure of a Lucas chain for some $n \in \mathbb{N}_+$ is intimately related to the prime number factorization of n. Theorem 1 will be very important for our lower bounds in section 6.

5. Trivial bounds. Since Lucas chains resemble computation sequences, we are interested in shortest chains for a given value.

DEFINITION 6. *For every number $n \in \mathbb{N}_+$ we let*

$$t(n) := \min\{\mathrm{len}(\chi) \mid \chi \text{ is a Lucas chain for } n\},$$
$$t'(n) := \min\{\mathrm{len}(\chi) \mid \chi \text{ is a simple Lucas chain for } n\}.$$

A Lucas chain χ is called optimal *if* $\mathrm{len}(\chi) = t(\mathrm{val}(\chi))$.

Note that $t'(1) = \infty$ because any simple chain has value ≥ 2; this will be inconsequential since we shall consider $t'(n)$ for $n \geq 2$ only.

We can already state some basic facts about the function t. Application of the factor method directly yields [9, Thm. 2]

$$t(a \cdot b) \leq t(a) + t(b). \tag{6}$$

Just choose optimal chains χ_1 and χ_2 for a and b, respectively, and compare (4) and (5). Denoting $\log_2$ by lg as usual, we also have the trivial lower bound

$$t(n) \geq \lceil \lg n \rceil \tag{7}$$

because by property (L), no element of a Lucas chain can be more than twice as big as any of its predecessors. Therefore the chains in Example 2 are obviously optimal.

5.1. A known upper bound. Montgomery developed the following *binary method* [9] for the construction of a Lucas chain for any given odd $n \geq 3$.

Let $d_0, d_1, \ldots, d_k$ be the digits in the binary representation of n, starting from the high end. We let $a_0 := d_0 = 1$ and inductively define

$$a_i = 2a_{i-1} + d_i \qquad \text{for} \quad i = 1, \ldots, k.$$

In other words, a_i has the binary representation $d_0 d_1 \ldots d_i$. Then

$$(a_0, a_0 + 1, a_1, a_1 + 1, \ldots, a_{k-1}, a_{k-1} + 1, a_k)$$

is a Lucas chain for n because the elements a_{i+1} and $a_{i+1} + 1$ can always be written as $2a_i$, $a_i + (a_i + 1)$, or $2(a_i + 1)$ so that the respective differences are either 0 or 1. This chain has no more than $2k + 1$ elements. Thus we get the upper bound

$$t(n) \leq 2\lfloor \lg n \rfloor. \tag{8}$$

By application of the factor method to $a = n/2$ and $b = 2$, this bound also carries over to even n.

Example 6. For $n = 37 = 100101_2$, the binary method yields the Lucas chain $(1, 2, 3, 4, 5, 9, 10, 18, 19, 37)$.

6. Lower bounds. It turns out that the trivial bound (7) can be substantially improved upon. Montgomery showed the following [9, Thm. 7].

THEOREM 2 (Montgomery). *Let n be a positive integer with s prime divisors (including multiplicities). Then the number of doubling steps in a Lucas chain for n cannot exceed s, and $n \leq 2^{s-1} F_{t(n)-s+3}$.*

This gives us a lower bound on $t(n)$ for any $n \in \mathbb{N}_+$. The aim of this section is to derive a similar bound that depends on the exact prime number factorization of n and not only on the number of prime factors. That result will then enable us to prove the desired lower bound of $t(n) \geq (1-\epsilon)\log_\phi n$ for the vast majority of integers.

6.1. A lower bound for simple chains. By definition, a simple Lucas chain contains exactly one doubling step. Since these are the most efficient steps available, we expect simple chains to grow notably slower than arbitrary Lucas chains can. The following simple but important lemma, which in different form already appeared in [9], captures this intuition.

LEMMA 1. *Let χ be a simple Lucas chain. Then its value is bounded by*

$$\operatorname{val}(\chi) \leq F_{\operatorname{len}(\chi)+2}.$$

Proof. Since every Lucas chain is also an addition chain, we may apply Theorem A from [5, p. 467]. Letting $d = 1$ there immediately yields the stated inequality. ∎

In order to rephrase Lemma 1 in terms of the function t', we reinterpret it in the following way: *The average growth of a simple Lucas chain of length k is no more than a factor of $\sqrt[k]{F_{k+2}}$ per step.*

DEFINITION 7. *For any integer $n \geq 2$, let $k := \min\{l \mid n \leq F_{l+2}\}$ and define*

$$\Phi(n) := \sqrt[k]{F_{k+2}}.$$

Indeed, this is a useful notion. We obtain the following bound.

PROPOSITION 3. *For every $n \geq 2$, we have*

$$t'(n) \geq \frac{\lg n}{\lg \Phi(n)}.$$

Proof. Let k be the unique integer that satisfies $F_{k+1} < n \leq F_{k+2}$. Then by Lemma 1, any simple chain for n has length at least k. Thus

$$t'(n) \geq k \geq \frac{\lg n}{\lg F_{k+2}}\, k = \frac{\lg n}{\lg \sqrt[k]{F_{k+2}}} = \frac{\lg n}{\lg \Phi(n)}. \qquad \blacksquare$$

6.2. From simple chains to arbitrary chains. Theorem 1 now tells us how to obtain a bound for $t(n)$ from Proposition 3. Any chain for n can be factored into simple chains, and the values of these factors are restricted by the possible (partial) factorizations of the integer n. Hence, we can apply Proposition 3 to all those factorizations and get

$$t(n) \geq \min\left\{ \sum_{i=1}^{d} \frac{\lg f_i}{\lg \Phi(f_i)} \;\middle|\; \prod_{i=1}^{d} f_i = n \right\}. \tag{9}$$

We can already use this formula to achieve nontrivial bounds on $t(n)$ for certain values of n.

Example 7. The prime factors of $n = 85$ are 5 and 17. Proposition 3 yields $t'(5) \geq 3$, $t'(17) > 5.5$, and $t'(85) > 8.9$. Since $(1,2,3,5) * (1,2,3,5,7,10,17)$ is a chain of length 9 for 85, we obtain $t(85) = 9$.

So this technique yields useful results, but it may become impractical due to the combinatorial explosion in cases where n has many factors. In the following we shall see how this drawback can be overcome. The key observation is that short simple chains are potentially more efficient than long ones because the shorter the chain the greater the effect of the initial doubling step on the average growth of the chain. In fact, Proposition 3 already captures this behavior in a very satisfying way. We shall see that it is sufficient to apply it only to the prime number factorization of n. In order to prove this formally, we need some basic facts about the function Φ.

6.3. Properties of the function Φ. The Fibonacci numbers are closely related to the golden ratio

$$\phi = \frac{1+\sqrt{5}}{2},$$

and we have the well-known formula [4, p. 83]

$$F_k = \frac{1}{\sqrt{5}}\left(\phi^k - \hat{\phi}^k\right),$$

where $\hat{\phi} = 1 - \phi = \frac{1}{2}(1-\sqrt{5})$. Since $\hat{\phi} = -\phi^{-1}$, we can restate this as

$$F_k = \frac{1}{\sqrt{5}}\left(\phi^k - (-\phi)^{-k}\right), \tag{10}$$

which will better suit our needs.

LEMMA 2. *For all $k \geq 1$ we have*

$$\sqrt[k+1]{F_{k+3}} < \sqrt[k]{F_{k+2}}.$$

Proof. We first raise both sides of the inequality to the $k(k+1)$st power and apply (10); thus we get

$$\begin{aligned}
&\left[\tfrac{1}{\sqrt{5}}\left(\phi^{k+3} - (-\phi)^{-k-3}\right)\right]^k < \left[\tfrac{1}{\sqrt{5}}\left(\phi^{k+2} - (-\phi)^{-k-2}\right)\right]^{k+1} \\
&\Leftrightarrow \sqrt{5}\,\left[\left(1 + (-1)^k\phi^{-2k-6}\right)\,\phi^{k+3}\right]^k < \left[\left(1 + (-1)^{k+1}\phi^{-2k-4}\right)\,\phi^{k+2}\right]^{k+1} \\
&\Leftrightarrow \sqrt{5}\,\phi^{-2} < \left(1 + (-1)^{k+1}\phi^{-2k-4}\right)^{k+1}\left(1 + (-1)^k\phi^{-2k-6}\right)^{-k}. \qquad (11)
\end{aligned}$$

Numerical computation of the left-hand side of (11) yields $\sqrt{5}\,\phi^{-2} < 0.86$. If k is odd, the right-hand side is greater than one and hence (11) follows. If k is even, the right-hand side of (11) equals

$$\begin{aligned}
&\left(1 - \phi^{-2k-4}\right)^{k+1}\left(1 + \phi^{-2k-6}\right)^{-k} \\
&\quad > \left(1 - \phi^{-2k-4}\right)^{k+1}\left(1 - \phi^{-2k-6}\right)^{k} \\
&\quad > \left(1 - \phi^{-2k-4}\right)^{2k+1} \\
&\quad > 1 - (2k+1)\,\phi^{-2k-4},
\end{aligned}$$

where the last step is by application of the Bernoulli inequality. Now

$$h(x) := 1 - (2x+1)\,\phi^{-2x-4} > \sqrt{5}\,\phi^{-2} \qquad \text{for} \quad x \geq 2$$

since numerical computation for $x = 2$ yields $1 - 5\,\phi^{-8} > 0.89 > \sqrt{5}\,\phi^{-2}$ and

$$h'(x) = \big((2x+1)\ln\phi^2 - 2\big)\phi^{-2x-4} > 0 \qquad \text{for} \quad x \geq 2. \qquad \square$$

LEMMA 3. *The sequence $(\Phi(n))_{n\geq 2}$ is monotonically decreasing. It converges towards the golden ratio:*

$$\lim_{n\to\infty} \Phi(n) = \phi.$$

Proof. The first statement is a direct consequence of Lemma 2. Equation (10) yields

$$\sqrt[k]{F_{k+2}} = \sqrt[k]{\tfrac{1}{\sqrt{5}}\big(\phi^{k+2} - (-\phi)^{-k-2}\big)} = \phi \cdot \sqrt[k]{\tfrac{\phi^2}{\sqrt{5}}\big(1 \pm \phi^{-2k-4}\big)},$$

and so the second statement follows. $\square$

6.4. The lower bound in closed form. We are now able to prove a lower bound for $t(n)$ that does not suffer from the combinatorial explosion of the rule (9).

THEOREM 3. *Let n be any positive integer, and let $n = \prod_1^e p_i$ be its factorization into prime numbers. Then we have*

$$t(n) \geq \sum_1^e \frac{\lg p_i}{\lg \Phi(p_i)}.$$

Proof. Let χ be an optimal chain for n and let $\chi_1 * \cdots * \chi_d$ be its decomposition into simple chains. We let $n_\mu := \mathrm{val}(\chi_\mu)$ be their corresponding values. Since $n = n_1 \cdots n_d$, there exists a partition $I_1, \ldots, I_d$ of the index set $\{1, \ldots, e\}$ so that

$$n_\mu = \prod_{i\in I_\mu} p_i$$

for every $\mu \in \{1, \ldots, d\}$. Since χ is optimal, every χ_μ must also be optimal. Thus, by (4) and Proposition 3 we have

$$\begin{aligned} t(n) &= \sum_{\mu=1}^d t(n_\mu) = \sum_{\mu=1}^d t'(n_\mu) \\ &\geq \sum_{\mu=1}^d \frac{\lg n_\mu}{\lg \Phi(n_\mu)} = \sum_{\mu=1}^d \sum_{i\in I_\mu} \frac{\lg p_i}{\lg \Phi(n_\mu)} \\ &\geq \sum_{\mu=1}^d \sum_{i\in I_\mu} \frac{\lg p_i}{\lg \Phi(p_i)} = \sum_{i=1}^e \frac{\lg p_i}{\lg \Phi(p_i)}, \end{aligned}$$

where the penultimate step makes use of Lemma 3. $\square$

Theorem 3 is a powerful and also practical tool for proving Lucas chains optimal. Let us again consider the chain from Example 5. The prime number factorization is

$84 = 2^2 \cdot 3 \cdot 7$. We have

$$\begin{aligned} t(84) &\geq 2\,\frac{\lg 2}{\lg \Phi(2)} + \frac{\lg 3}{\lg \Phi(3)} + \frac{\lg 7}{\lg \Phi(7)} \\ &= 2\,\frac{\lg 2}{\lg 2} + \frac{\lg 3}{\lg \sqrt{3}} + \frac{\lg 7}{\lg \sqrt[4]{8}} > 7.7, \end{aligned}$$

and thus $t(84) = 8$.

6.5. Comparison of the bounds. Since we used methods similar to those of Montgomery, the bounds from Theorem 3 are close to those from Theorem 2. For example, the latter also yields $t(84) \geq 8$, as we have computed from the former. But there are also cases in which the former is slightly better than the latter. As an example consider $n = 177$, where we get the lower bounds 11, which is the precise value of $t(177)$, and 10, respectively. The main advantage of Theorem 3, however, is its dependence on the prime factors of n and its implicit relation to the golden ratio through the function Φ. This will enable us to derive the general lower bound in the subsequent section.

7. A general lower bound. By now we have considered only concrete lower bounds for individual values. In this section we are going to show that the great majority of numbers n does not have Lucas chains shorter than $(1-\epsilon)\log_\phi n$ for any given $\epsilon > 0$.

For this, observe that our bound from Theorem 3 is closer to $\log_\phi n$ if n contains many large prime factors. The next definition captures this notion.

DEFINITION 8. *Let n be any positive integer with prime number factorization $n = \prod_1^e p_i$. Let $B \in \mathbb{N}_+$ and $\delta \in (0, 1]$. We call n a (B, δ)-*fat *number if*

$$\prod_{p_i \leq B} p_i < n^\delta,$$

*that is, it contains less than a δ-portion (logarithmically) of factors smaller than B. We call n (B, δ)-*smooth *if it is not (B, δ)-fat.*

The term "smooth number" is generally used for integers that contain no prime factors larger than a certain bound B. Note that this is just the special case $\delta = 1$ in the above definition.

As expected, it turns out that fat numbers cannot have short Lucas chains.

LEMMA 4. *Let n be a (B, δ)-fat integer, $B \geq 2$. Then we have*

$$t(n) \geq \frac{1-\delta}{\lg \Phi(B)} \lg n.$$

Proof. Let $\prod_1^e p_i$ be the factorization of n into prime numbers, and let

$$\begin{aligned} S &:= \{i \mid p_i \leq B\}, \\ L &:= \{i \mid p_i > B\} \end{aligned}$$

denote the collection of small and large factors indices, respectively. By Theorem 3 we have

$$\begin{aligned} t(n) &\geq \sum_{i \in S \cup L} \frac{\lg p_i}{\lg \Phi(p_i)} \geq \sum_{i \in L} \frac{\lg p_i}{\lg \Phi(p_i)} \\ &\geq \sum_{i \in L} \frac{\lg p_i}{\lg \Phi(B)} = \frac{1}{\lg \Phi(B)} \lg \prod_{i \in L} p_i \\ &> \frac{\lg n^{1-\delta}}{\lg \Phi(B)} = \frac{1-\delta}{\lg \Phi(B)} \lg n, \end{aligned}$$

where the last line follows from the (B, δ)-fatness of n. Note that we had to exclude $B = 1$ since $\Phi(1)$ is not defined. □

We want to know how many numbers are not of this kind; that is, how frequent are (B, δ)-smooth numbers for given B and δ? While much is known about the frequency of the "ordinary" smooth numbers with $\delta = 1$ (see, for example, [3]), our relaxed notion of smoothness has not yet been investigated. The following lemma gives us satisfactory estimates on the density of (B, δ)-smooth numbers. We let $[N, M] := \{n \in \mathbb{Z} \mid N \leq n \leq M\}$ denote the set of integers between N and M, and let $\pi(x)$ as usual count the numbers of primes less than or equal to x.

LEMMA 5. *For every bound $B \in \mathbb{N}_+$, every $\delta \in (0, 1)$, and every $N \in \mathbb{N}_+$, the interval $[N, 2N - 1]$ contains fewer than*

$$\left(N^{1-\delta} + 1\right) \binom{\delta \lg N + \lg B + \pi(B)}{\pi(B)}$$

(B, δ)-smooth numbers.

Proof. Let n be any (B, δ)-smooth number from the interval $[N, 2N - 1]$, and let $\prod_1^e p_i$ be its factorization into prime numbers. Let $S := \{i \mid p_i \leq B\}$ be the collection of its small factors indices. Then we have

$$\prod_{i \in S} p_i \geq n^\delta \geq N^\delta$$

since n is (B, δ)-smooth. We can successively remove indices from S to obtain a subset $S' \subseteq S$ that satisfies

$$N^\delta \leq f := \prod_{i \in S'} p_i < BN^\delta.$$

Hence, every (B, δ)-smooth number in the interval $[N, 2N - 1]$ has such a divisor f. Any such f is of the form

$$f = 2^{\sigma_2} 3^{\sigma_3} 5^{\sigma_5} \dots p^{\sigma_p},$$

where p is the greatest prime less than or equal to B. Since $f < BN^\delta$, all of the σ's are less than $\lg(BN^\delta)$. Thus, there are fewer than

$$\binom{\lg(BN^\delta) + \pi(B) - 1}{\pi(B) - 1} < \binom{\delta \lg N + \lg B + \pi(B)}{\pi(B)}$$

such f's. Every single $f \geq N^\delta$ divides no more than $N/N^\delta + 1$ numbers in the range $[N, 2N - 1]$, and hence the statement of the lemma follows. □

Now we are prepared to prove the announced asymptotic lower bound for Lucas chains.

THEOREM 4. *For any $\epsilon > \rho > 0$ and increasing $N \in \mathbb{N}_+$, there are only $\mathcal{O}(N^{1-\rho})$ numbers $n \in [N, 2N-1]$ satisfying*

$$t(n) \le (1-\epsilon)\log_\phi n,$$

where the constants hidden in the $\mathcal{O}$ depend on ϵ and ρ.

Proof. Let $\delta := (\epsilon+\rho)/2$ and choose an integer B so that

$$\lg \Phi(B) \le \frac{\lg \phi}{1-\frac{\epsilon-\rho}{2}};$$

by Lemma 3 such a B exists. Now Lemma 4 yields

$$t(n) > \frac{1-\delta}{\lg \Phi(B)} \lg n \ge (1-\delta)\Big(1-\frac{\epsilon-\rho}{2}\Big)\log_\phi n > (1-\epsilon)\log_\phi n$$

for any (B,δ)-fat integer n. Thus, only (B,δ)-smooth numbers can have shorter chains, but by Lemma 5 there are no more than

$$\big(N^{1-\delta}+1\big)\binom{\delta \lg N + \lg B + \pi(B)}{\pi(B)} \in \mathcal{O}\big(N^{1-\rho}\big)$$

of these in any interval $[N, 2N-1]$. □

8. Final remarks. Since $\log_\phi n \approx 1.44 \lg n$, Theorem 4 is a significant improvement on the trivial bound (7). But we may ask how close this comes to the optimum. Examples 5 and 7 show that there are cases in which our bounds are extremely sharp. Yet, the majority of numbers could still need chains much longer than Theorem 4 indicates.

We strongly believe that this is not the case. Comparison of the concrete bounds from Theorem 3 with heuristic computations of short chains for the first million natural numbers suggests that our bound is very sharp. It turned out that for all $n \le 10^6$ we have

$$t(n) \le \left\lceil \sum_1^e \frac{\lg p_i}{\lg \Phi(p_i)} \right\rceil + 2,$$

where $n = \prod_1^e p_i$ is the prime number factorization of n.

It seems that Theorem 4 already captures the behavior of the function t in a most fundamental way.

CONJECTURE 1. *The length function t satisfies*

$$\limsup \frac{t(n)}{\lg n} = \frac{1}{\lg \phi}.$$

In fact, Montgomery already asked in Problem 2 of [9] whether this upper bound holds. And in a private communication Donald E. Knuth conjectured the slightly stronger bound of $t(n) \le \log_\phi n + \mathcal{O}(1)$. Nevertheless, I do not expect Conjecture 1 to be shown in the near future. Though the heuristics have produced good results, it seems to be very hard to actually prove any significantly better upper bound than (8). By now we are not even able to show $t(n) \le \alpha \lg n$ for any $\alpha < 2$.

Acknowledgments. I thank my editor Joachim von zur Gathen for pointing me to Montgomery's paper and for many constructive hints. And I am very grateful for Peter L. Montgomery's permission to represent his results. I also want to thank Donald E. Knuth, who supplied me with a list of precise values of $t(n)$, for his encouragement. Above all, I am indebted to my teacher Arnold Schönhage for his friendly support and valuable advice.

REFERENCES

[1] F. Bergeron, J. Berstel, and S. Brlek, *Efficient computation of addition chains*, J. Théor. Nombres Bordeaux, 6 (1994), pp. 21–38.
[2] D. Bleichenbacher, *Efficiency and Security of Cryptosystems Based on Number Theory*, Ph.D. thesis, Swiss Federal Institute of Technology, Zürich, 1996.
[3] A. Hildebrand, *On the number of positive integers $\leq x$ and free of prime factors $> y$*, J. Number Theory, 22 (1986), pp. 298–307.
[4] D. E. Knuth, *The Art of Computer Programming*, Vol. 1, 3rd ed., Addison–Wesley, Reading, MA, 1997.
[5] D. E. Knuth, *The Art of Computer Programming*, Vol. 2, 3rd ed., Addison–Wesley, Reading, MA, 1998.
[6] M. Kutz, *Grundlegende Betrachtungen zu einer Variante von Additionsketten*, Diploma thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany, 2000 (in German).
[7] D. H. Lehmer, *An extended theory of Lucas' functions*, Ann. of Math., 31 (1930), pp. 419–448.
[8] R. Lidl and H. Niederreiter, *Finite Fields*, Addison–Wesley, Reading, MA, 1983.
[9] P. L. Montgomery, *Evaluating Recurrences of Form $X_{m+n} = f(X_m, X_n, X_{m-n})$ via Lucas Chains*, manuscript, 1992. Available via ftp from ftp://ftp.cwi.nl/pub/pmontgom/Lucas.ps.gz.
[10] W. B. Müller and W. Nöbauer, *Some remarks on public-key cryptosystems*, Studia Sci. Math. Hungar., 16 (1981), pp. 71–76.
[11] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.
[12] A. Scholz, *Aufgabe* 253, in Jahresbericht der Deutschen Mathematikervereinigung, Vol. 47, Teil II, B. G. Teubner, Leipzig and Berlin, 1937, pp. 41–42 (in German).
[13] A. Schönhage, *A lower bound for the length of addition chains*, Theoret. Comput. Sci., 1 (1975), pp. 1–12.
[14] P. J. Smith and M. J. J. Lennon, *LUC: A new public key system*, in Proceedings of the 9th IFIP International Symposium on Computer Security, E. G. Dougall, ed., Elsevier, New York, 1993, pp. 97–110.
[15] J. von zur Gathen, *Tests and Algorithms for Permutation Polynomials*, Tech. report, Department of Computer Science, The Australian National University, Canberra, 1989.
[16] S.-M. Yen and C.-S. Laih, *Fast algorithms for LUC digital signature computation*, IEE Proceedings - Computers and Digital Techniques, 142 (1995), pp. 165–169.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1909–1925

SIMPLE LEARNING ALGORITHMS FOR DECISION TREES AND MULTIVARIATE POLYNOMIALS*

NADER H. BSHOUTY† AND YISHAY MANSOUR‡

Abstract. In this paper we develop a new approach for learning decision trees and multivariate polynomials via interpolation of multivariate polynomials. This new approach yields simple learning algorithms for multivariate polynomials and decision trees over finite fields under *any* constant bounded product distribution. The output hypothesis is a (single) multivariate polynomial that is an ϵ-approximation of the target under *any* constant bounded product distribution.

The new approach demonstrates the learnability of many classes under any constant bounded product distribution and using membership queries, such as j-disjoint disjunctive normal forms (DNFs) and multivariate polynomials with bounded degree over any field.

The technique shows how to interpolate multivariate polynomials with bounded term size from membership queries only. This, in particular, gives a learning algorithm for an $O(\log n)$-depth decision tree from membership queries only and a new learning algorithm of *any* multivariate polynomial over sufficiently large fields from membership queries only. We show that our results for learning from membership queries only are the best possible.

Key words. learning interpolation, multivariate polynomial, decision tree learning

AMS subject classifications. 68Q32, 68Q25

PII. S009753979732058X

1. Introduction.

From the start of computational learning theory, great emphasis has been put on developing algorithmic techniques for various problems. It seems that great progress has been made in learning using membership queries, especially such functions as decision trees and multivariate polynomials. Generally speaking, three different techniques were developed for those tasks: the Fourier transform technique, the lattice-based techniques, and the multiplicity automata technique. All of the techniques use membership queries (which are also called substitution queries for nonbinary fields).

The Fourier transform technique is based on representing functions using a basis, where a basis function is essentially a parity of a subset of the input. Any function can be represented as a linear combination of the basis functions. Kushilevitz and Mansour [KM93] gave a general technique to recover the significant coefficients. They showed that this is sufficient for learning decision trees under the uniform distribution. Jackson [J94] extended the result to learning disjunctive normal form (DNF) under the uniform distribution. The output hypothesis is a majority of parities. (Also, Jackson [J95] generalizes his DNF learning algorithm from uniform distribution to any fixed constant bounded product distribution.)

The lattice-based techniques are, at a very high level, performing a traversal of the binary cube, moving from one node to its neighbor in order to reach some goal node. Angluin [A88] gave the first lattice-based algorithm for learning monotone DNF. Bshouty [Bs93] developed the monotone theory, which gives a technique for

*Received by the editors May 17, 1997; accepted for publication (in revised form) July 30, 1999; published electronically October 31, 2002. A preliminary version appeared in 36th Annual Symposium on Foundations of Computer Science, 1995, IEEE Computer Society, pp. 304–311.

http://www.siam.org/journals/sicomp/31-6/32058.html

†Department of Computer Science, Technion, Haifa, Israel (bshouty@cs.technion.ac.il).

‡School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel (mansour@cs.tau.ac.il). This author's research was supported in part by a grant from the Israel Science Foundation and by a grant from the Israel Ministry of Science and Art.

learning decision trees under any distribution. (The output hypothesis in that case is depth 3 formulas.) Schapire and Sellie [SS93] gave a lattice-based algorithm for learning multivariate polynomials over a finite field under any distribution. (Their algorithm depends polynomially on the size of the monotone polynomial that describes the function.)

Multiplicity automata theory is a well-studied field in automata theory. Recently, some very interesting connections were given, connecting learning such automata and learning decision trees and multivariate polynomials. Ohnishi, Seki, and Kasami [OSK94] and Bergadano and Varricchio [BV96] gave algorithms for learning multiplicity automata. Based on this work, Bergadano, Catlano, and Varricchio [BCV96] showed that this algorithm learns disjoint DNF. Then Beimel et al. [BBB+96] gave an algorithm that is based on Hankel matrices theory for learning multiplicity automata and showed that multivariate polynomials over any field are learnable in polynomial time. (In all the above algorithms the output hypothesis is a multiplicity automaton.)

All three techniques—the Fourier spectrum, the lattice-based, and the multiplicity automata algorithms—also give learnability of many other classes such as learning decision trees over parities (nodes contains parities) under constant bounded product distributions, learning CDNF (poly size DNF that has poly size CNF) under any distribution, and learning j-disjoint DNF (DNF where the intersection of any j terms is 0).

In this paper we develop a new approach for learning decision trees and multivariate polynomials via interpolation of multivariate polynomials over $GF(2)$. This new approach leads to simple learning algorithms for decision trees over the uniform and constant bounded product distributions, where the output hypotheses are a multivariate polynomial (parity of monotone terms).

The algorithm we develop gives a single hypothesis that approximates the target function with respect to *any* constant bounded product distribution. In fact the hypothesis is a good hypothesis under any distribution that *supports small terms*. That is, for any distribution D, where for a term T of size $\omega(\log n)$, we have $\Pr_D[T = 1] = 1/\omega(poly(n))$. Previous algorithms do not achieve this property.

It is also known that any DNF is learnable with membership queries under constant bounded product distribution [J95], where the output hypothesis is a majority of parities. Our contribution for j-disjoint DNF is to use an output hypothesis that is a parity of terms and to show that the output hypothesis is an ϵ-approximation of the target against any constant bounded distribution.

We also study the learnability of multivariate polynomials from membership queries only. We give a learning algorithm for multivariate polynomials over n variables with maximal degree $d < c|\mathcal{F}|$ for each variable, where $c < 1$ is constant, and with terms of size

$$k = O\left(\frac{|\mathcal{F}|}{d}(\log n + \log d)\right)$$

using only membership queries. This result implies learning in polynomial time decision trees of depth $O(\log n)$ with leaves from a field $\mathcal{F}$ from membership queries only.

This result is a generalization of the results in [B95b] and [RB89], where the learning algorithm uses membership and equivalence queries in the former and only membership queries in the latter.

The second result is a generalization of the result in [KM93] for learning Boolean decision trees from membership queries. The above result also gives an algorithm for

learning *any* multivariate polynomial over fields of size $q = n/(d(\log n + \log d))$ from membership queries only.

This result is a generalization of the results in [BT88, CDG+91, Z90] for learning multivariate polynomials under any field. Previous algorithms for learning multivariate polynomials over finite fields $\mathcal{F}$ required asking membership queries with assignments in some extension of the field $\mathcal{F}$ [CDG+91]. In [CDG+91] it is shown that an extension n of the field is sufficient to interpolate any multivariate polynomial (when membership queries with assignments from an extension field are allowed). Following this work, [HR96] gave an algorithm that interpolates polynomials over "large" finite fields, without using extension fields; for smaller fields they use an extension field.

The organization of the paper is as follows. In section 2 we define the learning model and the concept classes. In section 3 we give the algorithm for learning multivariate polynomial for the Boolean domain. In section 4 we give some background for multivariate interpolation. In section 5 we show how to reduce learning multivariate polynomials to zero testing and to other problems. Then in section 6 we give the algorithm for zero testing and also give a lower bound for zero testing multivariate polynomials.

2. The learning model and concept classes.

2.1. Learning models. The learning criterion we consider is *exact learning* [A88] and *PAC-learning* (probably approximately correct) [Val84].

In the exact learning model there is a function f called the *target function* $f : \mathcal{F}^n \to \mathcal{F}$ which is a member of a class functions C defined over the variable set $V_n = \{x_1, \ldots, x_n\}$ for some field $\mathcal{F}$. The goal of the learning algorithm is to output a formula h that is equivalent to f.

The learning algorithm performs a *membership query* (also called a *substitution query* for the nonbinary fields) by supplying an assignment a to the variables in $V_n = \{x_1, \ldots, x_n\}$ as input to a *membership oracle* and receives in return the value of $f(a)$. For our algorithms we will regard this oracle as a procedure $MQ_f()$. The procedure input is an assignment a and its output is $MQ_f(a) = f(a)$.

The learning algorithm performs an *equivalence query* by supplying any function h as input to an *equivalence oracle* with the oracle returning either "yes," signifying that h is equivalent to f, or a *counterexample*, which is an assignment b such that $h(b) \neq f(b)$. For our algorithms we will regard this oracle as a procedure $EQ_f(h)$. We say the *hypothesis* class of the learning algorithm is H if the algorithm supplies the equivalence oracle functions from H.

We say that a class of Boolean function C is *exactly learnable* in polynomial time if for any $f \in C$ over V_n there is an algorithm that runs in polynomial time, asks a polynomial number of queries (polynomial in n and in the size of the target function), and outputs a hypothesis h that is equivalent to f.

The PAC-learning model is as follows. There is a function f called the *target function* which is a member of a class of functions C defined over the variable set $V_n = \{x_1, \ldots, x_n\}$. There is a distribution D defined over the domain $\mathcal{F}^n$. The goal of the learning algorithm is to output a formula h that is ϵ-close to f with respect to some distribution D, that is,

$$\Pr_D[f(x) = h(x)] \geq 1 - \epsilon.$$

Such a function h is called an ϵ-approximation of f with respect to the distribution D.

In the PAC or *example query* model, the learning algorithm asks for an example from the *example oracle*, and receives an example $(a, f(a))$, where a is chosen from $\{0,1\}^n$ according to the distribution D.

We say that a class of Boolean functions C is *PAC-learnable* under the distribution D in polynomial time if for any $f \in C$ over V_n, there is an algorithm that runs in polynomial time, asks a polynomial number of queries (polynomial in n, $1/\epsilon$, $1/\delta$, and the size of the target function), and with probability at least $1-\delta$ outputs a hypothesis h that is an ϵ-approximation of f with respect to the distribution D.

It is known from [A88] that if a class is exactly learnable in polynomial time from equivalence queries and membership queries, then it is PAC-learnable with membership queries in polynomial time under any distribution D.

Let $\mathcal{D}$ be a set of distributions. We says that C is PAC-learnable under $\mathcal{D}$ if there is a PAC-learning algorithm for C such that for any distribution $D \in \mathcal{D}$ unknown to the learner and for any $f \in C$ the learning algorithm runs in polynomial time and outputs a hypothesis h that is an ϵ-approximation of f under any distribution $D' \in \mathcal{D}$.

2.2. The concept classes and distributions. A function over a field $\mathcal{F}$ is a function $f : X \to \mathcal{F}$ for some set X. All classes considered in this paper are classes of functions where $X = \mathcal{F}^n$. The elements of $\mathcal{F}^n$ are called *assignments*. We will consider the set of variables $V_n = \{x_1, \ldots, x_n\}$ where x_i will describe the value of the i-projection of the assignment in the domain $\mathcal{F}^n$ of f. For an assignment a, the ith entry of a will be denoted by a_i. A restriction α is a partial assignment, namely, it assigns values to some subset U_α of the variables V_n. A function f restricted by α, denoted by f_α, has as inputs the variables in $V_n - U_\alpha$. Given an assignment β for $V_n - U_\alpha$ the value of $f_\alpha(\beta)$ is $f(\alpha \cup \beta)$.

A *literal* is a nonconstant polynomial $p(x_i)$. A *monotone literal* is x_i^r for some nonnegative integer r. A *term* (*monotone term*) is a product of literals (monotone literals). A *multivariate polynomial* is a linear combination of monotone terms. A *multivariate polynomial with nonmonotone terms* is a linear combination of terms. The *degree* of a literal $p(x_i)$ is the degree of the polynomial p. The *size* of a term $p_{i_1}(x_{i_1}) \cdots p_k(x_{i_k})$ is k.

Let $MUL_{\mathcal{F}}(n,k,t,d)$ be the set of all multivariate polynomials over the field $\mathcal{F}$ over n variables with at most t monotone terms, where each term is of size at most k and each monotone literal is of degree at most d. For the binary field $\mathcal{B}$ the degree is at most $d = 1$ so we will use $MUL(n,k,t)$. $MUL^\star_{\mathcal{F}}(n,k,t,d)$ will be the set of all multivariate polynomials with nonmonotone terms with the above properties. We use $MUL^\star(n,k,t)$ when the field is the binary field. Throughout the paper we will assume that $t \geq n$. Since every term in $MUL^\star_{\mathcal{F}}(n,k,t,d)$ can be written as a multivariate polynomial in $MUL_{\mathcal{F}}(n,k,(d+1)^k,d)$ we have the following proposition.

PROPOSITION 2.1.

$$MUL^\star_{\mathcal{F}}(n,k,t,d) \subseteq MUL_{\mathcal{F}}(n,k,t(d+1)^k,d).$$

For the Boolean field $\mathcal{B} = \{0,1\}$ a DNF is a disjunction of terms. A j-disjoint DNF is a DNF where the conjunction of any j terms is 0. A k-DNF is a DNF with terms of size at most k literals.

A *decision tree* (with leaves from some field $\mathcal{F}$) over V_n is a binary tree whose nodes are labeled with variables from V_n and whose leaves are labeled with constants from $\mathcal{F}$. Each decision tree T represents a function $f_T : \{0,1\}^n \to \mathcal{F}$. To compute $f_T(a)$ we start from the root of the tree T: if the root is labeled with x_i, then $f_T(a) = f_{T_R}(a)$ if $a_i = 1$, where T_R is the right subtree of the root (i.e., the subtree

of the right child of the root with all its descendents). Otherwise (when $a_i = 0$), $f_T(a) = f_{T_L}(a)$, where T_L is the left subtree of the root. If T is a leaf, then $f_T(a)$ is the label of this leaf.

It is not hard to see that a Boolean decision tree of depth k can be represented in $MUL^\star(n, k, 2^k)$ (each leaf in the decision tree defines a term and the function is the sum of all terms), and that a j-disjoint k-DNF of size t can be represented in $MUL^\star(n, k(j-1), t^{j-1})$. (See, for example, [K94].) So for constant k and $d = O(\log n)$ the number of terms is polynomial.

For a DNF and a multivariate polynomial, f, we define $size(f)$ to be the number of terms in f. For a decision tree the size will be the number of leaves in the tree.

A *product distribution* is a distribution D that satisfies $D(a_1, \ldots, a_n) = \prod_i D_i(a_i)$ for some distributions D_i on $\mathcal{F}$. A product distribution is *fixed constant bounded* if there is a constant $0 < c < 1/2$, that is, independent of the number of variables n, such that for any variable x_i, $c \leq \text{Prob}[x_i = 1] \leq 1 - c$. A distribution D *supports small terms* if for every term of size $\omega(\log n)$, we have $\Pr_D[T = 1] = 1/\omega(poly(n))$, where n is the number of variables.

3. Simple algorithm for the Boolean domain. In this section we give an algorithm that PAC-learns with membership queries $MUL^\star(n, n, t)$ under any distribution that supports small terms in polynomial time in n and t. We remind the reader that we assume $t \geq n$. However, all the algorithms in the paper run in polynomial time also when $t < n$.

3.1. Zero test $MUL(n, k, t)$. We first show how to zero test elements in $MUL(n, k, t)$ in polynomial time in n and 2^k assuming k is known to the learner. The algorithm will run in polynomial time for $k = O(\log n)$. Let $f \in MUL(n, k, t)$. Choose a term $T = x_{i_1} \cdots x_{i_j}$, $j \leq k$, of maximal size in f. Choose any values from $\{0, 1\}$ for the variables not in T. The projection will not be the zero function because the term T will stay alive in the projection. Since the projection is a nonzero function with $j \leq k$ variables there is at least one assignment for $x_{i_1}, \ldots, x_{i_j}$ that gives value 1 for the function. This shows that for a random and uniform assignment a, $f(a) = 1$ with probability at least $1/2^j \geq 1/2^k$. So to zero test a function $f \in MUL(n, k, t)$, randomly and uniformly choose polynomial the number of assignments a_i. If $f(a_i)$ is zero for all the assignments, then with high probability we have $f \equiv 0$. Now from the above we have the following claim.

CLAIM 3.1. *For any $f \in MUL(n, k, t)$, $f \not\equiv 0$, the probability that $m = 2^k \log(1/\delta)$ randomly chosen elements $a_1, \ldots, a_m$ in $\{0, 1\}^n$ satisfies $f(a_1) = \ldots = f(a_m) = 0$ is at most δ.*

This implies the following claim.

CLAIM 3.2. *For $f \in MUL(n, O(\log t), t)$, there is a polynomial time probabilistic zero testing algorithm that succeeds with high probability.*

In Figure 1 we have the code for `zero_test`(h, α) that tests if the polynomial h is identical to the target function f on all the inputs that obey a given restriction α. For that case we have the following lemma.

LEMMA 3.3. *For any $f \in MUL(n, k, t)$, and any h, if $(h \oplus f)_\alpha \equiv 0$, then* `zero_test`$(h, \alpha)$ *returns TRUE, and if $(h \oplus f)_\alpha \not\equiv 0$, then* `zero_test`$(h, \alpha)$ *returns FALSE with probability at least $1 - \delta$.*

3.2. Learning $MUL(n, k, t)$. We now show how to reduce zero test to learning monotone polynomials.

```
FUNCTION MQ-hf(h: function, r: assignment);                    /* queries f ⊕ h at r */
RETURN h(r) ⊕ MQ_f(r);

FUNCTION zero_test(h: function, α: restriction);               /* testing if h ⊕ f is zero */
m ← 2^k log(1/δ)                                               /* number of queries */
FOR i = 1 to 2^k log(1/δ) DO
    α_i ← α;
    FOR j = 1 to n DO                                          /* extending α randomly */
        IF {x_j ← b_j} ∉ α_i, THEN
            α_i ← α_i ∪ {x_j ← random_bit()};
    END-FOR;
    IF MQ-hf(h, α_i) = 1, THEN                                 /* Evaluating h ⊕ f at α_i */
        RETURN FALSE;                                          /* Found an assignment for which h ⊕ f is not zero */
END-FOR;
RETURN TRUE;
```

FIG. 1. *The zero test function for polynomials. The function* `zero_test` *checks if the input polynomial h is identical to the target function f on the inputs that obey the restriction α. If they are identical, it outputs TRUE, and if they are not identical, it outputs (with high probability) FALSE. (The function* `random_bit()` *returns* 0 *with probability half and* 1 *with probability half.)*

Let $f \in MUL(n,k,t)$. We first show how to find one term in f. If $f(0) = 1$, then we know that $T = 1$ is a term in f. If $f(0) = 0$, then let $f_0 = f$. Since we can zero test we can find the minimal i_1 such that $f_0|_{x_1 \leftarrow 0,\ldots,\ldots,x_{i_1} \leftarrow 0} \equiv 0$. This implies that $f_{x_1 \leftarrow 0,\ldots,\ldots,x_{i_1-1} \leftarrow 0} = x_{i_1} f_1(x_{i_1+1},\ldots,x_n)$ for some multivariate polynomial f_1. If $f_1(0) = 1$, then we know that $T = x_{i_1}$ is a term in f. We continue recursively with $f_1 = f_{x_1 \leftarrow 0,\ldots,\ldots,x_{i_1-1} \leftarrow 0, x_{i_1} \leftarrow 1}$ until $f_j(0) \equiv 1$, in this case $T = x_{i_1} \cdots x_{i_j}$ is a term in f. Note that we invoke the function `zero_test` at most n times to recover a single term. A formal definition of `Find_Term`, the function that finds a single term, is found in Figure 2.

After we find a term T we define $\hat{f} = f + T$. This removes the term T from f, and thus $\hat{f} \in MUL(n,k,t-1)$. We continue recursively with $\hat{f}$ until we recover all the terms of f. Membership queries for $\hat{f}$ can be simulated by membership for f because $MQ_{\hat{f}}(a) = MQ_f(a) + T(a)$. The formal definition of `Monotone_Polynomial`, the function that interpolates a monotone polynomial, is found in Figure 2.

The function `Monotone_Polynomial` invokes the function `Find_Term` once for each of the terms. Each invocation of `Find_Term` may invoke `zero_test` at most n times. This gives the following claim.

CLAIM 3.4. *For $f \in MUL(n,k,t)$, the function* `Monotone_Polynomial` *returns a polynomial h, such that $h \equiv f$ with probability at least $1 - nt\delta$, uses $nt2^k \log \frac{1}{\delta}$ membership queries, and runs in time $O(nt2^k \log \frac{1}{\delta})$.*

In particular this gives the following claim.

CLAIM 3.5. *For $f \in MUL(n, O(\log t), t)$, there is a polynomial time probabilistic interpolation algorithm that succeeds with high probability to learn f from membership queries.*

3.3. Learning $MUL^\star(n,n,t)$. We now give a PAC-learning algorithm that learns $MUL^\star(n,n,t)$ under any distribution that support small terms. We first give the idea of the algorithm.

Let $f \in MUL^\star(n,n,t)$. To PAC-learn f we randomly choose an assignment a and define $f'(x) = f(x \oplus a)$. A term in f of size k will have on average $k/2$ monotone

```
FUNCTION Find_Term(h:function, a:offset, α: restriction);   /* Finds a term in f ⊕ h */
β ← α; T ← ∅;
FOR i = 1 to n DO
    IF{x_i ← b} ∉ α, THEN                                   /* check if variable is alive */
      tmp_β ← β ∪ {x_i ← a_i};                               /* try to restrict x_i */
      IF zero_test(h, tmp_β)=TRUE, THEN                     /* does x_i appear in all remaining terms? */
        T ← T ∪ {x_i};                                      /* add x_i to the term T */
        β ← β ∪ {x_i ← (1 − a_i)};                          /* restrict x_i */
        γ ← {x_j ← a_j | j ≥ i + 1, {x_j ← b} ∉ α};         /* complete the restriction β */
        IF MQ-hf(h, β ∪ γ) = 1, THEN
          RETURN T;                                         /* x_i was the only variable left. we are done! */
      ELSE β ← tmp_β;                                       /* there are terms without x_i, search for such a term*/
END-FOR;
```

```
FUNCTION Monotone_Polynomial(a: offset, α: restriction);
                                                            /* learn the monotone representation */
h = 0;                                                      /* initial hypothesis */
WHILE zero_test(h, a) =FALSE DO                             /* test if we are done? */
    h ← h ⊕ Find_Term(h, a, α);                             /* add another monotone term */
END-WHILE;
RETURN h;
```

FIG. 2. *The code of the learning algorithm for a polynomial using the representation of the monotone terms. The offset and the restriction are introduced to extend the algorithm to handle learning a polynomial with a small number of nonmonotone terms. If the target function, f, is a sparse monotone polynomial, we can set the offset, a, to the zero vector and the restriction, α, to be an empty set.*

literals in f', and terms with $k = \Omega(\log t)$ variable will have with high probability $\Omega(k)$ monotone literals.

We perform a zero restriction, i.e., for each i, with probability $1/2$ we substitute $x_i \leftarrow 0$ in f'. Since a term of size k in f has on average $k/2$ monotone literals after the shift $f(x \oplus a)$, in the zero restriction this term will be zero with probability (about) $1 - 2^{-k/2}$. This probability is greater than $1 - 1/poly(t)$ for $k = \Omega(\log t)$. Therefore with high probability all the terms of size more than $\Omega(\log t)$ will be removed by the zero restriction. This ensures that with high probability the projection f'' is in $MUL^{\star}(n, O(\log t), t)$, and therefore by Proposition 2.1 $f'' \in MUL(n, O(\log t), poly(t))$. Now we can use the algorithm in subsection 3.2, `Monotone_Polynomial`, to learn f''. Notice that for multivariate polynomial h (with monotone terms) when we performed a zero restriction, we deleted some of the monotone terms from h; therefore, the monotone terms of f'' are monotone terms of f'.

We continue to take zero restrictions and collect terms of f' until the sum of terms that appear in at least one restriction defines a multivariate polynomial which is a good approximation of f'. We get a good approximation of f' with respect to any distribution that supports small terms since we collect all the small size terms (i.e., terms of size $O(\log t)$).

THEOREM 3.6. *There is a polynomial time probabilistic PAC-learning algorithm with membership queries that learns $MUL^{\star}(n, n, t)$ under any distribution that supports small terms.*

We now prove that the function `Non_Monotone_Polynomial` (given in Figure 3) PAC-learns with membership queries any multivariate polynomial with at most t nonmonotone terms under distributions that support small terms. For the analysis of the correctness of the algorithm we first need to formalize the notion of distributions that

```
FUNCTION Non_Monotone_Polynomial();  /* learns a polynomial with t non-monotone terms */
a ← random_assignment();             /* we will learn f(x ⊕ a) */
r = (t/ε)^c (ln (t/4 (t/ε)^{4c}));
FOR j = 1 to r DO
    α_j ← ∅;                                   /* build a random restriction */
    FOR i = 1 to n DO
        IF random_bit()= TRUE,                 /* restrict x_i with probability half */
            THEN α_j ← α_j ∪ {x_i ← a_i}
    END-FOR;
    h_j ← Monotone_Polynomial(a, α_j);         /* find terms of f(a ⊕ x) restricted by α_j */
END-FOR;
T ← {T | T is a term of h_j and |T| ≤ 4c log(t/ε)}   /* collect all short term found */
h(x ⊕ a) = ⊕_{T∈T} T;
RETURN h(x ⊕ a);
```

```
FUNCTION random_assignment();
FOR i = 1 to n DO
    a_i ← random_bit();
END-FOR;
RETURN (a_1, ..., a_n);
```

FIG. 3. *The algorithm to learn a polynomial with t nonmonotone terms. First we fix a random offset a. Then we choose many zero restrictions, α_j, randomly. For each zero restriction we reconstruct the polynomial by finding its monotone terms. Finally, our hypothesis includes all the short terms.*

support small terms. The following is one way to define this notion.

DEFINITION 3.7. *Let $\mathcal{D}_{c,t,\epsilon}$ include any distribution D such that for any DNF f with t terms, each of size greater than $c\log(t/\epsilon)$, we have $\Pr_D[f=1] \le \epsilon$.*

Notice that all the constant bounded product distributions D, where $1-d \le \Pr_D[x_i=1] \le d$ for all i, are in $\mathcal{D}_{1/\log(1/d),t,\epsilon}$. In what follows we will assume that $c \ge 2$ and $\epsilon < 1/8$. We will use the Chernoff bound (see [ASE]).

LEMMA 3.8. *Let $X_1, \ldots, X_m \in \{0,1\}$ be independent random variables where $\Pr[X_i = 1] = 1/2$. Then for any γ we have*

$$\Pr\left[\sum_{i=1}^{m} X_i \le \frac{m}{2} - \gamma\right] \le 2^{-\frac{2\gamma^2}{m}}.$$

Let $f = T_1 \oplus \cdots \oplus T_t$ be a multivariate polynomial where $T_1, \ldots, T_t$ are terms and $|T_1| \le |T_2| \le \cdots \le |T_t|$. Our algorithm starts by choosing a random assignment a and defines $f'(x) = f(x \oplus a)$. All terms that are of size s (in f') will contain on average $s/2$ monotone literals. Therefore by the Chernoff bound we have the following lemma.

LEMMA 3.9. *With probability at least $1-\epsilon$, all the terms in f' of size more than $\alpha c\log(t/\epsilon)$ contain at least $(\alpha/4)c\log(t/\epsilon)$ monotone literals, where $\alpha \ge 4$ and $c \ge 2$.*

Proof. Let T be any term of size at least $\alpha c\log(t/\epsilon)$. Let $P(T)$ be the number of monotone literals in T. We have

$$\Pr\left[P(T) < \frac{\alpha}{4} c \log\frac{t}{\epsilon}\right] \le 2^{-\frac{\alpha c}{8}\log\frac{t}{\epsilon}} = \left(\frac{\epsilon}{t}\right)^{\frac{\alpha c}{8}} \le \frac{\epsilon}{t}.$$

Since the number of terms of f' is t the lemma follows. □

With probability at least $1-\epsilon$, all the terms of size more than $4c\log(t/\epsilon)$ will contain at least $c\log(t/\epsilon)$ monotone literals and all terms of size $8c\log(t/\epsilon)$ will contain

at least $2c\log(t/\epsilon)$ monotone literals. For the analysis, we now split the function f' into 3 functions f'_1, f'_2, and f'_3. The function $f'_1 = T_1 \oplus \cdots \oplus T_{t_1}$ will contain all terms that are of size at most $4c\log(t/\epsilon)$. The function $f'_2 = T_{t_1+1} \oplus \cdots \oplus T_{t_2}$ will contain all terms of size between $4c\log(t/\epsilon)$ and $8c\log(t/\epsilon)$, and the function $f'_3 = T_{t_2+1} \oplus \cdots \oplus T_t$ will contain all terms of size more than $8c\log(t/\epsilon)$.

Since $f'_1 \in MUL^\star(n, 4c\log(t/\epsilon), t)$, we have $f'_1 \in MUL(n, 4c\log(t/\epsilon), t(t/\epsilon)^{4c})$ by Proposition 2.1. Similarly, $f'_2 \in MUL(n, 8c\log(t/\epsilon), t(t/\epsilon)^{8c})$.

Our algorithm will find all the terms in f'_1, some of the terms in f'_2, and none of the terms in f'_3. The following lemma shows that this is sufficient in order to learn f.

LEMMA 3.10. *Let $f \in MUL^\star(n, n, t)$, $a \in \{0,1\}^n$ be any assignment, and $f'(x) = f(x \oplus a)$. Let $g = f'_1 \oplus h$, where h is a multivariate polynomial that contains some of the monotone terms in f'_2. Then for any $D \in \mathcal{D}_{c,t,\epsilon}$ we have*

$$\Pr_D[g \neq f'] \leq \epsilon,$$

assuming that all the terms in f' of size more than $\alpha c\log(t/\epsilon)$ contain at least $(\alpha/4)c\log(t/\epsilon)$ monotone literals, where $\alpha \geq 4$ and $c \geq 2$.

Proof. The error is

$$\Pr_D[g \neq f'] = \Pr_D[(f'_1 \oplus h) \oplus f' = 1] = \Pr_D[h \oplus f'_2 \oplus f'_3 = 1].$$

Let

$$f'_2 = \hat{T}_{t_1+1}\tilde{T}_{t_1+1} \oplus \cdots \oplus \hat{T}_{t_2}\tilde{T}_{t_2} \in MUL^\star(n, 8c\log(t/\epsilon), t),$$

where $T_i = \hat{T}_{t_i}\tilde{T}_{t_i}$, $\hat{T}_{t_i}$ is the part of the term that contains monotone literals and $\tilde{T}_{t_i}$ is the part that contains the nonmonotone literals. If $\hat{T}_{t_1+1} = x_{i_1} \cdots x_{i_k}$ and $\tilde{T}_{t_1+1} = \bar{x}_{j_1} \cdots \bar{x}_{j_l}$, then notice that when we change $\hat{T}_{t_1+1}\tilde{T}_{t_1+1}$ to the sum of monotone terms, we get

$$\sum_{S \subseteq \{j_1,\ldots,j_l\}} \hat{T}_{t_1+1} \prod_{q \in S} x_{j_q}.$$

So every monotone term in f'_2 will contain one of the terms $\hat{T}_i$, $t_1 + 1 \leq i \leq t_2$. Therefore we can write $f'_2 = \hat{T}_{t_1+1}f'_{2,1} \oplus \cdots \oplus \hat{T}_{t_2}f'_{2,t_2-t_1}$, where $f'_{2,i}$ are multivariate polynomials with monotone terms. Since h is a multivariate polynomial that contains some of the terms in f'_2, we have $f'_2 \oplus h = \hat{T}_{t_1+1}h_{2,1} \oplus \cdots \oplus \hat{T}_{t_2}h_{2,t_2-t_1}$. By our assumption, $|\hat{T}_i| \geq c\log(t/\epsilon)$, for $t_1 + 1 \leq i \leq t_2$, and $|T_i| \geq 2c\log(t/\epsilon)$, for $i \geq t_2 + 1$. Since D is a distribution that supports small terms, i.e., $D \in \mathcal{D}_{c,t,\epsilon}$, we have

$$\Pr_D[(h \oplus f_2) \oplus f_3 = 1] \leq \Pr_D[\hat{T}_{t_1+1} \vee \cdots \vee \hat{T}_{t_2} \vee T_{t_2+1} \vee \cdots \vee T_t] \leq \epsilon,$$

and the lemma follows. ∎

The algorithm will proceed as follows. We choose

$$r = \left(\frac{t}{\epsilon}\right)^c \left(\ln\left(\frac{t}{4}\left(\frac{t}{\epsilon}\right)^{4c}\right)\right)$$

zero restrictions $\alpha_1, \ldots, \alpha_r$ for f'. Recall that a zero restriction α of f' is a function f'_α where, with probability $1/2$, $x_i \leftarrow 0$ and, with probability $1/2$, the variable x_i remains alive. We will show that with some constant probability we have the following:

(A) For every monotone term in f_1' there is a restriction α_i such that $(f_1')_{\alpha_i}$ contains this term.

(B) For every $i = 1, \ldots, r$ we have $(f_3')_{\alpha_i} \equiv 0$.

We define $T(\alpha_i) = 0$ if the restriction α_i and the term T share a variable, and $T(\alpha_i) = *$ if they do not share a variable. The next lemma proves (A), namely, that all the small terms would be recovered (as monotone terms).

LEMMA 3.11. *Let $f \in MUL^\star(n, n, t)$, $a \in \{0,1\}^n$ be any assignment, and $f'(x) = f(x \oplus a)$. The probability that for each monotone term T of f_1', whose size is at most $4c\log(t/\epsilon)$, there exists a random restriction α_i, such that $T(\alpha_i) \neq 0$ is at least $3/4$.*

Proof. Let $\mathcal{T}_1$ be the set of monotone terms in f_1'. By the definition of f_1', we have that $|\mathcal{T}_1| \leq t(t/\epsilon)^{4c}$ and every term in $\mathcal{T}_1$ is of size at most $4c\log(t/\epsilon)$. Let A be the event that for each term $T \in \mathcal{T}_1$ there is a restriction α_i such that $T(\alpha_i) \neq 0$. We have

$$\begin{aligned}
\Pr[\text{not } A] &= \Pr[(\exists T \in \mathcal{T}_1)(\forall \alpha_i)\ T(\alpha_i) = 0] \\
&\leq t\left(\frac{t}{\epsilon}\right)^{4c} \Pr[(\forall \alpha_i)\ T(\alpha_i) = 0] \\
&\leq t\left(\frac{t}{\epsilon}\right)^{4c} \left(1 - \frac{1}{2^{c\log(t/\epsilon)}}\right)^r = \frac{1}{4},
\end{aligned}$$

which completes the proof of the lemma. ∎

The next lemma proves (B), namely, that with high probability none of the large terms survives any of the zero restrictions.

LEMMA 3.12. *Let $f \in MUL^\star(n, n, t)$ and let $a \in \{0,1\}^n$ be a random assignment. The probability that for some term T of $f'(x) = f(x \oplus a)$, whose size is at least $8c\log(t/\epsilon)$, there exists a random restriction α_i, such that T is a term of f'_{α_i}, $1 \leq i \leq r$, is at most 2ϵ, where the probability is taken over the choice of a and α_i.*

Proof. Let $\mathcal{T}_3$ be the set of terms of f' of size at least $8c\log(t/\epsilon)$. Clearly the number of terms in $\mathcal{T}_3$ is at most t. Let B be the event that for every $T \in \mathcal{T}_3$ and each restriction α_i, $T(\alpha_i) = 0$, and therefore f'_{α_i} does not contain T. Let B' be the event that we select an a such that every term T of f', whose size is at least $8c\log(t/\epsilon)$, has at least $2c\log(t/\epsilon)$ monotone literals.

By Lemma 3.9, with probability at least $1 - \epsilon$, event B' holds. Assume that B' holds. We compute the probability that B does not hold, given that B' holds.

$$\begin{aligned}
\Pr[\text{not } B \mid B'] &= \Pr[(\exists i)(\exists T \in \mathcal{T}_3)\ T(\alpha_i) \neq 0] \\
&\leq rt\left(\frac{1}{2}\right)^{2c\log(t/\epsilon)} \\
&\leq \epsilon.
\end{aligned}$$

Clearly,

$$\Pr[\text{not } B] \leq \Pr[\text{not } B'] + \Pr[\text{not } B \mid B'] \leq 2\epsilon,$$

which completes the proof of the lemma. ∎

From Lemmas 3.11 and 3.12 we have that with probability at least $3/4 - 2\epsilon$ all the projections f'_{α_i} contain terms of size at most $8c\log(t/\epsilon)$ and for each monotone term of f_1' there is a projection f'_{α_i} that contains the term. Therefore, the algorithm proceeds by learning each projection $f'_{\alpha_i} \in MUL(n, 8c\log(t/\epsilon), t(t/\epsilon)^{8c})$ using the monotone

polynomial algorithm and collecting all the terms of size at most $4c\log(t/\epsilon)$. This ensures that with probability $3/4-2\epsilon$ we recover all the terms of f_1'. By Lemma 3.10 this implies that the error is at most ϵ. (The details of the algorithm are found in Figure 3.)

The running time of the function `Non_Monotone_Polynomial`, and the number of membership queries, are bounded by

$$O\left(r\cdot\left(t\left(\frac{t}{\epsilon}\right)^{8c}\right)\cdot n\cdot 2^{8c\log(t/\epsilon)}\log 1/\delta\right)=O\left(\left(\frac{t}{\epsilon}\right)^{16c}t^2 n\log(1/\delta)\ln(t/\epsilon)\right).$$

Recall that the constant c depends on the distribution. For example, for the uniform distribution, $c=1$.

The above analysis algorithm can also be used to learn functions $f:\{0,1\}^n\to\mathcal{F}$ of the form $f=\lambda_1T_1+\cdots+\lambda_tT_t$, where $\lambda_i\in\mathcal{F}$, T_i are terms and $+$ is the addition of a field $\mathcal{F}$. These functions can be computed as follows. For an assignment a, $f(a)=\sum_{T_i(a)=1}\lambda_i$. This gives the learnability of decision trees with leaves that contain elements from the field $\mathcal{F}$.

4. Multivariate interpolation. In this section we show how to generalize the above algorithm for any multivariate polynomial over any field. Let

$$f=\sum_{\alpha\in\mathcal{I}}a_\alpha x_1^{\alpha_1}\cdots x_n^{\alpha_n}$$

be a multivariate polynomial over the field $\mathcal{F}$, where $a_\alpha\in\mathcal{F}$ and $\alpha_1,\ldots,\alpha_n$ are integers. We will denote the class of all multivariate polynomials over the field $\mathcal{F}$ and over the variables $x_1,\ldots,x_n$ by $\mathcal{F}[x_1,\ldots,x_n]$. The number of terms of f is denoted by $|f|$. We have $|f|=|\mathcal{I}|$ when all a_α are not zero. When $f=0$ then $|f|=0$ and when $f=c\in\mathcal{F}\backslash\{0\}$ then $|f|=1$. Let d be the maximal degree of variables in f, i.e., $\mathcal{I}\subseteq[d]^n$ where $[d]=\{0,1,\ldots,d\}$. Suppose $\mathcal{F}'=\{\gamma_0,\ldots,\gamma_d\}\subseteq\mathcal{F}$ are $d+1$ distinct field constants, where $\gamma_0=0$ is the zero of the field. A univariate polynomial $f(x_1)\in\mathcal{F}[x_1]$ over the field $\mathcal{F}$ of degree at most d can be interpolated from membership queries as follows. Suppose that

$$f(x_1)=\Delta^{(d)}(f)x_1^d+\cdots+\Delta^{(1)}(f)x_1+\Delta^{(0)}(f),$$

where $\Delta^{(i)}(f)$ is the coefficient of x^i in f in its polynomial representation. Then

$$\begin{cases} f(\gamma_0) &=& \Delta^{(d)}(f)\gamma_0^d+\cdots+\Delta^{(1)}(f)\gamma_0+\Delta^{(0)}(f),\\ f(\gamma_1) &=& \Delta^{(d)}(f)\gamma_1^d+\cdots+\Delta^{(1)}(f)\gamma_1+\Delta^{(0)}(f),\\ \vdots & \vdots & \vdots\\ f(\gamma_d) &=& \Delta^{(d)}(f)\gamma_d^d+\cdots+\Delta^{(1)}(f)\gamma_d+\Delta^{(0)}(f). \end{cases}$$

This is a linear system of equations and can be solved for $\Delta^{(i)}(f)$ as follows:

$$\frac{\det\begin{vmatrix} \gamma_0^d & \cdots & \gamma_0^{i+1} & f(\gamma_0) & \gamma_0^{i-1} & \cdots & \gamma_0 & 1\\ \gamma_1^d & \cdots & \gamma_1^{i+1} & f(\gamma_1) & \gamma_1^{i-1} & \cdots & \gamma_1 & 1\\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots\\ \gamma_d^d & \cdots & \gamma_d^{i+1} & f(\gamma_d) & \gamma_d^{i-1} & \cdots & \gamma_d & 1 \end{vmatrix}}{\det|V(\gamma_0,\ldots,\gamma_d)|}, \tag{1}$$

where $V(\gamma_0, \ldots, \gamma_d)$ is the Vandermonde matrix.

If f is a multivariate polynomial, then f can be written as

$$f(x_1, \ldots, x_n) = \Delta^{(d)}(f)x_1^d + \cdots + \Delta^{(1)}(f)x_1 + \Delta^{(0)}(f),$$

where $\Delta^{(i)}(f)$ is a multivariate polynomial over the variables $x_2, \ldots, x_n$. We can still use (1) to find $\Delta^{(i)}(f)$ by replacing each $f(\gamma_i)$ with $f(\gamma_i, x_2, \ldots, x_n)$. Notice that from the first equation in the system, since $\gamma_0 = 0$, we have

$$\Delta^{(0)}(f) = f(0, x_2, \ldots, x_n). \tag{2}$$

From (1) a membership query for $\Delta^{(i)}$ can be simulated using $d+1$ membership queries to f. From (2), a membership query to $\Delta^{(0)}$ can be simulated using one membership query to f.

We now extend the Δ operators as follows: for $\mathbf{i} = (i_1, \ldots, i_k) \in [d]^k$,

$$\Delta^{\mathbf{i}} = \Delta^{i_k}\Delta^{i_{k-1}} \cdots \Delta^{i_1}.$$

Here Δ always operates on the variable with the smallest index. So Δ^{i_1} operates on x_1 in f to give a function f' that depends on $x_2, \ldots, x_n$. Then Δ^{i_2} operates on x_2 in f' and so on. We will also write $x^{\mathbf{i}}$ for the term $x_1^{i_1}x_2^{i_2} \cdots x_k^{i_k}$. The weight of $\mathbf{i}$, denoted by $wt(\mathbf{i})$, is the number of nonzero entries in $\mathbf{i}$.

The operator $\Delta^i(f)$ gives the coefficient of x_1^i in f when represented in $\mathcal{F}[x_2, \ldots, x_n][x_1]$; the operator $\Delta^{\mathbf{i}}(f)$ gives the coefficient of $x^{\mathbf{i}}$ when f is represented in

$$\mathcal{F}[x_{k+1}, \ldots, x_n][x_1, \ldots, x_k].$$

Suppose $\mathcal{I} \subseteq [d]^k$ is such that $\Delta^{\mathbf{i}}f \neq 0$ for all $\mathbf{i} \in \mathcal{I}$ and $\Delta^{\mathbf{i}}f = 0$ for all $\mathbf{i} \notin \mathcal{I}$, that is, $x^{\mathbf{i}}$ for $\mathbf{i} \in \mathcal{I}$ are the k-suffixes of all terms of f. Here the k-suffix of a term $x_1^{i_1} \cdots x_n^{i_n}$ is $x_1^{i_1} \cdots x_k^{i_k}$. Since $\mathbf{i} \in \mathcal{I}$ if and only if $x^{\mathbf{i}}$ is a k-suffix of some term in f, it is clear that $|\mathcal{I}| \leq |f|$ and we must have

$$f = \sum_{\mathbf{i} \in \mathcal{I}} (\Delta^{\mathbf{i}}f)x^{\mathbf{i}}.$$

Now we will show how to simulate membership queries for $(\Delta^{\mathbf{i}}f)(x_{k+1}, \ldots, x_n)$, $\mathbf{i} \in \mathcal{I}$, using a polynomial number (in n and $|f|$) of membership queries to f. Suppose we want to find $(\Delta^{\mathbf{i}}f)(c)$ for some $c \in \mathcal{F}^{n-k}$ using membership queries to f. We take r assignments $\hat{\gamma}_1, \ldots, \hat{\gamma}_r \in \mathcal{F}^k$ and ask membership queries for $(\hat{\gamma}_i, c)$ for all $i = 1, \ldots, r$. If $f(\hat{\gamma}_i, c) = \omega_i$, then

$$\begin{cases} \sum_{\mathbf{i} \in \mathcal{I}} (\Delta^{\mathbf{i}}f)(c)\hat{\gamma}_1^{\mathbf{i}} = \omega_1, \\ \quad \vdots \qquad\qquad \vdots \quad \vdots \\ \sum_{\mathbf{i} \in \mathcal{I}} (\Delta^{\mathbf{i}}f)(c)\hat{\gamma}_r^{\mathbf{i}} = \omega_r. \end{cases}$$

Now if $\mathcal{I} = \{\mathbf{i}_1, \ldots, \mathbf{i}_r\}$ and $\det|\mathbf{M}[\hat{\gamma}_j; \mathbf{i}_j]| \neq 0$ for

$$\mathbf{M}[\hat{\gamma}_j; \mathbf{i}_j] = \begin{bmatrix} \hat{\gamma}_1^{\mathbf{i}_1} & \cdots & \hat{\gamma}_1^{\mathbf{i}_r} \\ \vdots & \vdots & \vdots \\ \hat{\gamma}_r^{\mathbf{i}_1} & \cdots & \hat{\gamma}_r^{\mathbf{i}_r} \end{bmatrix},$$

then the above linear system of equations can be solved in time $poly(r) = poly(|\mathcal{I}|) \leq poly(|f|)$. The solution gives $(\Delta^{\mathbf{i}}f)(c)$. The existence of $\hat{\gamma}_i$ where the above determinant is not zero will be proven in the next section.

5. Reducing learning to zero testing (for any field). In this section we show how to use the results from the previous section to learn multivariate polynomials.

Let $MUL_{\mathcal{F}}(n,k,t,d)$ be the set of all multivariate polynomials over the field $\mathcal{F}$ over n variables with t terms, where each term is of size k and the maximal degree of each variable is at most d. We would like to answer the following questions. Let $f \in MUL_{\mathcal{F}}(n,k,t,d)$.

1. Is there a polynomial time algorithm that uses membership queries to f and decides whether $f \equiv 0$?
2. Given $i \leq n$, is there a polynomial time algorithm that uses membership queries to f and decides whether f depends on x_i?
3. Given $\{\mathbf{i}_1, \ldots, \mathbf{i}_r\} \subseteq [d]^n$, where $wt(\mathbf{i}_j) \leq k$ for all j and $r \leq t$, is there an algorithm that runs in polynomial time and finds $\gamma_1, \ldots, \gamma_r \in \mathcal{F}^k$ such that

$$\begin{vmatrix} \gamma_1^{\mathbf{i}_1} & \cdots & \gamma_1^{\mathbf{i}_r} \\ \vdots & \vdots & \vdots \\ \gamma_r^{\mathbf{i}_1} & \cdots & \gamma_r^{\mathbf{i}_r} \end{vmatrix} \neq 0?$$

4. Is there a polynomial time algorithm that uses membership queries to f and identifies f?

When we say polynomial time we usually mean polynomial time in n, k, t, and d, but all the results of this section hold for any time complexity T if we allow a blow up of $poly(n,t)$ in the complexity.

We show that 1, 2, and 4 are equivalent and $1 \Rightarrow 3$. Obviously $2 \Rightarrow 1$, $4 \Rightarrow 1$, and $4 \Rightarrow 2$. We will show $1 \Rightarrow 2$, $1 \Rightarrow 3$, and $1 + 2 + 3 \Rightarrow 4$.

To prove $1 \Rightarrow 2$ notice that $f \in MUL_{\mathcal{F}}(n,k,t,d)$ is independent of x_i if and only if $g = f|_{x_i \leftarrow 1} - f|_{x_i \leftarrow 0} \equiv 0$. Since g is the coefficient of x_i in f, we have $g \in MUL_{\mathcal{F}}(n-1,k,t,d)$. Therefore we can zero test g in polynomial time.

To prove $1 \Rightarrow 3$, let $\gamma_1, \ldots, \gamma_s$ be a zero test for functions in $MUL_{\mathcal{F}}(n,k,t,d)$, that is, run the algorithm that zero test for the input 0 and take all the membership queries in the algorithm $\gamma_1, \ldots, \gamma_s$. We now have $f \in MUL_{\mathcal{F}}(n,k,t,d)$ is 0 if and only if $f(\gamma_i) = 0$ for all $i = 1, \ldots, s$. Consider the $s \times r$ matrix with rows $[\gamma_j^{\mathbf{i}_1}, \ldots, \gamma_j^{\mathbf{i}_r}]$. If this matrix has rank r, then we choose r linearly independent rows. If the rank is less than r, then its columns are dependent, and therefore there are constants c_i, $i = 1, \ldots, r$, such that

$$\sum_{i=1}^{r} c_i \gamma_j^{\mathbf{i}_i} = 0 \quad \text{for} \quad j = 1, \ldots, s.$$

This shows that the multivariate polynomial $\sum_{i=1}^{r} c_i x^{\mathbf{i}_i}$ is 0 for all $\gamma_1, \ldots, \gamma_s$. Since $\sum_{i=1}^{r} c_i x^{\mathbf{i}_i}$ is in $MUL_{\mathcal{F}}(n,k,t,d)$ we get a contradiction.

Now we show that $1 + 2 + 3 \Rightarrow 4$. This will use results from the previous section. The algorithm first checks whether f depends on x_1, and if yes it generates a tree with a root labeled with x_1 that has d children. The ith child is the tree for $\Delta^i(f)$. If the function is independent of x_1, it builds a tree with one child for the root. The child is $\Delta^0(f)$. We then recursively build the tree for the children. The previous section shows how to simulate membership queries at each level in polynomial time. This algorithm obviously works and its correctness follows immediately from the previous section and 1–3 above.

The complexity of the algorithm is the size of the tree times the membership query simulation. The size of the tree at each level is bounded by the number terms in f, and the depth of the tree is bounded by n; therefore, the tree has at most $O(nt)$ nonzero nodes. The total number of nodes is at most a factor of d from the nonzero nodes. Thus the algorithm has complexity the same as zero testing with a blow up of $poly(n,t,d)$ queries and time.

Now that we have reduced the problem to zero testing, we will investigate in the next section the complexity of zero testing of $MUL_{\mathcal{F}}(n,k,t,d)$.

6. Zero test of $MUL_{\mathcal{F}}(n,k,t,d)$. In this section we will study the zero testing of $MUL_{\mathcal{F}}(n,k,\star,d)$ when the number of terms is unknown and might be exponentially large. The time complexity for the zero testing should be polynomial in n and d. (We have $k<n$ so it is also polynomial in k.) We will show the following theorem.

THEOREM 6.1. *The class $MUL_{\mathcal{F}}(n,k,\star,d)$, where $d \le c|\mathcal{F}|$, is zero testable in randomized polynomial time in n, d, and t (here t is not the number of terms in the target, but only a free parameter) for some constant $c<1$ if and only if*

$$k = O\left(\frac{|\mathcal{F}|}{d}(\log n + \log d + \log t)\right).$$

The algorithm for the zero testing is simply to randomly and uniformly choose $poly(n,d)$ points a_i from $\mathcal{F}^n$ and query f at a_i and receive $f(a_i)$. If for all the points a_i, f is zero, then with high probability $f \equiv 0$.

This theorem implies the following one.

THEOREM 6.2. *The class $MUL_{\mathcal{F}}(n,k,t,d)$, where $d < c|\mathcal{F}|$ for some constant c, is learnable in randomized polynomial time (in n, d, and t) from membership queries if*

$$k = O\left(\frac{|\mathcal{F}|}{d}(\log n + \log d + \log t)\right).$$

Proof of Theorem 6.1 *(upper bound).* Let $\phi(n,k,d)$ be the maximal possible number of roots of a multivariate polynomial in $MUL_{\mathcal{F}}(n,k,\star,d)$. We will show the following facts:

1. $\phi(n,k,d) \le |\mathcal{F}|^{n-k}\phi(k,k,d)$,
2. $\phi(k,k,d) \le |\mathcal{F}|^k - (|\mathcal{F}|-d)^k$, and
3. $\phi(1,1,d) = d$.

This implies that if $f \not\equiv 0$, when we randomly uniformly choose an assignment $a \in \mathcal{F}^n$, we have

$$\begin{aligned}
\Pr_a[f(a)\neq 0] &\ge 1 - \frac{\phi(n,k,d)}{|\mathcal{F}|^n} \\
&\ge 1 - \frac{\phi(k,k,d)}{|\mathcal{F}|^k} \\
&\ge 1 - \frac{|\mathcal{F}|^k - (|\mathcal{F}|-d)^k}{|\mathcal{F}|^k} \\
&\ge \left(1 - \frac{d}{|\mathcal{F}|}\right)^k \\
&\ge e^{-O\left(\frac{dk}{|\mathcal{F}|}\right)}.
\end{aligned}$$

For $d \leq c|\mathcal{F}|$ we have that this probability is bounded by $\frac{1}{poly(n,d,t)}$. Therefore the expected running time to detect that f is not 0 is $poly(n,d,t)$.

It remains to prove conditions (1) and (2). To prove (1) let $f \in MUL_{\mathcal{F}}(n,k,\star,d)$ with a maximal number of roots. Let m be a term in f with a maximal number of variables. Suppose, without loss of generality, that $m = x_1^{i_1} \cdots x_k^{i_k}$. For any substitution $a_{k+1}, \ldots, a_n$ of the variables $x_{k+1}, \ldots, x_n$, the term m will stay alive in the projection $g = f|_{x_i \leftarrow a_i, i=k+1,\ldots,n}$ because it is maximal in f. Since g has at most $\phi(k,k,d)$ roots, the result (1) follows.

The proof of (2) is similar to the proof of Schwartz [Sch80] and Zippel [Zip79]. Let $f \in MUL_{\mathcal{F}}(k,k,\star,d)$. Write f as polynomial in $\mathcal{F}[x_2,\ldots,x_d][x_1]$,

$$f = f_d x_1^d + f_{d-1} x_1^{d-1} + \cdots + f_0.$$

Let t be the number of roots of f_d. Since $f_d \in MUL_{\mathcal{F}}(k-1,k-1,\star,d)$, we have

$$t \leq \phi(k-1,k-1,d).$$

For $|\mathcal{F}|^{k-1} - t$ assignments a for $x_2,\ldots,x_d$, we have $f_d(a) \neq 0$. For those assignments we get a polynomial in x_1 of degree d that has at most d roots for x_1. For t assignments a for $x_2,\ldots,x_k$, we have that f_d is zero and then the possible values of x_1 (to get a root for f) are bounded by $|\mathcal{F}|$. This implies that

$$\begin{aligned}\phi(k,k,d) &\leq d(|\mathcal{F}|^{k-1} - t) + t|\mathcal{F}| \\ &= d|\mathcal{F}|^{k-1} + (|\mathcal{F}| - d)t \\ &\leq d|\mathcal{F}|^{k-1} + (|\mathcal{F}| - d)\phi(k-1,k-1,d).\end{aligned}$$

The theorem follows by induction on k. □

Proof of Theorem 6.1 *(lower bound).* Let $\mathcal{A}$ be a randomized algorithm that zero tests $f \in MUL_{\mathcal{F}}(n,k,\star,d)$. Algorithm $\mathcal{A}$ asks membership queries to f and if $f \not\equiv 0$, it returns with probability at least 2/3 the answer "no." If all the membership queries in the algorithm return 0, the algorithm returns the answer "yes" indicating that $f \equiv 0$.

We run the algorithm for $f \equiv 0$. Let $D_1,\ldots,D_l$, $l = (dnt)^\lambda$, be the distributions that the membership assignments $a_1,\ldots,a_l$ are chosen to zero test f. Notice that if all membership queries answers are 0 while running the algorithm for $f \equiv 0$, it would again choose membership queries according to the distributions $D_1,\ldots,D_l$. Now randomly and uniformly choose $\gamma_{i,j} \in \mathcal{F}$, $i = 1,\ldots,p$, $j = 1,\ldots,d$, and define

$$f^\star = \prod_{i=1}^{p} \prod_{j=1}^{d} (x_i - \gamma_{i,j}),$$

where $p = 2\lambda \frac{|\mathcal{F}|}{d}(\ln n + \ln d + \ln t)$. Note that the degree of $f^\star$ is d and the size of each term is at most p. Let $I[f^*(a_i)] = 1$ if $f^\star(a_i) \neq 0$ and 0 otherwise. For every input a we have

$$E_{f^\star}\left[I[f^\star(a)]\right] = \left(1 - \frac{1}{|\mathcal{F}|}\right)^{pd}.$$

Therefore,

$$E_{f^\star} E_{(\forall i) a_i \in D_i \{0,1\}^n} \left[\bigvee_i I[f^\star(a_i)]\right] \leq E_{f^\star} E_{(\forall i) a_i \in D_i \{0,1\}^n} \left[\sum_i I[f^\star(a_i)]\right]$$

$$\leq E_{(\forall i)a_i \in D_i}\{0,1\}^n \left[\sum_i E_{f^\star} I[f^\star(a_{i_0})]\right]$$
$$= l\left[\left(1 - \frac{1}{|\mathcal{F}|}\right)^{pd}\right]$$
$$\leq \frac{2}{3}.$$

This shows that there exists $f^\star \not\equiv 0$ such that running algorithm $\mathcal{A}$ for $f^\star$ will give the wrong answer, i.e., "yes," with probability more than 2/3. This is a contradiction. □

Acknowledgments. We would like to thank the anonymous referees for their comments.

REFERENCES

[A88] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.

[ASE] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[BT88] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 301–309.

[Bs93] N. H. Bshouty, *Exact learning of boolean functions via the monotone theory*, Inform. Comput., 123 (1995), pp. 146–153.

[B95b] N. H. Bshouty, *A note on learning multivariate polynomials under the uniform distribution*, in Proceedings of the Annual ACM Workshop on Computational Learning Theory, 1995.

[BBB+96] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio, *On the applications of multiplicity automata in learning*, in Proceedings of the 37th Symposium on Foundations of Computer Science, 1996, IEEE Computer Society, pp. 349–358.

[BCV96] F. Bergadano, D. Catalano, and S. Varricchio, *Learning sat-k-DNF formulas from membership queries*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 126–130.

[BV96] F. Bergadano and S. Varricchio, *Learning behaviors of automata from multiplicity and equivalence queries*, SIAM J. Comput., 25 (1996), pp. 1268–1280.

[CDG+91] M. Clausen, A. Dress, J. Grabmeier, and M. Karpinski, *On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields*, Theoret. Comput. Sci., 84 (1991), pp. 151–164.

[HR96] M.-D. A. Huang and A. J. Rao, *Interpolation of sparse multivariate polynomials over large finite fields with applications*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 508–517.

[J94] J. Jackson, *An efficient membership-query algorithm for learning DNF with respect to the uniform distribution*, in Proceeding of the 35th Annual Symposium on Foundations of Computer Science, 1994, IEEE Computer Society.

[J95] J. Jackson, *On Learning DNF and Related Circuit Classes from Helpful and Not-So-Helpful Teachers*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1995.

[K94] R. Khardon, *On using the Fourier transform to learn disjoint DNF*, Inform. Process. Lett., 49 (1994), pp. 219–222.

[KM93] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput., 22 (1993), pp. 1331–1348.

[OSK94] H. Ohnishi, H. Seki, and T. Kasami, *A polynomial time learning algorithm for recognizable series*, IEIICE Transactions on Information and Systems, E77-D(10)(5) (1994), pp. 1077–1085.

[RB89] R. M. Roth and G. M. Benedek, *Interpolation and approximation of sparse multivariate polynomials over GF*(2), SIAM J. Comput., 20 (1991), pp. 291–314.

[SS93] R. E. Schapire and L. M. Sellie, *Learning sparse multivariate polynomial over a field with queries and counterexamples*, in Proceedings of the Sixth Annual ACM Workshop on Computational Learning Theory, 1993.

[Val84] L. Valiant, *A theory of the learnable*, Commun. ACM, 27 (1984), pp. 1134–1142.

[Z90] R. Zippel, *Interpolating polynomials from their values*, J. Symbolic Comput., 9 (1990), pp. 375–403.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1926–1947

A DETERMINISTIC POLYNOMIAL-TIME ALGORITHM FOR HEILBRONN'S PROBLEM IN THREE DIMENSIONS*

HANNO LEFMANN† AND NIELS SCHMITT‡

Abstract. Heilbronn conjectured that among arbitrary n points in the two-dimensional unit square $[0,1]^2$, there must be three points which form a triangle of area $O(1/n^2)$. This conjecture was disproved by a nonconstructive argument of Komlós, Pintz, and Szemerédi [*J. London Math. Soc.*, 25 (1982), pp. 13–24], who showed that for every n there exists a configuration of n points in the unit square $[0,1]^2$ where all triangles have area $\Omega(\log n/n^2)$. Here we will consider a three-dimensional analogue of this problem and show how to find deterministically in polynomial time n points in the unit cube $[0,1]^3$ such that the volume of every tetrahedron among these n points is $\Omega(\log n/n^3)$.

Key words. Heilbronn's triangle problem, arrangements of simplices, independent sets in hypergraphs

AMS subject classifications. 68W25, 68R05, 05C69

PII. S0097539701395115

1. Introduction.

An old conjecture of Heilbronn states that for every distribution of n points in the two-dimensional unit square $[0,1]^2$ (or unit disc) there exist three distinct points which form a triangle of area $O(1/n^2)$. Erdős observed that this conjecture, if true, would be best possible, as for n a prime the points $(i, i^2 \bmod n)_{i=0,\ldots,n-1}$ on the moment-curve in the $n \times n$ grid would show after rescaling; see [2]. However, Komlós, Pintz, and Szemerédi [15] disproved Heilbronn's conjecture by proving that for every n there exists a configuration of n points in the unit square $[0,1]^2$ with every three points forming a triangle of area $\Omega(\log n/n^2)$. Using techniques from derandomization, this existence argument was made constructive in [6], where a polynomial time algorithm was given, which finds n points in $[0,1]^2$ achieving this lower bound $\Omega(\log n/n^2)$ on the minimum triangle area. Upper bounds on Heilbronn's triangle problem were given by Roth [17], [18], [19], [20], [21] and Schmidt [23] in a series of papers (see Rothschild and Straus [22] for related results), and the current best upper bound $O(1/n^{8/7-\varepsilon})$ for arbitrarily small $\varepsilon > 0$ is due to Komlós, Pintz, and Szemerédi [14].

Using arguments from Kolmogorov complexity, Jiang, Li, and Vitány [12] recently proved that if n points are dropped uniformly at random and independently of each other in the unit square $[0,1]^2$, then the expected value of the smallest area of a triangle among these n points is $\Theta(1/n^3)$.

Also, Barequet [3] recently considered a k-dimensional version of Heilbronn's problem. For a subset $S = \{p_0, \ldots, p_k\} \subset \mathbb{R}^k$ of $(k+1)$ points, the set $S^* = \{p_0 + \sum_{i=1}^k \lambda_i \cdot (p_i - p_0) \mid \sum_{i=1}^k \lambda_i \leq 1; \lambda_1, \ldots, \lambda_k \in [0,1]\}$ is called a *simplex*. If $k = 3$, then S^* is called a *tetrahedron*. The *volume* of the simplex $S^* \subset \mathbb{R}^k$ is defined

*Received by the editors September 12, 2001; accepted for publication (in revised form) May 31, 2002; published electronically October 31, 2002. A preliminary version of this paper appeared in *Proceedings of the 5th Latin American Symposium on Theoretical Informatics LATIN'02*, Lecture Notes in Comput. Sci. 2286, Springer-Verlag, New York, pp. 165–180.

http://www.siam.org/journals/sicomp/31-6/39511.html

†Fakultät für Informatik, Technische Universität Chemnitz, Straße der Nationen 62, D-09107 Chemnitz, Germany (lefmann@informatik.tu-chemnitz.de).

‡Lehrstuhl Mathematik und Informatik, Fakultät für Mathematik, Ruhr-Universität Bochum, D-44780 Bochum, Germany (nschmitt@lmi.ruhr-uni-bochum.de).

by $\mathrm{vol}(S^*) := 1/k \cdot h \cdot \mathrm{vol}(S')$, where h is the distance of p_k to the affine space generated by $p_0, \ldots, p_{k-1}$ and S' is in this space the simplex generated by $p_0, \ldots, p_{k-1}$.

For given dimension $k \geq 3$, Barequet showed that for every n there exist n points in the k-dimensional unit cube $[0,1]^k$ such that the minimum volume of every simplex spanned by any $(k+1)$ of these n points is $\Omega(1/n^k)$. Barequet gave three different approaches for proving his lower bound. The first one, for dimension $k = 3$, uses a Greedy-type argument (also see [23] for the case $k = 2$), and Barequet obtained a configuration of n points in the three-dimensional unit cube $[0,1]^3$ such that the minimum volume of every tetrahedron is $\Omega(1/n^4)$. Indeed, this Greedy-type argument yields an on-line algorithm for the problem. The second approach yields a better lower bound, was worked out for arbitrary but fixed dimension $k \geq 3$, and uses a random argument: $2n$ points are dropped uniformly at random and independently of each other in the k-dimensional unit cube $[0,1]^k$. For some suitable constant $c_k > 0$, the expected number of $(k+1)$-point simplices with volume at most $\beta := c_k/n^k$ is at most n. Deleting one point from every such small simplex with volume at most β yields the existence of n points in $[0,1]^k$ with every $(k+1)$-point simplex having minimum volume $\Omega(1/n^k)$. The third approach, however, is similar to Erdős' construction (and according to Bollobás [5] was known to him), namely, taking the points $P_l = ((l^j \bmod n)/n)_{j=1,\ldots,k}$ for $l = 0, 1, \ldots, n-1$ on the moment-curve. The volume of any $(k+1)$-point simplex is given by a Vandermonde determinant, which cannot vanish for n a prime, rescaled by a factor $\Theta(1/n^k)$ and this gives a minimum value for the volume of any $(k+1)$ points of these n points on the moment-curve of $\Omega(1/n^k)$.

The corresponding problem for dimension $k = 1$ is trivial as there are always n points in the unit interval $[0,1]$ with minimum distance between two distinct points $\Omega(1/n)$, and this bound cannot be improved.

In [16] Barequet's lower bound was improved by a factor $\Theta(\log n)$ for dimensions $k \geq 3$ using a probabilistic existence argument based on a variant of Theorem 2.2. For the proof the continuous structure of the unit cube $[0,1]^k$ was crucial.

THEOREM 1.1 (see [16]). *For every fixed integer $k \geq 2$ and for every n, there exists a configuration of n points in the k-dimensional unit cube $[0,1]^k$ such that the volume of any simplex spanned by any $(k+1)$ points is $\Omega(\log n/n^k)$.*

Here we will give for dimension $k = 3$ a deterministic polynomial-time algorithm for the result in Theorem 1.1.

THEOREM 1.2. *For every positive integer n, one can find deterministically in polynomial time a configuration of n points in the unit cube $[0,1]^3$ such that the volume of any tetrahedron spanned by any four of these points is $\Omega(\log n/n^3)$.*

The proof of Theorem 1.2 is based on techniques from combinatorics and number theory. Some of our arguments are given for the case of arbitrary dimension $k \geq 3$, where appropriate. However, so far we are able only to provide a deterministic polynomial-time algorithm for the case $k = 3$.

2. Hypergraphs. In our arguments we will use hypergraphs. It will turn out that the notions *independence number of a hypergraph* and *2-cycles* are important in our considerations.

DEFINITION 2.1. *Let $\mathcal{G} = (V, \mathcal{E})$ be a hypergraph; i.e., each edge $E \in \mathcal{E}$ is a nonempty subset of V of arbitrary cardinality. The hypergraph $\mathcal{G}$ is* k-uniform *if every edge $E \in \mathcal{E}$ contains exactly k vertices.*

A subset $I \subseteq V$ is called independent *if I contains no edge $E \in \mathcal{E}$. The largest size of an independent set in $\mathcal{G}$ is called the* independence number $\alpha(\mathcal{G})$.

In a k-uniform hypergraph $\mathcal{G} = (V, \mathcal{E})$, $k \geq 3$, a 2-cycle *is a pair $\{E_1, E_2\}$ of distinct edges $E_1, E_2 \in \mathcal{E}$ with $|E_1 \cap E_2| \geq 2$. A 2-cycle $\{E_1, E_2\}$ in $\mathcal{G}$ is called* $(2, j)$-cycle *if $|E_1 \cap E_2| = j$, where $j = 2, \ldots, k-1$.*

Let $B_k(\rho) = \{x \in \mathbb{R}^k \mid \|x\| \leq \rho\} \subset \mathbb{R}^k$ be the k-dimensional ball around the origin with radius ρ. We will reformulate our combinatorial, geometrical problem as a problem of finding a large independent set in a suitably defined hypergraph. To do so, we will discretize the three-dimensional search space $[0,1]^3$; namely, we will consider only points from the set $B_3(\rho) \cap \mathbb{Z}^3$, where ρ will be of suitable size, i.e., polynomial in n. With this discretization, we also have to take care of tetrahedra of volume 0—these are degenerate tetrahedra.

For some parameter $\beta > 0$ and for the given set of grid-points in $B_3(\rho) \cap \mathbb{Z}^3$, we form a hypergraph $\mathcal{G}(\beta) = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ with the vertex set V being this set $B_3(\rho) \cap \mathbb{Z}^3$ of $\Theta(\rho^3)$ grid-points. The edges contain three or four vertices. The 4-element edges $E \in \mathcal{E}_4$ are determined by all subsets of four points from the set $B_3(\rho) \cap \mathbb{Z}^3$, no three on a line, which form a tetrahedron of volume at most β, where later we will set $\beta := \rho^3 \cdot \log n / n^3$. For technical reasons we will also use 3-element edges $E' \in \mathcal{E}_3$, determined by triples of points from $B_3(\rho) \cap \mathbb{Z}^3$ which lie on a line. Then an independent set in this hypergraph $\mathcal{G}(\beta)$ corresponds to a subset of points in the set $B_3(\rho) \cap \mathbb{Z}^3$, where no tetrahedron has "small" volume, i.e., all tetrahedra have a volume bigger than β, which after rescaling yields the desired result. In order to show the existence of a large independent set, we will use the following result due to Ajtai, Komlós, Pintz, Spencer, and Szemerédi [1] (see [10]), stated here in an algorithmic variant proven in [4] (see [11]).

THEOREM 2.2 (see [1], [4], [10], [11], [7]). *Let $k \geq 3$ be a fixed integer. Let $\mathcal{G} = (V, \mathcal{E})$ be a k-uniform hypergraph with average degree $t^{k-1} = k \cdot |\mathcal{E}|/|V|$. If for some constant $\gamma > 0$ the hypergraph $\mathcal{G}$ contains at most*

$$|V| \cdot t^{2k-j-1-\gamma}$$

$(2, j)$-cycles for $j = 2, \ldots, k-1$, then one can find in $\mathcal{G}$ in polynomial time an independent set of size

$$\Omega\left(\frac{|V|}{t} \cdot (\log t)^{1/(k-1)}\right). \tag{1}$$

If, in addition to the k-element edges, the hypergraph $\mathcal{G} = (V, \mathcal{E})$ contains also at most $|V| \cdot t^{i-1-\delta}$ many i-element edges for some $\delta > 0$ and $i = 2, \ldots, k-1$, then (1) *holds, too.*

If the parameter t^{k-1} is an upper bound on the average degree of the hypergraph $\mathcal{G}$ with respect to the k-element edges, then (1) *holds, too.*

In recent years, several applications of Theorem 2.2 have been found; see [4]. Here we will give another application of this deep result.

A main part of the proof of Theorem 1.2 consists of counting the degenerate (respectively, nondegenerate) tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$. The road map for its proof is as follows. We want to apply Theorem 2.2 for $k = 4$ to the hypergraph $\mathcal{G}(\beta)$. To do so, in sections 4 and 5 we will bound from above the number of 3- and 4-element edges in $\mathcal{G}(\beta)$, which are the number of collinear triples of points and the number of tetrahedra of volume at most β in $B_3(\rho) \cap \mathbb{Z}^3$, respectively. Then, in section 6 we will estimate the numbers of $(2, j)$-cycles, $j = 2, 3$, among the 4-element edges in $\mathcal{G}(\beta)$, that is, the number of pairs of tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$ with volume at most β, which

have j vertices in common. With these estimates we will see that the assumptions of Theorem 2.2 are fulfilled, and the desired result will follow.

In section 3, we will first recall and explain some tools, which we will use in our arguments.

3. Grids in $\mathbb{Z}^k$. We will use some results from linear algebra and number theory, which will be stated in the following.

3.1. Grids.

DEFINITION 3.1. *A* grid L *of* $\mathbb{Z}^k$ *is a subset of* $\mathbb{Z}^k$*, which is generated by all linear combinations of some linearly independent vectors* $q_1, \dots, q_m \in \mathbb{Z}^k$*, where all coefficients are integers; i.e.,* $L = \mathbb{Z}q_1^\top + \cdots + \mathbb{Z}q_m^\top$.

The parameter $m = \operatorname{rank}(L)$ *is called the* rank *of the grid* L*, and the set* $Q = \{q_1, \dots, q_m\}$ *is called the* basis *of* L.

DEFINITION 3.2. *Let* $Q = \{q_1, \dots, q_m\} \subset \mathbb{Z}^k$ *be a set of linearly independent vectors.*

(i) *The* $k \times m$ generator matrix *of* Q *(up to the ordering of the vectors) is defined by*

$$G(Q) := (q_1, \dots, q_m)_{k \times m}.$$

(ii) *The* fundamental parallelepiped F_Q *of* Q *is the following set:*

$$F_Q := \left\{ \sum_{i=1}^{m} \alpha_i \cdot q_i \mid 0 \le \alpha_i \le 1,\ i = 1, \dots, m \right\} \subseteq \mathbb{R}^k .$$

The vertices *of the fundamental parallelepiped* F_Q *are all the points* $\sum_{i=1}^{m} \alpha_i \cdot q_i$ *with* $\alpha_1, \dots, \alpha_m \in \{0, 1\}$.

(iii) *The* volume *of the fundamental parallelepiped* $F_Q \subseteq \mathbb{R}^k$ *of* Q *is given by*

$$\operatorname{vol}(F_Q) := \left(\det(G(Q)^\top \cdot G(Q))\right)^{1/2},$$

where $G(q)^\top$ *is the transpose of the generator matrix* $G(Q)$.

The following result can be found in [8].

LEMMA 3.3 (see [8]). *Let* Q *and* Q' *be two bases of a grid* L *in* $\mathbb{Z}^k$*. Then the volumes of the corresponding fundamental parallelepipeds are equal; i.e.,* $\operatorname{vol}(F_Q) = \operatorname{vol}(F_{Q'})$*. The parameter* $d(L) := \operatorname{vol}(F_Q)$ *is called the* determinant *of the grid* L.

For integers $a_1, \dots, a_n \in \mathbb{Z}$, which are not all equal to 0, let $\gcd(a_1, \dots, a_n)$ denote the *greatest common divisor* of $a_1, \dots, a_n$. From elementary number theory we recall the well-known lemma of Bézout.

LEMMA 3.4. *Let* $a_1, \dots, a_k \in \mathbb{Z}$ *be integers, which are not all equal to* 0*. Then there exist integers* $y_1, \dots, y_k \in \mathbb{Z}$ *such that*

$$a_1 \cdot y_1 + \cdots + a_k \cdot y_k = \gcd(a_1, \dots, a_k) .$$

LEMMA 3.5 (see [9]). *Let* $a = (a_1, \dots, a_k) \in (\mathbb{Z} \setminus \{0\})^k$ *be a sequence of nonzero integers with* $\gcd(a_1, \dots, a_k) = 1$*. Then the set* L *of all solutions in* $\mathbb{Z}^k$ *of the equation*

$$a_1 \cdot X_1 + \cdots + a_k \cdot X_k = 0$$

is a grid in $\mathbb{Z}^k$ *with* $\operatorname{rank}(L) = k - 1$.

An algorithm for determining a basis of a grid $L \subseteq \mathbb{Z}^k$ can be found in [9].

We use the standard scalar product $\langle a, b\rangle := \sum_{i=1}^{k} a_i \cdot b_i$ for vectors $a = (a_1, \ldots, a_k)^\top \in \mathbb{R}^k$ and $b = (b_1, \ldots, b_k)^\top \in \mathbb{R}^k$. The Euclidean distance $\operatorname{dist}(a, b)$ of the corresponding points is defined by $\operatorname{dist}(a, b) := (\sum_{i=1}^{k}(a_i - b_i)^2)^{1/2}$. The length of a vector $a \in \mathbb{R}^k$ is defined by $\|a\| := \sqrt{\langle a, a\rangle}$. For a point $p \in \mathbb{R}^k$ and a real affine subspace $V \subseteq R^k$, let $\operatorname{dist}(p, V) := \min\{\operatorname{dist}(p, v) \mid v \in V\}$. For vectors $q_1, \ldots, q_m \in \mathbb{R}^k$ let $\operatorname{span}(q_1, \ldots, q_m)$ be the linear space over the reals, generated by $q_1, \ldots, q_m$. Where appropriate in this paper, we will not distinguish between points $p \in \mathbb{Z}^k$ and the corresponding vectors $p^\top \in \mathbb{Z}^k$.

The following results can be found in [13].

LEMMA 3.6 (see [13]). *Let V be a $(k-1)$-dimensional linear subspace of $\mathbb{R}^k$, and let $a \in \mathbb{R}^k \setminus \{0^k\}$ be any nonzero vector which is orthogonal to V. The distance $\operatorname{dist}(p, V)$ of every point $p \in \mathbb{R}^k$ to the subspace V is given by*

$$\operatorname{dist}(p, V) = \frac{|\langle p^\top, a\rangle|}{\|a\|} .$$

LEMMA 3.7 (see [13]). *Let $q_1, \ldots, q_m \in \mathbb{R}^k$ be linearly independent vectors. Then, with $U := \operatorname{span}(q_1, \ldots, q_{m-1})$, the volume of the fundamental parallelepiped $F_{\{q_1,\ldots,q_m\}}$ satisfies*

$$\operatorname{vol}(F_{\{q_1,\ldots,q_m\}}) = \operatorname{dist}(q_m, U) \cdot \operatorname{vol}(F_{\{q_1,\ldots,q_{m-1}\}}) .$$

LEMMA 3.8 (see [8]). *Let U and L be grids in $\mathbb{Z}^k$ with $U \subseteq L$ and $\operatorname{rank}(L) = \operatorname{rank}(U) = m$. Then the following holds:*

(i) *There exists a positive integer $\lambda \in \mathbb{N}\setminus\{0\}$ such that $\lambda \cdot L = \{\lambda \cdot x \mid x \in L\} \subseteq U$.*

(ii) *For every basis $b_1, \ldots, b_m$ of L there is a basis $a_1, \ldots, a_m$ of U of the following form:*

$$\begin{aligned} a_1 &= v_{1,1} \cdot b_1, \\ a_2 &= v_{2,1} \cdot b_1 + v_{2,2} \cdot b_2, \\ &\vdots \\ a_m &= v_{m,1} \cdot b_1 + \cdots + v_{m,m} \cdot b_m, \end{aligned} \tag{2}$$

with $v_{i,j} \in \mathbb{Z}$ and $v_{i,i} \neq 0$ for $1 \leq j \leq i \leq m$.

(iii) *For each basis $a_1, \ldots, a_m$ of U, there is a basis $b_1, \ldots, b_m$ of L such that (2) is fulfilled.*

Lemma 3.8 can be made constructive in polynomial time, for example, by using for (ii) a variant of the LLL-algorithm (Lenstra, Lenstra, Lovàsz); see [9].

In our arguments we will use only part (iii) of Lemma 3.8. However, for the proof of part (iii), parts (i) and (ii) are used. A proof of Lemma 3.8 can be found in [8, Theorem I, p. 11–13].

Crucial to our arguments is the following result.

LEMMA 3.9. *Let $k \in \mathbb{N}$ be fixed. Let L be a grid in $\mathbb{Z}^k$ with $\operatorname{rank}(L) = m$, and let $a_1, \ldots, a_m \in L$ be linearly independent. Then there exists a basis $b_1, \ldots, b_m$ of L with $\|b_i\| = O(\max_{j=1,\ldots,m} \|a_j\|)$ for $i = 1, \ldots, m$.*

Proof. The arguments are similar to those in [8, Lemma 8, p. 135–136]. For completeness we include the proof.

By Lemma 3.8(iii) we can find a basis $c_1, \dots, c_m \in \mathbb{Z}^k$ of L such that

$$\begin{aligned} a_1 &= v_{1,1} \cdot c_1, \\ a_2 &= v_{2,1} \cdot c_1 + v_{2,2} \cdot c_2, \\ &\vdots \\ a_m &= v_{m,1} \cdot c_1 + \cdots + v_{m,m} \cdot c_m, \end{aligned} \tag{3}$$

with $v_{i,j} \in \mathbb{Z}$ and $v_{i,i} \neq 0$ for $1 \leq j \leq i \leq m$. With this we will construct the desired basis $b_1, \dots, b_m \in \mathbb{Z}^k$ of L with $\|b_i\| = O(\max_{l=1,\dots,m} \|a_l\|)$ for $i = 1, \dots, m$.

For $i = 1, \dots, m$ we proceed as follows. If $v_{i,i} \in \{+1, -1\}$, then we set $b_i := v_{i,i} \cdot a_i$; hence $\|b_i\| = \|a_i\|$. Otherwise, for $|v_{i,i}| \geq 2$, we solve the system of equations (3) successively for $c_1, \dots, c_m$, and we obtain

$$c_i = \frac{1}{v_{i,i}} \cdot a_i + l_{i,i-1} \cdot a_{i-1} + \cdots + l_{i,1} \cdot a_1$$

for $i = 1, \dots, m$ with rational numbers $l_{i,i-1}, \dots, l_{i,1} \in \mathbb{Q}$. For each $j < i$ we choose integers $t_{i,j} \in \mathbb{Z}$ such that

$$|t_{i,j} + l_{i,j}| \leq \frac{1}{2} .$$

Then, we set $k_{i,j} := t_{i,j} + l_{i,j}$ for $j < i$ and $k_{i,i} := 1/v_{i,i}$ and fix b_i by

$$\begin{aligned} b_i &:= k_{i,i} \cdot a_i + k_{i,i-1} \cdot a_{i-1} + \cdots + k_{i,1} \cdot a_1 \\ &= c_i + t_{i,i-1} \cdot a_{i-1} + \cdots + t_{i,1} \cdot a_1; \end{aligned}$$

hence $b_i \in L$, and $b_1, \dots, b_m$ is a basis of L. By construction we have $|k_{i,j}| \leq 1/2$ for $j \leq i$; hence

$$\|b_i\| \leq \|k_{i,i} \cdot a_i\| + \|k_{i,i-1} \cdot a_{i-1}\| + \cdots + \|k_{i,1} \cdot a_1\| \leq \frac{i}{2} \cdot \max_{j=1,\dots,i} \|a_j\| . \qquad \square$$

3.2. Maximal grids in $\mathbb{Z}^k$. In our arguments we will use the notion of *maximal* grids.

DEFINITION 3.10. *A grid L in $\mathbb{Z}^k$ is called* m-maximal *if* $\operatorname{rank}(L) = m$ *and there exists no other grid L' in $\mathbb{Z}^k$ with $rank(L') = m$, which contains L as a proper subset.*

A vector $a = (a_1, \dots, a_k)^\top \in \mathbb{Z}^k \setminus \{0^k\}$ is called *primitive*, if $\gcd(a_1, \dots, a_k) = 1$ and $a_j > 0$ for $j = \min\{i \mid a_i \neq 0\}$.

For a subset $A = \{x_1, \dots, x_m\} \subseteq \mathbb{R}^k$ of vectors, let $A^\perp = \{a \in \mathbb{R}^k \mid \langle a, x_1 \rangle = \cdots = \langle a, x_m \rangle = 0\}$ be the *orthogonal of A*.

Mainly we will deal here with $(k-1)$-maximal grids in $\mathbb{Z}^k$.

LEMMA 3.11. *Let $a = (a_1, \dots, a_k)^\top \in \mathbb{Z}^k \setminus \{0^k\}$ be an integer-valued vector, where not all entries are equal to 0. Then the set $L_a = (\mathbb{R} \cdot a)^\perp \cap \mathbb{Z}^k$ of all solutions of the equation $a_1 \cdot X_1 + \cdots + a_k \cdot X_k = 0$ over $\mathbb{Z}^k$ is a $(k-1)$-maximal grid in $\mathbb{Z}^k$.*

Proof. By linearity and symmetry we can assume that $\gcd(a_1, \dots, a_k) = 1$, that $a_i \neq 0$ for $i = 1, \dots, r$, and that $a_i = 0$ for $i > r$. By Lemma 3.5 the set of all solutions of the equation $a_1 \cdot X_1 + \cdots + a_r \cdot X_r = 0$ over $\mathbb{Z}^r$ is a grid L^* in $\mathbb{Z}^r$ with $\operatorname{rank}(L^*) = r-1$. Then the set of all solutions of the equation $a_1 \cdot X_1 + \cdots + a_k \cdot X_k = 0$ over $\mathbb{Z}^k$ is the grid $L = L^* \times \mathbb{Z}^{k-r}$ in $\mathbb{Z}^k$ with $\operatorname{rank}(L) = k-1$.

To see the maximality of the grid L, let L' be a grid in $\mathbb{Z}^k$ with $\operatorname{rank}(L') = k-1$, and $L \subseteq L'$. The set $V \subseteq \mathbb{R}^k$ of all solutions of the equation $a_1 \cdot X_1 + \cdots + a_k \cdot X_k = 0$ over $\mathbb{R}^k$ is a $(k-1)$-dimensional linear subspace of $\mathbb{R}^k$. Then we have $L' \subseteq \operatorname{span}(L') = \operatorname{span}(L) = V$; hence each vector in L' is a solution of the equation $a_1 \cdot X_1 + \cdots + a_k \cdot X_k = 0$, and thus $L' \subseteq L$. Therefore, L is a $(k-1)$-maximal grid in $\mathbb{Z}^k$. □

LEMMA 3.12. *For every grid L in $\mathbb{Z}^k$ with $\operatorname{rank}(L) = k-1 \geq 1$ there is exactly one primitive vector $a_L = (a_1, \ldots, a_k)^\top \in \mathbb{Z}^k \setminus \{0^k\}$ with $a_L \perp L$; i.e., $\langle a_L, x^\top \rangle = 0$ for each $x \in L$.*

Proof. Let $Q \subseteq \mathbb{Z}^k$ be a basis of L, and let $G(Q)$ be its generator matrix. The system of linear equations

$$G(Q)^\top \cdot (X_1, \ldots, X_k)^\top = 0 \tag{4}$$

has a nontrivial solution $(a_1, \ldots, a_k) \in \mathbb{Z}^k \setminus \{0^k\}$, and every solution X satisfies $X \perp L$. Dividing each entry of $(a_1, \ldots, a_k)$ by $\gcd(a_1, \ldots, a_k)$ and possibly multiplying the resulting vector by -1, we obtain a primitive vector $a_L = (a'_1, \ldots, a'_k)^\top \in \mathbb{Z}^k \setminus \{0^k\}$, which is a solution of (4). Since the rank of the matrix $G(Q)$ is equal to $k-1$, the solutions of the system (4) in $\mathbb{R}^k$ form a one-dimensional subspace, and therefore the vector a_L is unique. □

COROLLARY 3.13.

(i) *For every grid L' in $\mathbb{Z}^k$ with $\operatorname{rank}(L') = k-1$ there is exactly one $(k-1)$-maximal grid L in $\mathbb{Z}^k$ with $L' \subseteq L$.*

(ii) *There is a bijective mapping between the set of all $(k-1)$-maximal grids L in $\mathbb{Z}^k$ and the set of all primitive vectors a_L in $\mathbb{Z}^k$; i.e., $a_L \in \mathbb{Z}^k$ is the unique primitive normal vector of the grid L.*

DEFINITION 3.14. *Let L be a $(k-1)$-maximal grid in $\mathbb{Z}^k$. A* residue class *of L is a set L' of the form $L' = x + L$ with $x \in \mathbb{Z}^k$.*

LEMMA 3.15. *Let L be a $(k-1)$-maximal grid in $\mathbb{Z}^k$ with primitive normal vector $a_L \in \mathbb{Z}^k$. Then there exists a point $v \in \mathbb{Z}^k \setminus L$ such that $\mathbb{Z}^k$ can be partitioned into the* residue classes *$s \cdot v + L$, $s \in \mathbb{Z}$, i.e.,*

$$\mathbb{Z}^k = \biguplus_{s \in \mathbb{Z}} (s \cdot v + L) \,,$$

where $\uplus$ symbolizes the union of pairwise disjoint sets.

Moreover, for each point $x \in L$ it is

$$\operatorname{dist}(s \cdot v + x, \operatorname{span}(L)) = \frac{|s|}{\|a_L\|} \,.$$

Proof. For every point $x \in L$ we have $\langle x^\top, a_L \rangle = 0$ and for each point $v \in \mathbb{Z}^k \setminus L$ it is $\langle v^\top, a_L \rangle \in \mathbb{Z} \setminus \{0\}$. As a_L is primitive, the greatest common divisor of its entries is equal to 1, and by Lemma 3.4 there exists a point $v \in \mathbb{Z}^k \setminus L$ such that $\langle v^\top, a_L \rangle = 1$.

Using $a_L \perp L$ and Lemma 3.6 we infer the following for each integer $s \in \mathbb{Z}$ and each point $x \in L$:

$$\begin{aligned} \operatorname{dist}(s \cdot v + x, \operatorname{span}(L)) &= \frac{|\langle s \cdot v^\top + x^\top, a_L \rangle|}{\|a_L\|} \\ = \frac{|\langle s \cdot v^\top, a_L \rangle + \langle x^\top, a_L \rangle|}{\|a_L\|} &= \frac{|s|}{\|a_L\|} \,; \end{aligned}$$

hence distinct residue classes of L have a distance which is a multiple of $1/\|a_L\|$.

Now let $p \in \mathbb{Z}^k$ be an arbitrary point. We will show that p is contained in some residue class of L. With $s := \langle p^\top, a_L\rangle \in \mathbb{Z}$ we have

$$\langle p^\top - s \cdot v^\top, a_L\rangle = \langle p^\top, a_L\rangle - s \cdot \langle v^\top, a_L\rangle = s - s = 0\,;$$

thus $p - s \cdot v \in L$; hence $p \in s \cdot v + L$.

To see that $(s \cdot v + L) \cap (t \cdot v + L) = \emptyset$ for $s \neq t$, we assume to the contrary that $s \cdot v + x = t \cdot v + y$ for some $x, y \in L$. Then we have $(s-t) \cdot v = y - x$, and since $v \notin L$ but $y - x \in L$, we conclude that $s = t$, which is a contradiction. □

LEMMA 3.16. *Let L be a $(k-1)$-maximal grid in $\mathbb{Z}^k$ with primitive normal vector $a_L = (a_1, \ldots, a_k)^\top \in \mathbb{Z}^k$ and with basis Q. Then the determinant $d(L)$ of L, i.e., the volume of the fundamental parallelepiped determined by Q, satisfies*

$$d(L) = \mathrm{vol}(F_Q) = \|a_L\|.$$

Proof. Let $Q = \{q_1, \ldots, q_{k-1}\}$ be a basis of L. As stated in Lemma 3.15 there exists a vector $q_k \in \mathbb{Z}^k$ such that $\mathbb{Z}^k = \uplus_{s\in\mathbb{Z}}(s \cdot q_k + L)$; i.e., $\{q_1, \ldots, q_k\}$ is a basis of $\mathbb{Z}^k$. Since $\mathrm{dist}(q_k, \mathrm{span}(L)) = 1/\|a_L\|$, we infer by Lemma 3.7, where $U := \mathrm{span}(q_1, \ldots, q_{k-1})$, that

$$1 = d(\mathbb{Z}^k) = \mathrm{vol}(F_{\{q_1,\ldots,q_k\}}) = \frac{1}{\|a_L\|} \cdot d(L). \qquad \square$$

3.3. Simplices and maximal grids in $\mathbb{Z}^k$.

DEFINITION 3.17. *For a subset $S = \{p_0, \ldots, p_k\} \subset \mathbb{R}^k$ of $(k+1)$ points, the set*

$$S^* = \left\{ p_0 + \sum_{i=1}^{k} \lambda_i \cdot (p_i - p_0) \mid \sum_{i=1}^{k} \lambda_i \leq 1; \lambda_1, \ldots, \lambda_k \in [0,1] \right\}$$

is called a simplex. *For short, we identify S and S^* and call each of them* simplex *and specify by calling the points $p \in \{p_0, \ldots, p_k\}$, which satisfy $(\{p_0, \ldots, p_k\}\backslash\{p\})^* \neq S^*$,* vertices *of the simplex.*

(i) *The* rank *of the simplex S is defined by*

$$\mathrm{rank}(S) = \dim(\mathrm{span}(\{p_1 - p_0, \ldots, p_k - p_0\}))\,.$$

(ii) *The simplex S is* nondegenerate, *if* $\mathrm{rank}(S) = k$. *If* $\mathrm{rank}(S) < k$, *we call S a* degenerate simplex. *A simplex $S = \{p_0, \ldots, p_k\} \subset \mathbb{R}^k$ is called a* triangle *for $k = 2$, and for $k = 3$ it is called a* tetrahedron.

(iii) *The* volume *of a simplex $S = \{p_0, \ldots, p_k\} \subset \mathbb{R}^k$ is defined by*

$$\mathrm{vol}(S) = \frac{1}{k} \cdot h \cdot \mathrm{vol}(S')\,,$$

where h is the distance of p_k to the affine space generated by $p_0, \ldots, p_{k-1}$, and $S' = \{p_0, \ldots, p_{k-1}\}$.

Recall that $B_k(\rho) = \{x \in \mathbb{R}^k \mid \|x\| \leq \rho\} \subset \mathbb{R}^k$ is the k-dimensional ball around the origin with radius $\rho \in \mathbb{R}_0^+$. The following result will be used frequently in our arguments.

LEMMA 3.18. *Let $S \subseteq B_k(\rho) \cap \mathbb{Z}^k$ be a set of points with* $\mathrm{rank}(S) \leq k - 1$. *Then there exists a $(k-1)$-maximal grid L of $\mathbb{Z}^k$ such that S is contained in some*

residue class $v + L$ of L for some $v \in \mathbb{Z}^k$, and L has a basis $q_1, \dots, q_{k-1} \subset \mathbb{Z}^k$ with $\max_{i=1,\dots,k-1} \|q_i\| = O(\rho)$.

Proof. Let $S = \{p_0, \dots, p_m\} \subseteq B_k(\rho) \cap \mathbb{Z}^k$ with $r = \operatorname{rank}(S)$. The vectors $p_1 - p_0, \dots, p_m - p_0$ span a grid L' in $\mathbb{Z}^k$ with $\operatorname{rank}(L') = r$, and have length $\|p_i - p_0\| \le 2 \cdot \rho$ for $i = 1, \dots, m$. We take $(k - 1 - r)$ unit vectors from $\mathbb{Z}^k \setminus L'$, add them to L', and we obtain a grid L'' of $\mathbb{Z}^k$ with $\operatorname{rank}(L'') = k - 1$ and $L' \subseteq L''$. By Corollary 3.13 there is exactly one $(k-1)$-maximal grid L of $\mathbb{Z}^k$ with $L' \subseteq L'' \subseteq L$. By Lemma 3.9 we can find a basis $q_1, \dots, q_{k-1}$ of L with $\|q_i\| = O(\rho)$ for $i = 1, \dots, k-1$. Then we have $S \subseteq p_0 + L$. ∎

THEOREM 3.19. *Let $k \in \mathbb{N}$ be fixed. Let L be a $(k-1)$-maximal grid of $\mathbb{Z}^k$ with primitive normal vector $a_L \in \mathbb{Z}^k$, and let $Q = \{q_1, \dots, q_{k-1}\}$ be a basis of L with* $\max_i \|q_i\| = O(\rho)$. *Then the following holds:*

(i) *The primitive normal vector a_L satisfies $\|a_L\| = O(\rho^{k-1})$.*

(ii) *There are $O(\rho \cdot \|a_L\|)$ different residue classes L' of L with $L' \cap B_k(\rho) \neq \emptyset$.*

(iii) *For every residue class L' of L it is $|L' \cap B_k(\rho)| = O\big(\rho^{k-1}/\|a_L\|\big)$.*

Proof. (i) By Lemma 3.16 we have $\|a_L\| = d(L) = \operatorname{vol}(F_Q)$, and by using the assumption $\max_i \|q_i\| = O(\rho)$ we know that $F_Q \subseteq B_k(c \cdot \rho)$ for some constant $c > 0$. Since $\dim(F_Q) = k - 1$, the volume $\operatorname{vol}(F_Q)$ of the parallelepiped F_Q is $O(\rho^{k-1})$.

(ii) By Lemma 3.15 the distances of different residue classes of L are multiples of $1/\|a_L\|$. The Euclidean distance between any two points in the ball $B_k(\rho)$ is at most $2 \cdot \rho$; hence $O(\rho \cdot \|a_L\|)$ distinct residue classes of L have a nonempty intersection with $B_k(\rho)$.

(iii) The volume of a $(k-1)$-dimensional space S intersected with $B_k(\rho)$ is $O(\rho^{k-1})$. Since $F_Q \subseteq B_k(c \cdot \rho)$ and $\operatorname{vol}(F_Q) = \|a_L\|$, we can cover the set $S \cap B_k(\rho)$ by $O\big(\rho^{k-1}/\|a_L\|\big)$ distinct translates of the parallelepiped F_Q. As L is maximal, the interior of F_Q (only the vertices are excluded) contains no points from L, which finishes the proof. ∎

3.4. Representations by sums of squares. We will use some results from elementary number theory. For integers $k, d \in \mathbb{N}$ let $r_k(d)$ be the number of vectors $(x_1, \dots, x_k)^\top \in \mathbb{Z}^k$ with $x_1^2 + \dots + x_k^2 = d$.

LEMMA 3.20. *For fixed integers $k \in \mathbb{N}$ and all integers $n \in \mathbb{N}$ it is*

$$\sum_{d=1}^{n} r_k(d) = \Theta(n^{k/2}) \,. \tag{5}$$

Proof. The sum $\sum_{d=1}^{n} r_k(d)$ counts the number of grid-points in $\mathbb{Z}^k$ in the k-dimensional ball $B_k(\sqrt{n})$. If we put around each of these grid-points k-dimensional unit cubes $[0,1]^k$, with centers being the grid-points, then all the points of these unit cubes are contained in a k-dimensional ball around the origin with radius $\sqrt{n} + \sqrt{k}/2$, since the diagonal of every k-dimensional unit cube $[0,1]^k$ has length $\sqrt{k}$. Moreover, the unit cubes cover the k-dimensional ball around the origin with radius $\sqrt{n} - \sqrt{k}/2$. Since the number of unit cubes is equal to $\sum_{d=1}^{n} r_k(d)$, we infer

$$\frac{\pi^{k/2}}{\Gamma(k/2+1)} \cdot \left(\sqrt{n} - \frac{\sqrt{k}}{2}\right)^k \le \sum_{d=1}^{n} r_k(d) \le \frac{\pi^{k/2}}{\Gamma(k/2+1)} \cdot \left(\sqrt{n} + \frac{\sqrt{k}}{2}\right)^k \;;$$

thus $\sum_{d=1}^{n} r_k(d) = \Theta(n^{k/2})$. (Recall that $R^k \cdot \pi^{k/2}/\Gamma(k/2+1)$ is the volume of the k-dimensional ball around the origin with radius $R \in \mathbb{R}^+$.) ∎

COROLLARY 3.21. *For fixed integers $k, r \in \mathbb{N}$ and for all integers $n \in \mathbb{N}$, it is*

$$\sum_{d=1}^{n} \frac{r_k(d)}{d^r} = \begin{cases} O\left(n^{k/2-r}\right) & \text{if } k/2-r>0, \\ O\left(\log n\right) & \text{if } k/2-r=0, \\ O(1) & \text{if } k/2-r<0. \end{cases}$$

Proof. We assume without loss of generality that n is a power of 2, i.e., $n = 2^l$. Set $n_i = 2^i$ for $i = 0, \ldots, l$. With (5) we infer for some constant $c_k > 0$ that

$$\sum_{d=1}^{n} \frac{r_k(d)}{d^r} \le \sum_{i=0}^{l} \sum_{d=n_i}^{n_{i+1}} \frac{r_k(d)}{d^r} \le \sum_{i=0}^{l} \sum_{d=n_i}^{n_{i+1}} \frac{r_k(d)}{n_i^r} = \sum_{i=0}^{l} 2^{-i\cdot r} \cdot \sum_{d=n_i}^{n_{i+1}} r_k(d)$$
$$\le \sum_{i=0}^{l} 2^{-i\cdot r} \cdot \sum_{d=1}^{n_{i+1}} r_k(d) \le \sum_{i=0}^{l} 2^{-i\cdot r} \cdot c_k \cdot (n_{i+1})^{k/2} = c_k \cdot 2^{k/2} \cdot \sum_{i=0}^{l} 2^{i\cdot(k/2-r)} .$$

The sum $\sum_{i=0}^{l} 2^{i\cdot(k/2-r)}$ is bounded from above as follows: (i) for $k/2-r>0$ by $O\left(2^{l\cdot(k/2-r)}\right) = O\left(n^{k/2-r}\right)$, (ii) for $k/2-r=0$ by $O(l) = O(\log n)$, and (iii) for $k/2-r<0$ by $O(1)$. ∎

For a $(k-1)$-maximal grid L in $\mathbb{Z}^k$ with primitive normal vector $a_L \in \mathbb{Z}^k$ and a positive integer d, we denote by $r_k(d; a_L)$ the number of grid-points P in L such that the square of the Euclidean distance between P and the origin O is equal to d, i.e., $(\mathrm{dist}(O,P))^2 = d$.

In our arguments we will use the following variants of Lemma 3.20 and Corollary 3.21.

LEMMA 3.22. *Let $k \in \mathbb{N}$ be a fixed integer. Let L be a $(k-1)$-maximal grid in $\mathbb{Z}^k$ with primitive normal vector $a_L \in \mathbb{Z}^k$. For all integers $n \in \mathbb{N}$, it is*

$$\sum_{d=1}^{n} r_k(d; a_L) = O\left(\frac{n^{(k-1)/2}}{\|a_L\|}\right) . \tag{6}$$

Proof. The sum $\sum_{d=1}^{n} r_k(d; a_L)$ is equal to the number of grid-points in L in the k-dimensional ball $B_k(\sqrt{n})$. By Theorem 3.19(iii) this sum $\sum_{d=1}^{n} r_k(d; a_L)$ is $O((\sqrt{n})^{k-1}/\|a_L\|)$, from which inequality (6) follows. ∎

With (6) the following is straightforward with the same arguments used in the proof of Corollary 3.21.

COROLLARY 3.23. *Let $k, r \in \mathbb{N}$ be fixed integers and let L be a $(k-1)$-maximal grid in $\mathbb{Z}^k$ with primitive normal vector $a_L \in \mathbb{Z}^k$. For all positive integers $n \in \mathbb{N}$, it is*

$$\sum_{d=1}^{n} \frac{r_k(d; a_L)}{d^r} = \begin{cases} O\left(\frac{1}{\|a_L\|} \cdot n^{(k-1)/2-r}\right) & \text{if } (k-1)/2-r>0, \\ O\left(\frac{1}{\|a_L\|} \cdot \log n\right) & \text{if } (k-1)/2-r=0, \\ O\left(\frac{1}{\|a_L\|}\right) & \text{if } (k-1)/2-r<0. \end{cases}$$

4. Degenerate simplices in $B_k(\rho) \cap \mathbb{Z}^k$.

THEOREM 4.1. *Let $k \in \mathbb{N}$ be a fixed integer. The number $D_k(\rho)$ of degenerate simplices in $B_k(\rho) \cap \mathbb{Z}^k$ satisfies $D_k(\rho) = O\left(\rho^{k^2} \cdot \log \rho\right)$.*

The set of all these degenerate simplices in $B_k(\rho) \cap \mathbb{Z}^k$ can be constructed in time polynomial in ρ.

Proof. For fixed $k \in \mathbb{N}$, by inspecting every $(k+1)$-element subset $S \subset B_k(\rho) \cap \mathbb{Z}^k$ of points, we can determine all degenerate simplices in $B_k(\rho) \cap \mathbb{Z}^k$ in time $O(\rho^{k(k+1)})$,

since there are $O\big(\binom{\rho^k}{k+1}\big)$ $(k+1)$-element subsets at all. To check whether or not $\mathrm{vol}(S) = 0$, one simply computes in time $O(1)$ the determinant of the matrix with columns $p_1 - p_0, \ldots, p_k - p_0$.

By Lemma 3.18 each degenerate $(k+1)$-element subset of points in $B_k(\rho) \cap \mathbb{Z}^k$ is contained in a residue class L' of some $(k-1)$-maximal grid L in $\mathbb{Z}^k$, where L has a basis $q_1, \ldots, q_{k-1} \in \mathbb{Z}^k$ with $\|q_i\| = O(\rho)$ for $i = 1, \ldots, k-1$. By Theorem 3.19(i) it suffices to consider all primitive normal vectors $a_L \in \mathbb{Z}^k$ of length $\|a_L\| = O(\rho^{k-1})$ and the corresponding residue classes L' of L with $L' \cap B_3(\beta) \neq \emptyset$.

Having fixed a $(k-1)$-maximal grid L in $\mathbb{Z}^k$, determined by its primitive normal vector $a_L \in \mathbb{Z}^k$, by Theorem 3.19(ii) there are $O(\rho \cdot \|a_L\|)$ residue classes L' of the grid L, which intersect the ball $B_k(\rho)$ in a nonempty set.

By Theorem 3.19(iii) each set $L' \cap B_k(\rho)$ contains $O(\rho^{k-1}/\|a_L\|)$ points. From each set $L' \cap B_k(\rho)$ we can select $(k+1)$ points in $O\big(\binom{\rho^{k-1}/\|a_L\|}{k+1}\big)$ ways to obtain a degenerate simplex. This implies the following upper bound on the number $D_k(\rho)$ of degenerate simplices in $B_k(\rho) \cap \mathbb{Z}^k$:

$$D_k(\rho) = O\left(\sum_{a \in \mathbb{Z}^k,\, \|a\| = O(\rho^{k-1})} \rho \cdot \|a\| \cdot \binom{\rho^{k-1}/\|a\|}{k+1}\right)$$

$$= O\left(\rho^{k^2} \cdot \sum_{a \in \mathbb{Z}^k,\, \|a\| = O(\rho^{k-1})} \frac{1}{\|a\|^k}\right) = O\left(\rho^{k^2} \cdot \sum_{d=1}^{O(\rho^{2k-2})} \frac{r_k(d)}{d^{k/2}}\right) = O(\rho^{k^2} \cdot \log \rho)\,,$$

since by Corollary 3.21 we have $\sum_{d=1}^{n} r_k(d)/d^{k/2} = O(\log n)$. ∎

5. Nondegenerate tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$. From now on we consider only the case of dimension $k = 3$. We will determine for positive reals β the number $N_3(\rho; \beta)$ of nondegenerate tetrahedra $S = \{p_0, \ldots, p_3\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with volume $\mathrm{vol}(S) \leq \beta$. Recall that the *volume* of a (possibly degenerate) tetrahedron S is defined by

$$\mathrm{vol}(S) = \frac{1}{3} \cdot h \cdot G\,,$$

where h is the distance between p_3 and the affine space defined by p_0, p_1, p_2, and G is the area of the triangle p_0, p_1, p_2.

We will show in this section the following result.

THEOREM 5.1. *The number $N_3(\rho; \beta)$ of nondegenerate tetrahedra $S \subseteq B_3(\rho) \cap \mathbb{Z}^3$ with $\mathrm{vol}(S) \leq \beta$ satisfies $N_3(\rho; \beta) = O(\beta \cdot \rho^9)$.*

The set of all these nondegenerate tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$ can be constructed in time polynomial in ρ.

Proof. By inspecting every 4-element subset $S = \{p_0, \ldots, p_3\} \subset B_3(\rho) \cap \mathbb{Z}^3$, we can determine all tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$ with volume at most β in time $O(\rho^{12})$, since checking whether or not $\mathrm{vol}(S) \leq \beta$ can be done in time $O(1)$.

First we consider the number of nondegenerate triangles S in $B_3(\rho) \cap L$ for a 2-maximal grid L in $\mathbb{Z}^3$, where we distinguish whether $\mathrm{area}(S) \geq v$ or $\mathrm{area}(S) \leq v$ for some real number $v > 0$.

LEMMA 5.2. *Let L be a 2-maximal grid in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. Let $v, \beta > 0$ be real numbers. For every residue class L' of L the number of nondegenerate tetrahedra $S = \{p_0, p_1, p_2, p_3\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with $S \setminus \{p_3\} \subseteq L'$ and*

$\operatorname{area}(S \setminus \{p_3\}) \geq v$ *and* $\operatorname{vol}(S) \leq \beta$ *is at most*

$$O\left(\frac{\beta \cdot \rho^8}{v \cdot \|a_L\|^3}\right).$$

Proof. Let L' be a residue class of a 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. By Lemma 3.18 we can assume that L has a basis $q_1, q_2 \in \mathbb{Z}^3$, with $\|q_1\|, \|q_2\| = O(\rho)$. By Theorem 3.19(iii) the set $L' \cap B_3(\rho)$ contains $O(\rho^2/\|a_L\|)$ points; hence we can choose $O(\binom{\rho^2/\|a_L\|}{3})$ different sets of three points from this set. In particular, for every real $v > 0$, we can choose from the set $L' \cap B_3(\rho)$ three vertices of a nondegenerate triangle S' with $\operatorname{area}(S') \geq v$ in $O(\binom{\rho^2/\|a_L\|}{3})$ ways. Since the desired tetrahedra should have volume at most β, the corresponding fourth point has distance $O(\beta/v)$ from the real affine space generated by L'. By Lemma 3.15 the distance between distinct residue classes of L is a multiple of $1/\|a_L\|$, and since for every residue class L'' of L the set $L'' \cap B_3(\rho)$ contains $O(\rho^2/\|a_L\|)$ points, the fourth point can be chosen in

$$O\left(\frac{\beta}{v} \cdot \|a_L\| \cdot \frac{\rho^2}{\|a_L\|}\right) = O\left(\frac{\beta \cdot \rho^2}{v}\right)$$

ways. Altogether we obtain for the number of desired simplices

$$O\left(\binom{\rho^2/\|a_L\|}{3} \cdot \frac{\beta \cdot \rho^2}{v}\right) = O\left(\frac{\beta \cdot \rho^8}{v \cdot \|a_L\|^3}\right). \qquad \square$$

Next we will consider the nondegenerate triangles in $B_3(\rho) \cap \mathbb{Z}^3$ with area at most v. For this case we will use the following lemma.

LEMMA 5.3. *Let L be a 2-maximal grid in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. For every residue class L' of L, and every two fixed distinct points P and Q on L', the number of nondegenerate triangles S in $L' \cap B_3(\rho)$ with vertices P and Q and* $\operatorname{area}(S) \leq v$ *is*

$$O\left(\frac{v \cdot \rho}{\operatorname{dist}(P, Q) \cdot \|a_L\|}\right).$$

Proof. By an affine mapping $f\colon L \longrightarrow \mathbb{Z}^2$ we transform the 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$ (or any residue class L' of L) into the standard two-dimensional rectangular grid $\mathbb{Z}^2$ with basis $(1,0)^\top$ and $(0,1)^\top$. Points in $L \cap B_3(\beta)$ become points within an ellipsoid C. For a basis $a = (a_1, a_2, a_3)^\top \in \mathbb{Z}^3$, $b = (b_1, b_2, b_3)^\top \in \mathbb{Z}^3$ of the grid L, let $f(a) := (1,0)^\top$ and $f(b) := (0,1)^\top$. We can assume that $L' = L$, and that $P = (0,0,0)$ and $Q = \lambda \cdot a + \mu \cdot b$ (for some $\lambda, \mu \in \mathbb{Z}$) are the two given points. Via the mapping $f\colon L' \longrightarrow \mathbb{Z}^2$ we obtain the points $f(P) = P' = (0,0)$ and $f(Q) = Q' = (\lambda, \mu)$.

Points $P, Q, R \in L' \cap B_3(\rho)$ become the grid points $P', Q', R' \in C \cap \mathbb{Z}^2$ in the ellipsoid C. If $\operatorname{area}(P, Q, R) = v$, then $\operatorname{area}(P', Q', R') = v/\|a_L\|$, as can be easily seen.

Let $g = \gcd(\lambda, \mu)$, and set $\lambda' := \lambda/g$ and $\mu' := \mu/g$. The line ℓ (residue class) through the points P' and Q' in $\mathbb{Z}^2$ has the primitive normal vector $a_N := (\mu', -\lambda')^\top$.

To estimate the number of points $R \in L' \cap B_3(\rho)$ such that $\operatorname{area}(P, Q, R) \leq v$, we compute the number of points $R' \in C \cap \mathbb{Z}^2$ such that $\operatorname{area}(P', Q', R') \leq v/\|a_L\|$. The distance of R' to the line ℓ is at most $2 \cdot v/(\|a_L\| \cdot \operatorname{dist}(P', Q'))$. By Lemma 3.15

distinct residue classes ℓ' of ℓ have a distance which is a multiple of $1/\|a_N\|$; thus there are

$$O\left(\frac{v \cdot \|a_N\|}{\|a_L\| \cdot \operatorname{dist}(P', Q')}\right)$$

lines ℓ' with $R' \in \ell'$ to consider. Since the distance between any two points in $B_3(\rho)$ is at most $2 \cdot \rho$, the line ℓ intersects the ellipsoid C in two points with distance D, where

$$D = O\left(\frac{\rho \cdot \operatorname{dist}(P', Q')}{\operatorname{dist}(P, Q)}\right).$$

Every line ℓ' parallel to ℓ intersects the ellipsoid C in two points whose distance D' is at most D. To see this, consider the line ℓ'' obtained by reflecting the line ℓ' at ℓ. By symmetry, the line ℓ'' intersects the ellipsoid C in two points which also have distance D'. From the convexity of the ellipsoid it follows that $D' \le D$. By Lemma 3.16 the distance between points in $\ell' \cap \mathbb{Z}^2$ is a multiple of $\|a_N\|$, and we infer that $|\ell' \cap E \cap \mathbb{Z}^2| = O(D'/\|a_N\|) = O(D/\|a_N\|)$.

Thus, we have the following upper bound on the number of triangles in $L' \cap B_3(\rho)$ with area at most v and with fixed vertices P and Q:

$$O\left(\frac{v \cdot \|a_N\|}{\|a_L\| \cdot \operatorname{dist}(P', Q')} \cdot \frac{D}{\|a_N\|}\right) = O\left(\frac{v \cdot \rho}{\operatorname{dist}(P, Q) \cdot \|a_L\|}\right). \qquad \square$$

COROLLARY 5.4. *Let L be a 2-maximal grid in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. For every residue class L' of L, the number of nondegenerate triangles S in $L' \cap B_3(\rho)$ with* $\operatorname{area}(S) \le v$ *is*

$$O\left(\frac{v \cdot \rho^4}{\|a_L\|^3}\right).$$

Proof. Since $|L' \cap B_3(\rho)| \le |L \cap B_3(\rho)|$ and L' is a residue class of L, we can assume that $L' = L$. By Lemma 3.18 we can assume that the grid L has a basis $q_1, q_2 \in \mathbb{Z}^3$ with $\|q_1\|, \|q_2\| = O(\rho)$. Fix a point $P \in L' \cap B_3(\rho)$; by Theorem 3.19(iii) there are $O(\rho^2/\|a_L\|)$ possibilities for this. For every integer $d \in \mathbb{N}$ there are at most $r_3(d; a_L)$ points Q in $L' \cap B_3(\rho)$ such that $(\operatorname{dist}(P, Q))^2 = d$. With $d = O(\rho^2)$, Lemma 5.3, and Corollary 3.23, the number of nondegenerate triangles S in the set $L' \cap B_3(\rho)$ with $\operatorname{area}(S) \le v$ is

$$\begin{aligned} O\left(\frac{\rho^2}{\|a_L\|} \cdot \sum_{d=1}^{O(\rho^2)} \frac{v \cdot \rho \cdot r_3(d; a_L)}{d^{1/2} \cdot \|a_L\|}\right) &= O\left(\frac{v \cdot \rho^3}{\|a_L\|^2} \cdot \sum_{d=1}^{O(\rho^2)} \frac{r_3(d; a_L)}{d^{1/2}}\right) \\ &= O\left(\frac{v \cdot \rho^4}{\|a_L\|^3}\right). \qquad \square \end{aligned}$$

LEMMA 5.5. *Let L be a 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. Then for every nondegenerate triangle S in L it is* $\operatorname{area}(S) \ge \|a_L\|/2$.

Proof. Let Q be a basis of L. The minimum area of a nondegenerate triangle in L is half of the volume of the fundamental parallelepiped F_Q, where $\operatorname{vol}(F_Q) = \|a_L\|$ by Lemma 3.16. $\square$

LEMMA 5.6. *Let L be a 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. Let L' be any residue class of L. Let $v > 0$ be a real number. The number of nondegenerate tetrahedra $S = \{p_0, \dots, p_3\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with $S \setminus \{p_3\} \subseteq L'$ and* $\operatorname{area}(S \setminus \{p_3\}) \le v$ *and* $\operatorname{vol}(S) \le \beta$ *is*

$$O\left(\frac{\beta \cdot v \cdot \rho^6}{\|a_L\|^4}\right).$$

Proof. By Lemma 3.18 we can assume that the grid L has a basis $q_1, q_2 \in \mathbb{Z}^3$, with $\|q_1\|, \|q_2\| = O(\rho)$. Let $S' = \{p_0, p_1, p_2\}$ be a nondegenerate triangle in $L' \cap B_3(\rho)$ with $\operatorname{area}(S') \le v$. By Corollary 5.4 there are $O(v \cdot \rho^4/\|a_L\|^3)$ of them. By Lemma 5.5 we have $\operatorname{area}(S') \ge \|a_L\|/2$. To select the point $p_3 \in B_3(\rho) \cap \mathbb{Z}^3$, the requirement $\operatorname{vol}(S' \cup \{p_3\}) \le \beta$ has to be satisfied; thus the distance between the point p_3 and the affine space defined by S' is $O(\beta/\|a_L\|)$. By Lemma 3.15 the distance between distinct residue classes of L is a multiple of $1/\|a_L\|$. By Theorem 3.19(iii) for every residue class L' of L the set $L' \cap B_3(\rho)$ contains $O(\rho^2/\|a_L\|)$ points. Hence, for a fixed triangle S', the point p_3 can be chosen in

$$O\left(\frac{\beta}{\|a_L\|} \cdot \|a_L\| \cdot \frac{\rho^2}{\|a_L\|}\right) = O\left(\frac{\beta \cdot \rho^2}{\|a_L\|}\right) \tag{7}$$

ways. Thus, the number of tetrahedra $S = \{p_0, \dots, p_3\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with $\operatorname{vol}(S) \le \beta$ and $S \setminus \{p_3\} \subseteq L'$ and $\operatorname{area}(S \setminus \{p_3\}) \le v$ is

$$O\left(\frac{v \cdot \rho^4}{\|a_L\|^3} \cdot \frac{\beta \cdot \rho^2}{\|a_L\|}\right) = O\left(\frac{\beta \cdot v \cdot \rho^6}{\|a_L\|^4}\right). \qquad \square$$

LEMMA 5.7. *Let L be a 2-maximal grid in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$. For every residue class L' of L the number of nondegenerate tetrahedra $S = \{p_0, p_1, p_2, p_3\} \subseteq B_3(\rho) \cap \mathbb{Z}^3$ with $S \setminus \{p_3\} \subseteq L'$ and* $\operatorname{vol}(S) \le \beta$ *is $O(\beta \cdot \rho^7/\|a_L\|^{7/2})$.*

Proof. From Lemmas 5.2 and 5.6 we infer that, for every residue class L' of a 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$ and for every real $v > 0$, the number of tetrahedra $S = \{p_0, \dots, p_3\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with $S \setminus \{p_3\} \subseteq L'$ and $\operatorname{vol}(S) \le \beta$ is

$$O\left(\frac{\beta \cdot \rho^8}{v \cdot \|a_L\|^3} + \frac{\beta \cdot v \cdot \rho^6}{\|a_L\|^4}\right). \tag{8}$$

We have

$$\frac{\beta \cdot v \cdot \rho^6}{\|a_L\|^4} = \frac{\beta \cdot \rho^8}{v \cdot \|a_L\|^3} \qquad \text{if} \qquad v = \rho \cdot \|a_L\|^{1/2}.$$

For a given vector $a_L \in \mathbb{Z}^3$ we set $v(a_L) := \rho \cdot \|a_L\|^{1/2}$. Then, (8) becomes

$$O\left(\frac{\beta \cdot \rho^8}{v \cdot \|a_L\|^3} + \frac{\beta \cdot v \cdot \rho^6}{\|a_L\|^4}\right) = O\left(\frac{\beta \cdot \rho^7}{\|a_L\|^{7/2}}\right). \qquad \square$$

Now we can finish the proof of Theorem 5.1. By Lemma 3.18 we can assume that the grid L has a basis $q_1, q_2 \in \mathbb{Z}^3$ with $\|q_1\|, \|q_2\| = O(\rho)$; hence $\|a_L\| = O(\rho^2)$ by Theorem 3.19(i). By Theorem 3.19(ii), for a fixed primitive normal vector $a_L \in \mathbb{Z}^3$,

there are $O(\rho \cdot \|a_L\|)$ distinct residue classes L' of the grid L, which intersect $B_3(\rho)$ in a nonempty set. Summing over all possible grids L, we have with Lemma 5.7 that

$$N_3(\rho;\beta) = O\left(\sum_{\substack{a\in\mathbb{Z}^3 \\ \|a\|=O(\rho^2)}} \rho \cdot \|a\| \cdot \frac{\beta\cdot\rho^7}{\|a\|^{7/2}} \right) = O\left(\beta\cdot\rho^8 \cdot \sum_{\substack{a\in\mathbb{Z}^3 \\ \|a\|=O(\rho^2)}} \frac{1}{\|a\|^{5/2}} \right)$$
$$= O\left(\beta\cdot\rho^8 \cdot \sum_{d=1}^{O(\rho^4)} \frac{r_3(d)}{d^{5/4}} \right) = O(\beta\cdot\rho^9),$$

where the last equation follows with Corollary 3.21, i.e., $\sum_{d=1}^{n} r_3(d)/d^{5/4} = O(n^{1/4})$. This finishes the proof of Theorem 5.1. □

6. Properties of the hypergraph $\mathcal{G}(\beta)$. For some real number $\beta > 0$, which will be fixed below, we form a hypergraph $\mathcal{G}(\beta) = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ with the vertex set V being the set $B_3(\rho) \cap \mathbb{Z}^3$ of $\Theta(\rho^3)$ grid-points and $\mathcal{E}_i$ being the set of all i-element edges, $i = 3, 4$. The hypergraph $\mathcal{G}$ contains 3- and 4-element edges. The 4-element edges $E \in \mathcal{E}_4$ are determined by all 4-element subsets of V, no three of them on a line, which form a tetrahedron of volume at most β including degenerate tetrahedra. The 3-element edges $E' \in \mathcal{E}_3$ are formed by all collinear triples from $V = B_3(\rho)\cap\mathbb{Z}^3$. Then an independent set in this hypergraph $\mathcal{G}(\beta)$ corresponds to a set of points in $B_3(\rho)\cap\mathbb{Z}^3$, where all tetrahedra have volume bigger than β. In order to apply Theorem 2.2 we will show that the assumptions there are satisfied; i.e., we will give upper bounds on the numbers of 3- and 4-element edges and the numbers of 2-cycles among the 4-element edges. Set

$$\beta := \frac{\log n}{n^3} \cdot \rho^3,$$

where $\rho = n^{1+\varepsilon}$ for some fixed $\epsilon > 0$; thus $\beta = n^{3\varepsilon} \cdot \log n$.

First we estimate the average degree t^3 of the hypergraph $\mathcal{G} = (V, \mathcal{E}_4)$ among the 4-element edges. By Theorems 4.1 and 5.1 we can bound the number of 4-element edges in $\mathcal{G}$ by

$$|\mathcal{E}_4| = O(\rho^9 \cdot \log\rho + \beta\cdot\rho^9) = O(\beta\cdot\rho^9);$$

hence, with $|V| = \Theta(\rho^3)$, the average degree of $\mathcal{G}$ is

$$\frac{4\cdot|\mathcal{E}_4|}{|V|} = O\left(\frac{\beta\cdot\rho^9}{\rho^3}\right) = O\left(\beta\cdot\rho^6\right).$$

In our application of Theorem 2.2 we will calculate with an upper bound on t; thus from now on for some constant $c > 0$ we use

$$t^3 = c\cdot\beta\cdot\rho^6. \tag{9}$$

Next we will bound the number $|\mathcal{E}_3|$ of 3-element edges in $\mathcal{G}(\beta)$, that is, the number of collinear triples in $B_3(\rho)\cap\mathbb{Z}^3$.

We choose two points P and Q from $B_3(\rho)\cap\mathbb{Z}^3$ in $O(\rho^6)$ ways. There are $O(\rho)$ points in $B_3(\rho)\cap\mathbb{Z}^3$ on the line through P and Q; hence

$$|\mathcal{E}_3| = O(\rho^7)\ . \tag{10}$$

To satisfy the assumptions of Theorem 2.2, we need to have for some $\delta > 0$ that

$$|\mathcal{E}_3| = O(\rho^3 \cdot t^{2-\delta}) \,. \tag{11}$$

Then (11) is satisfied for $0 < \delta < 2\varepsilon/(2+3\varepsilon)$, as can be seen with (9) and (10) for the constant $c' = c^{2/3-\delta/3}$ from the following:

$$\rho^3 \cdot t^{2-\delta} = c' \cdot \beta^{2/3-\delta/3} \cdot \rho^{7-2\delta} = c' \cdot \rho^7 \cdot \left(\beta^{2/3-\delta/3} \cdot \rho^{-2\delta}\right)$$
$$= c' \cdot \rho^7 \cdot (\log n)^{2/3-\delta/3} \cdot n^{2\varepsilon-3\varepsilon\delta-2\delta} = \Omega(\rho^7) \,.$$

Next we will give upper bounds on the number of 2-cycles in our hypergraph $\mathcal{G} = (V, \mathcal{E}_4)$. We will distinguish two types of 2-cycles, namely, $(2,2)$-cycles and $(2,3)$-cycles. In the following we will always assume by Lemma 3.18 that the 2-maximal grids L in $\mathbb{Z}^3$ under consideration have a basis $q_1, q_2 \in \mathbb{Z}^3$, with $\|q_1\|, \|q_2\| = O(\rho)$; hence by Theorem 3.19(i) we have $\|a_L\| = O(\rho^2)$.

6.1. (2, 2)-cycles. Let us consider first the number $s_{2,2}(\mathcal{G})$ of $(2,2)$-cycles in our hypergraph $\mathcal{G} = (V, \mathcal{E}_4)$, that is, the number of pairs of tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$, which have exactly two vertices in common, and both tetrahedra have volume at most β.

We distinguish three cases: (a) both tetrahedra are degenerate, (b) one tetrahedron is degenerate and the other is nondegenerate, and (c) both tetrahedra are nondegenerate. The corresponding numbers of $(2,2)$-cycles are denoted by $s_{2,2}(\mathcal{G}; dd)$, $s_{2,2}(\mathcal{G}; dn)$, $s_{2,2}(\mathcal{G}; nn)$, respectively.

Case (a). Both tetrahedra are degenerate. By Theorem 4.1 there are $O\left(\rho^9 \cdot \log \rho\right)$ degenerate tetrahedra in the set $B_3(\rho) \cap \mathbb{Z}^3$. Fix one of these tetrahedra. The second degenerate tetrahedron is contained in a residue class M' of a 2-maximal grid M in $\mathbb{Z}^3$ and has two vertices in common with the first one, say, $P = (p_1, p_2, p_3) \in M'$ and $Q = (q_1, q_2, q_3) \in M'$. Fix a primitive normal vector $b_M := (b_1, b_2, b_3) \in \mathbb{Z}^3$ with $\|b_M\| = O(\rho^2)$, which belongs to a 2-maximal grid M in $\mathbb{Z}^3$, where $P, Q \in M'$ for some residue class M' of the grid M. If $P, Q \in M'$, then with $y_i := p_i - q_i$ for $i = 1, 2, 3$ it must hold that

$$b_1 \cdot y_1 + b_2 \cdot y_2 + b_3 \cdot y_3 = 0 \,,$$

where $(y_1, y_2, y_3) \neq (0, 0, 0)$. By Theorem 3.19(iii) the set $M' \cap B_3(\rho)$ contains $O(\rho^2/\|b_M\|)$ points; by Lemma 3.18 we can assume that M has a basis $q_1, q_2 \in \mathbb{Z}^3$ with $\|q_1\|, \|q_2\| = O(\rho)$. Having already fixed the vertices P and Q of the second degenerate tetrahedron, two further vertices from M' can be chosen in $O(\binom{\rho^2/\|b_M\|}{2})$ ways. Summing over all possible residue classes M' of 2-maximal grids M in $\mathbb{Z}^3$ with $P, Q \in M'$, we obtain, using Corollary 3.21, the following upper bound on the number of possibilities for the second tetrahedron, where, neglecting constant factors, we assume in the calculations that $y_3 \neq 0$:

$$\left(\sum_{\substack{b=(b_1,b_2,b_3)\in\mathbb{Z}^3;\|b\|=O(\rho^2) \\ b_1y_1+b_2y_2+b_3y_3=0}} \binom{\rho^2/\|b\|}{2} \right)$$
$$= O\left(\rho^4 \cdot \sum_{\substack{(b_1,b_2)\in\mathbb{Z}^2 \\ |b_1|,|b_2|=O(\rho^2)}} \frac{1}{b_1^2 + b_2^2 + (b_1 \cdot y_1/y_3 + b_2 \cdot y_2/y_3)^2} \right)$$

$$= O\left(\rho^4 \cdot \sum_{\substack{(b_1,b_2)\in\mathbb{Z}^2 \\ |b_1|,|b_2|=O(\rho^2)}} \frac{1}{b_1^2+b_2^2}\right) = O\left(\rho^4 \cdot \sum_{d=1}^{O(\rho^4)} \frac{r_2(d)}{d}\right) = O(\rho^4 \cdot \log\rho). \tag{12}$$

Hence the number $s_{2,2}(\mathcal{G}; dd)$ of pairs of degenerate tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$, which have two vertices in common, satisfies

$$s_{2,2}(\mathcal{G}; dd) = O\left(\rho^9 \cdot \log\rho \cdot \rho^4 \cdot \log\rho\right) = O\left(\rho^{13} \cdot (\log\rho)^2\right). \tag{13}$$

Case (b). One tetrahedron is degenerate and the other is nondegenerate and has volume at most β. By Theorem 5.1 the number of nondegenerate tetrahedra with volume at most β in $B_3(\rho) \cap \mathbb{Z}^3$ is $O(\beta \cdot \rho^9)$. Fix such a tetrahedron and fix two of its vertices, say, $P = (p_1, p_2, p_3)$ and $Q = (q_1, q_2, q_3)$, where $y_i := p_i - q_i$ for $i = 1, 2, 3$. As in case (a), for a primitive normal vector $b_M = (b_1, b_2, b_3) \in \mathbb{Z}^3$ with $\|b_M\| = O(\rho^2)$, using Theorem 3.19(i), we must have for the case that $P, Q \in M'$ for some residue class M' of M that

$$b_1 \cdot y_1 + b_2 \cdot y_2 + b_3 \cdot y_3 = 0.$$

Two further vertices of the second degenerate tetrahedron can be chosen from M' in $O(\binom{\rho^2/\|b_M\|}{2})$ ways; hence as in case (a) and using (12), altogether we have $O(\rho^4 \cdot \log\rho)$ possibilities for this. We infer for the number $s_{2,2}(\mathcal{G}; dn)$ of pairs of tetrahedra, where one is nondegenerate with volume at most β and the other is degenerate, the following upper bound:

$$s_{2,2}(\mathcal{G}; dn) = O\left(\beta \cdot \rho^9 \cdot \rho^4 \cdot \log\rho\right) = O\left(\beta \cdot \rho^{13} \cdot \log\rho\right). \tag{14}$$

Case (c). Both tetrahedra have volume at most β, are nondegenerate, and have two vertices in common. To count this number $s_{2,2}(\mathcal{G}; nn)$ of $(2,2)$-cycles, we choose a primitive normal vector $a_L \in \mathbb{Z}^3$ of a 2-maximal grid L in $\mathbb{Z}^3$ with $\|a_L\| = O(\rho^2)$. Then we fix a point $P \in \mathbb{Z}^3 \cap B_3(\rho)$. There is exactly one residue class L' of L with $P \in L'$. For fixed integers $d > 0$ consider a second point $Q \in L'$ with $(\mathrm{dist}(P, Q))^2 = d$, there are at most $r_3(d; a_L)$ of these points. The points P and Q are the two common vertices of the two tetrahedra.

Let $v > 0$ be a real number. By Lemma 5.3 there are $O(v \cdot \rho/(\|a_L\| \cdot d^{1/2}))$ points $R \in L' \cap B_3(\rho)$ such that $\mathrm{area}(P, Q, R) \le v$ and the triangle determined by P, Q, R is nondegenerate. Then the fourth point from $B_3(\rho) \cap \mathbb{Z}^3$ of a tetrahedron with volume at most β can be chosen in $O(\beta \cdot \rho^2/\|a_L\|)$ ways; see (7). If we assume that the third point $R \in L' \cap B_3(\rho)$ satisfies $\mathrm{area}(P, Q, R) > v$, then there are $O(\rho^2/\|a_L\|)$ choices for the point R and the fourth point from $B_3(\rho) \cap \mathbb{Z}^3$ can be chosen in $O(\beta \cdot \rho^2/v)$ ways; thus we have

$$O\left(\frac{\beta \cdot v \cdot \rho^3}{\|a_L\|^2 \cdot d^{1/2}} + \frac{\beta \cdot \rho^4}{v \cdot \|a_L\|}\right) \tag{15}$$

possibilities for the third and fourth vertices of the first tetrahedron. With $v(a_L) := \rho^{1/2} \cdot \|a_L\|^{1/2} \cdot d^{1/4}$ this upper bound (15) becomes

$$O\left(\frac{\beta \cdot \rho^{7/2}}{\|a_L\|^{3/2} \cdot d^{1/4}}\right). \tag{16}$$

Now we consider the second nondegenerate tetrahedron in $B_3(\rho) \cap \mathbb{Z}^3$ with the already fixed vertices $P = (p_1, p_2, p_3)$ and $Q = (q_1, q_2, q_3)$. A third vertex $R \in B_3(\rho) \cap \mathbb{Z}^3$ of the second tetrahedron determines uniquely a 2-maximal grid M in $\mathbb{Z}^3$, such that P, Q, R are contained in a residue class M' of M. These 2-maximal grids M in $\mathbb{Z}^3$ with primitive normal vector $b_M = (b_1, b_2, b_3) \in \mathbb{Z}^3$ with $\|b_M\| = O(\rho^2)$, where $P - Q \in M$ and $P, Q \in M'$, satisfy

$$b_1 \cdot y_1 + b_2 \cdot y_2 + b_3 \cdot y_3 = 0, \tag{17}$$

where $y_i := p_i - q_i$ for $i = 1, 2, 3$. Since the third point $R \in M$ of the nondegenerate tetrahedron satisfies $R \in B_3(\rho) \cap \mathbb{Z}^3$, there are only $O(\rho^3)$ choices for the primitive normal vector $b_M \in \mathbb{Z}^3$. Let $C \subseteq \mathbb{Z}^3$ be the set of all possible choices for b_M.

Having fixed P and Q, the number of possibilities to extend these two points to the second tetrahedron in $B_3(\rho) \cap \mathbb{Z}^3$ is by (16)

$$O\left(\sum_{b \in C} \frac{\beta \cdot \rho^{7/2}}{\|b\|^{3/2} \cdot d^{1/4}}\right) = O\left(\frac{\beta \cdot \rho^{7/2}}{d^{1/4}} \cdot \sum_{b \in C} \frac{1}{\|b\|^{3/2}}\right) = O\left(\frac{\beta \cdot \rho^{17/4}}{d^{1/4}}\right).$$

The last equality can be seen as follows. Since we already fixed the two points P and Q, there are $O(\rho^3)$ possibilities to choose a residue class M' (of a 2-maximal grid M in $\mathbb{Z}^3$) such that $P, Q \in M'$ and where $|M' \cap B_3(\rho)| \geq 3$. Assume that $y_3 \neq 0$. Then by (17) we have that $(b_1, b_2, b_3), (b_1, b_2, b_3') \in C$ implies that $b_3 = b_3'$. We infer using Corollary 3.21 that

$$\sum_{b=(b_1,b_2,b_3) \in C} \frac{1}{\|b\|^{3/2}} = O\left(\sum_{b=(b_1,b_2,b_3) \in C} \frac{1}{(b_1^2 + b_2^2)^{3/4}}\right) = O\left(\sum_{d=1}^{O(\rho^3)} \frac{r_2(d)}{d^{3/4}}\right) = O(\rho^{3/4}).$$

The second equality can be seen from the following.

LEMMA 6.1. *Let $S \subset \mathbb{Z}^2$ be a finite set and let $\alpha > 0$. Then it holds that*

$$\sum_{(s_1,s_2) \in S} \frac{1}{(s_1^2 + s_2^2)^{\alpha}} = O\left(\sum_{d=1}^{|S|} \frac{r_2(d)}{d^{\alpha}}\right).$$

Proof. This is due to the fact that the function $f(x) = 1/x$ is monotone decreasing and that $\sum_{d=1}^{N} r_2(d) = \theta(N)$ by Lemma 3.20; i.e., there are $\Theta(N)$ representations of positive integers $i < N$ as a sum of two squares. ∎

Thus, we infer with Corollaries 3.21 and 3.23 that

$$\begin{aligned}
s_{2,2}(\mathcal{G}; nn) &= O\left(\sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \rho^3 \cdot \sum_{d=1}^{O(\rho^2)} r_3(d; a) \cdot \frac{\beta \cdot \rho^{7/2}}{\|a\|^{3/2} \cdot d^{1/4}} \cdot \frac{\beta \cdot \rho^{17/4}}{d^{1/4}}\right) \\
&= O\left(\beta^2 \cdot \rho^{43/4} \cdot \sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \frac{1}{\|a\|^{3/2}} \cdot \sum_{d=1}^{O(\rho^2)} \frac{r_3(d; a)}{d^{1/2}}\right) \\
&= O\left(\beta^2 \cdot \rho^{47/4} \cdot \sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \frac{1}{\|a\|^{5/2}}\right) = O\left(\beta^2 \cdot \rho^{47/4} \cdot \sum_{d=1}^{O(\rho^4)} \frac{r_3(d)}{d^{5/4}}\right) \\
&= O(\beta^2 \cdot \rho^{51/4}). \qquad (18)
\end{aligned}$$

To satisfy the assumptions of Theorem 2.2, we must have for some suitable constant $\gamma > 0$ that

$$\begin{aligned} s_{2,2}(\mathcal{G}) &= s_{2,2}(\mathcal{G}; dd) + s_{2,2}(\mathcal{G}; dn) + s_{2,2}(\mathcal{G}; nn) \\ &= O\left(\rho^3 \cdot t^{5-\gamma}\right) = O\left(\rho^{13-2\gamma} \cdot \beta^{5/3-\gamma/3}\right), \end{aligned} \tag{19}$$

where $t = c^{1/3} \cdot \beta^{1/3} \cdot \rho^2$ by (9) with $\beta = \rho^3 \cdot \log n/n^3$ and $\rho = n^{1+\varepsilon}$ for some constant $\varepsilon > 0$.

Considering (13), (14), and (18), we have $\rho^{13} \cdot (\log \rho)^2 = O(\beta \cdot \rho^{13} \cdot \log \rho)$, and $\beta^2 \cdot \rho^{51/4} = O(\beta \cdot \rho^{13} \cdot \log \rho)$ for $0 < \varepsilon < 1/11$. Thus it suffices to consider only case (b). In this case (b) we infer with (14) for $0 < \gamma < 2\varepsilon/(2+3\varepsilon)$:

$$\frac{\beta \cdot \rho^{13} \cdot \log \rho}{\rho^{13-2\gamma} \cdot \beta^{5/3-\gamma/3}} = \frac{\rho^{2\gamma} \cdot \log \rho}{\beta^{2/3-\gamma/3}} = O\left(\frac{(\log n)^{1/3+\gamma/3}}{n^{2\varepsilon-3\varepsilon\gamma-2\gamma}}\right) = o(1). \tag{20}$$

Thus by (20) the upper bound (19) holds for $0 < \varepsilon < 1/11$ and $0 < \gamma < 2\varepsilon/(2+3\varepsilon)$.

6.2. (2, 3)-cycles. Let us now consider the number $s_{2,3}(\mathcal{G})$ of $(2,3)$-cycles in our hypergraph $\mathcal{G} = (V, \mathcal{E}_4)$, that is, the number of pairs of tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$, both with volume at most β, having exactly three vertices in common. As in the case of $(2,2)$-cycles, we distinguish three cases: (a) both tetrahedra are degenerate, (b) one tetrahedron is degenerate and the other one is nondegenerate, and (c) both tetrahedra are nondegenerate. The corresponding numbers of $(2,3)$-cycles are denoted by $s_{2,3}(\mathcal{G}; dd)$, $s_{2,3}(\mathcal{G}; nd)$, $s_{2,3}(\mathcal{G}; nn)$, respectively. We consider only those configurations where the three common points *do not* lie on a line, as care is taken of these triples by the 3-element edges of the hypergraph $\mathcal{G}(\beta)$.

Case (a). Both tetrahedra are degenerate and have three vertices in common, which do not lie on a line. Thus the two tetrahedra are contained in a unique residue class L' of a 2-maximal grid L in $\mathbb{Z}^3$ with $L' \cap B_3(\rho) \neq \emptyset$, determined by some primitive normal vector $a_L \in \mathbb{Z}^3$ with $\|a_L\| = O(\rho^2)$ by Theorem 3.19(i). By Theorem 3.19(iii) the set $L' \cap B_3(\rho)$ contains $O(\rho^2/\|a_L\|)$ points. We can choose the five vertices of the two tetrahedra in $O(\binom{\rho^2/\|a_L\|}{5})$ ways. By Theorem 3.19(ii), there are $O(\rho \cdot \|a_L\|)$ residue classes of L intersecting $B_3(\rho)$ in a nonempty set; hence we have for these pairs of tetrahedra the upper bound

$$\begin{aligned} s_{2,3}(\mathcal{G}; dd) &= O\left(\sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \rho \cdot \|a\| \cdot \binom{\rho^2/\|a\|}{5}\right) \\ &= O\left(\rho^{11} \cdot \sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \frac{1}{\|a\|^4}\right) = O\left(\rho^{11} \cdot \sum_{d=1}^{O(\rho^4)} \frac{r_3(d)}{d^2}\right) = O(\rho^{11}), \end{aligned} \tag{21}$$

since by Corollary 3.21 we have $\sum_{d=1}^{n} r_3(d)/d^2 = O(1)$.

Case (b). One tetrahedron is nondegenerate with volume at most β and the other one is degenerate, and they have three vertices in common. By Theorem 5.1 there are $O(\beta \cdot \rho^9)$ nondegenerate tetrahedra in $B_3(\rho) \cap \mathbb{Z}^3$ with volume at most β. Fix one of it and choose three of its vertices, say p_0, p_1, p_2, which are common to both tetrahedra. Since the first tetrahedron is nondegenerate, the points p_0, p_1, p_2 uniquely determine a residue class L' of a maximal grid L in $\mathbb{Z}^3$, and since the second tetrahedron is

degenerate, the fourth point p_3' of the second degenerate tetrahedron is contained in L'. With $|L' \cap B_3(\rho)| = O(\rho^2)$ there are $O(\rho^2)$ choices for the point p_3', and we obtain

$$s_{2,3}(\mathcal{G}; dn) = O(\beta \cdot \rho^9 \cdot \rho^2) = O(\beta \cdot \rho^{11}). \tag{22}$$

Case (c). Both tetrahedra are nondegenerate, each with volume at most β, and they have three vertices in common. Fix a 2-maximal grid L in $\mathbb{Z}^3$ with primitive normal vector $a_L \in \mathbb{Z}^3$, where $\|a_L\| = O(\rho^2)$, and then fix a residue class L' of L with $L' \cap B_3(\rho) \neq \emptyset$. We count the pairs of nondegenerate tetrahedra $S = \{p_0, p_1, p_2, p_3\}$ and $S' = \{p_0, p_1, p_2, p_3'\}$ in $B_3(\rho) \cap \mathbb{Z}^3$ with $p_0, p_1, p_2 \in L'$ and $\mathrm{vol}(S) \leq \beta$ and $\mathrm{vol}(S') \leq \beta$.

The number of nondegenerate triangles in $L' \cap B_3(\rho)$ with area at most v is $O(v \cdot \rho^4/\|a_L\|^3)$ by Corollary 5.4. A fourth point of a tetrahedron from $B_3(\rho) \cap \mathbb{Z}^3$ with volume at most β can be chosen in $O(\beta \cdot \rho^2/\|a_L\|)$ ways.

On the other hand, the number of triangles in $L' \cap B_3(\rho)$ with area at least v is $O(\binom{\rho^2/\|a_L\|}{3})$, and a fourth point of a tetrahedron from $B_3(\rho) \cap \mathbb{Z}^3$ with maximum volume β can be chosen in $O(\beta \cdot \rho^2/v)$ ways.

Altogether, using $v = v(a_L) = \rho^{2/3} \cdot \|a_L\|^{2/3}$, the number of such pairs of tetrahedra is

$$\begin{aligned} &O\left(\frac{v \cdot \rho^4}{\|a_L\|^3} \cdot \left(\frac{\beta \cdot \rho^2}{\|a_L\|}\right)^2 + \frac{\rho^6}{\|a_L\|^3} \cdot \left(\frac{\beta \cdot \rho^2}{v}\right)^2\right) \\ &= O\left(\frac{v \cdot \beta^2 \cdot \rho^8}{\|a_L\|^5} + \frac{\beta^2 \cdot \rho^{10}}{v^2 \cdot \|a_L\|^3}\right) = O\left(\frac{\beta^2 \cdot \rho^{26/3}}{\|a_L\|^{13/3}}\right). \end{aligned}$$

Summing over all grids L in $\mathbb{Z}^3$ with $\|a_L\| = O(\rho^2)$ and over all $O(\rho \cdot \|a_L\|)$ residue classes L' of L with $L' \cap B_3(\rho) \neq \emptyset$, we obtain by Corollary 3.21

$$\begin{aligned} s_{2,3}(\mathcal{G}; nn) &= O\left(\sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \rho \cdot \|a\| \cdot \frac{\beta^2 \cdot \rho^{26/3}}{\|a\|^{13/3}}\right) \\ &= O\left(\beta^2 \cdot \rho^{29/3} \cdot \sum_{\substack{a \in \mathbb{Z}^3 \\ \|a\| = O(\rho^2)}} \frac{1}{\|a\|^{10/3}}\right) = O\left(\beta^2 \cdot \rho^{29/3} \cdot \sum_{d=1}^{O(\rho^4)} \frac{r_3(d)}{d^{5/3}}\right) \\ &= O(\beta^2 \cdot \rho^{29/3}). \end{aligned} \tag{23}$$

To satisfy the assumptions in Theorem 2.2, we must have, for some suitable constant $\gamma > 0$, that

$$s_{2,3}(\mathcal{G}) = s_{2,3}(\mathcal{G}; dd) + s_{2,3}(\mathcal{G}; dn) + s_{2,3}(\mathcal{G}; nn) = O(\rho^3 \cdot t^{4-\gamma}), \tag{24}$$

where $t = c^{1/3} \cdot \beta^{1/3} \cdot \rho^2$ by (9); i.e., using the estimates (21), (22), and (23), we need for some $\gamma > 0$ the following:

$$\rho^{11} + \beta \cdot \rho^{11} + \beta^2 \cdot \rho^{29/3} = O\left(\rho^{11-2\gamma} \cdot \beta^{4/3-\gamma/3}\right),$$

where $\beta := \rho^3 \cdot \log n/n^3$ and $\rho = n^{1+\varepsilon}$ with $\varepsilon > 0$.

Since we have $\rho^{11} = O(\beta \cdot \rho^{11})$ and $\beta^2 \cdot \rho^{29/3} = O(\beta \cdot \rho^{11})$ for $0 < \varepsilon < 4/5$, it suffices to consider only case (b). For this case (b) with (22) and $0 < \gamma < \varepsilon/(2+3\varepsilon)$, we have

$$s_{2,3}(\mathcal{G}; dn) = O\left(\beta \cdot \rho^{11}\right) = O\left(\rho^{11-2\gamma} \cdot \beta^{4/3-\gamma/3}\right),$$

since

$$\frac{\beta \cdot \rho^{11}}{\rho^{11-2\gamma} \cdot \beta^{4/3-\gamma/3}} = \frac{\rho^{2\gamma}}{\beta^{1/3-\gamma/3}} = \frac{1}{(\log n)^{1/3-\gamma/3} \cdot n^{\varepsilon-3\gamma\varepsilon-2\gamma}} = o(1).$$

For $0 < \varepsilon < 1/11$ and $0 < \gamma < \varepsilon/(2+3\varepsilon)$, (24), (11), and (19) are satisfied; thus all assumptions of Theorem 2.2 are fulfilled. Now we apply this theorem to our hypergraph $\mathcal{G}(\beta) = (V, \mathcal{E}_3 \cup \mathcal{E}_4)$ with average degree $t^3 = O(\beta \cdot \rho^6)$ for the 4-element edges, see (9), and we find in time polynomial in $\rho = n^{1+\varepsilon}$ and hence in n an independent set in $\mathcal{G}(\beta)$ of size

$$\begin{aligned}\Omega\left(\frac{|V|}{t} \cdot (\log t)^{1/3}\right) &= \Omega\left(\frac{\rho^3}{\beta^{1/3} \cdot \rho^2} \cdot \left(\log(\beta^{1/3} \cdot \rho^2)\right)^{1/3}\right) \\ &= \Omega\left(\frac{\rho}{(\log n)^{1/3} \cdot n^{\varepsilon}} \cdot (\log n)^{1/3}\right) = \Omega(n).\end{aligned}$$

Thus we have found in polynomial time $\Omega(n)$ points in $B_3(\rho) \cap \mathbb{Z}^3$ such that the volume of every tetrahedron is bigger than $\rho^3 \cdot \log n/n^3$. After rescaling and adapting constant factors, we obtain n points in the unit cube $[0,1]^3$ such that the volume of every tetrahedron is $\Omega(\log n/n^3)$.

REFERENCES

[1] M. Ajtai, J. Komlós, J. Pintz, J. Spencer, and E. Szemerédi, *Extremal uncrowded hypergraphs*, J. Combin. Theory Ser. A, 32 (1982), pp. 321–335.

[2] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley, New York, 1992.

[3] G. Barequet, *A lower bound for Heilbronn's triangle problem in d dimensions*, SIAM J. Discrete Math., 14 (2001), pp. 230–236.

[4] C. Bertram-Kretzberg and H. Lefmann, *The algorithmic aspects of uncrowded hypergraphs*, SIAM J. Comput., 29 (1999), pp. 201–230.

[5] B. Bollobás, *personal communication*, 2001.

[6] C. Bertram–Kretzberg, T. Hofmeister, and H. Lefmann, *An algorithm for Heilbronn's problem*, SIAM J. Comput., 30 (2000), pp. 383-390.

[7] C. Bertram-Kretzberg, H. Lefmann, V. Rödl, and B. Wysocka, *Proper bounded edge-colorings (extended abstract)*, in Proceedings of the 1st International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS), Combinatorics, Complexity and Logic, D. S. Bridges et al., eds., Springer–Verlag, New York, 1996, pp. 121–130.

[8] J. W. S. Cassels, *An Introduction to the Geometry of Numbers*, Vol. 99, Springer-Verlag, New York, 1971.

[9] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer–Verlag, New York, 1993.

[10] R. A. Duke, H. Lefmann, and V. Rödl, *On uncrowded hypergraphs*, Random Structures Algorithms, 6 (1995), pp. 209–212.

[11] A. Fundia, *Derandomizing Chebychev's inequality to find independent sets in uncrowded hypergraphs*, Random Structures Algorithms, 8 (1996), pp. 131–147.

[12] T. Jiang, M. Li, and P. Vitányi, *The average-case area of Heilbronn-type triangles*, Random Structures Algorithms, 20 (2002), pp. 206–219.

[13] M. Koecher, *Lineare Algebra und Analytische Geometrie*, 4th ed., Springer–Verlag, Berlin, 1997.

[14] J. Komlós, J. Pintz, and E. Szemerédi, *On Heilbronn's triangle problem*, J. London Math. Soc., 24 (1981), pp. 385–396.

[15] J. Komlós, J. Pintz, and E. Szemerédi, *A lower bound for Heilbronn's problem*, J. London Math. Soc., 25 (1982), pp. 13–24.
[16] H. Lefmann, *On Heilbronn's problem in higher dimension*, in Proceedings of the 11th ACM–SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2000, pp. 60–64.
[17] K. F. Roth, *On a problem of Heilbronn*, J. London Math. Soc., 26 (1951), pp. 198–204.
[18] K. F. Roth, *On a problem of Heilbronn* II, Proc. London Math. Soc., 25 (1972), pp. 193–212.
[19] K. F. Roth, *On a problem of Heilbronn* III, Proc. London Math. Soc., 25 (1972), pp. 543–549.
[20] K. F. Roth, *Estimation of the area of the smallest triangle obtained by selecting three out of n points in a disc of unit area*, in Proceedings of Symposia in Pure Mathematics, Vol. 24, AMS, Providence, RI, 1973, pp. 251–262.
[21] K. F. Roth, *Developments in Heilbronn's triangle problem*, Adv. Math., 22 (1976), pp. 364–385.
[22] B. L. Rothschild and E. G. Straus, *On triangulations of the convex hull of n points*, Combinatorica, 5 (1985), pp. 167–179.
[23] W. M. Schmidt, *On a problem of Heilbronn*, J. London Math. Soc., 4 (1972), pp. 545–550.

SIAM J. COMPUT.
Vol. 31, No. 6, pp. 1948–1958

THE MINIMIZATION PROBLEM FOR BOOLEAN FORMULAS*

EDITH HEMASPAANDRA† AND GERD WECHSUNG‡

Abstract. More than a quarter of a century ago, the question of the complexity of determining whether a given Boolean formula is minimal motivated Meyer and Stockmeyer to define the polynomial hierarchy. This problem (in the standard formalized version—that of Garey and Johnson) has been known for decades to be coNP-hard and in $\mathrm{NP}^{\mathrm{NP}}$, and yet no one had even been able to establish (many-one) NP-hardness. In this paper, we show that and more: The problem in fact is (many-one) hard for parallel access to NP.

Key words. Boolean formula minimization, parallel access, computational complexity, polynomial hierarchy, reductions

AMS subject classifications. 68Q17, 03D15

PII. S0097539799362639

1. Introduction.

More than a quarter of a century ago, Meyer and Stockmeyer defined the polynomial hierarchy [12]. They were led to this by looking at the language `Minimal`: the set of those Boolean formulas for which there does not exist a shorter equivalent formula. The best upper bound they could find for $\overline{\texttt{Minimal}}$ was an NP machine with access to an NP oracle. Thus, the class $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ was born, and the rest of the polynomial hierarchy was defined analogously.

But what can one say about the complexity of $\overline{\texttt{Minimal}}$? Is it Σ_2^p-complete? Nobody knows. In fact, no better upper bound than Σ_2^p is known, and until the present paper no lower bounds were known. Similarly large gaps between upper and lower bounds occur in other standard definitions of the minimization problem for Boolean formulas. Typically, these problems are trivially in Σ_2^p and have trivial coNP lower bounds, and nothing better was known until the present paper. In this paper we study the complexity of these problems and prove much better lower bounds. Namely, we establish hardness for parallel access to NP in those cases where coNP was the best previously known lower bound, and we establish coNP-hardness for `Minimal`.

There exist different definitions for the minimization problem for Boolean formulas. We will look at the three classic definitions: the original Meyer and Stockmeyer definition [12], the definition from Stockmeyer's seminal polynomial hierarchy paper [15], and the definition from Garey and Johnson [5].

1. Meyer and Stockmeyer [12]. `Minimal`: Given a formula ϕ, is it true that there does not exist a formula equivalent to ϕ that is of smaller size than ϕ?

2. Garey and Johnson [5]. `Minimum Equivalent Expression` (`MEE`, for short): Given a Boolean formula ϕ and a nonnegative integer k, is it true that there exists a

*Received by the editors November 18, 1998; accepted for publication (in revised form) June 3, 2002; published electronically November 6, 2002. A preliminary version of this paper appeared in *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, 1997, IEEE Computer Society Press, pp. 575–584. The research was supported in part by grants NSF-INT-9513368/DAAD-315-PRO-of-ab and NSF-INT-9815095/DAAD-315-PPP-gü-ab.

http://www.siam.org/journals/sicomp/31-6/36263.html

†Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623 (eh@cs.rit.edu). This work was done in part while the author was at Le Moyne College and while visiting Friedrich-Schiller-Universität Jena.

‡Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, 07743 Jena, Germany (wechsung@informatik.uni-jena.de). This work was done in part while the author was visiting Le Moyne College and Rochester Institute of Technology.

Boolean formula of size at most k that is equivalent to ϕ?

3. Stockmeyer [15]. MEE restricted to formulas in DNF (disjunctive normal form) (here and in the rest of the paper, restrictions also apply to the minimum equivalent formula). This problem is called MIN in [15]; we will call it MEE_{DNF}: Given a Boolean formula ϕ in DNF and a nonnegative integer k, is it true that there exists a Boolean formula in DNF of size at most k that is equivalent to ϕ?

In all these cases, $Size(\phi)$ is defined as the number of occurrences of propositional variables in ϕ. Note that $Size(true) = Size(false) = 0$. Of course, Minimal and MEE depend on the underlying language. We will look at Boolean formulas built from propositional variables, the operators $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg\}$, and the constants *true* and *false*. However, we will show that our results are fairly robust in the sense that they hold for any truth-functionally complete subset of $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg, true, false\}$.

Before this paper, the only known results on the complexity of these problems were as follows:

1. Trivial Σ_2^p upper bounds for MEE, MEE_{DNF}, and $\overline{\text{Minimal}}$: Given a formula ϕ and integer k, guess a formula ψ of size at most k (guess a DNF formula for MEE_{DNF}, and guess ψ of size less than $Size(\phi)$ for $\overline{\text{Minimal}}$), and verify that ϕ and ψ are equivalent. Determining equivalence between two formulas is in coNP.

2. Trivial coNP lower bounds for MEE and MEE_{DNF}, by a reduction from "tautology for DNF formulas," since ϕ in DNF is a tautology iff ϕ with all variables set to *true* is true and ϕ has an equivalent formula of size 0.

Garey and Johnson [5] state that MEE is NP-hard, but by this they merely mean NP-*Turing*-hard. In this paper, when we speak of hardness, we always mean *many-one*-hardness.

The results of this paper are as follows:

1. Minimal is coNP-hard.
2. MEE and MEE_{DNF} are hard for parallel access to NP.
3. Minimal restricted to positive formulas is coNP-hard.
4. MEE restricted to positive formulas is NP-hard.

Papadimitriou and Zachos [14] introduced and discussed a complexity class capturing the power of parallel access to NP, denoted by $\text{P}_{||}^{\text{NP}}$: the class of problems that can be solved by a P machine with access to an NP oracle with the restriction that all the queries to the NP oracle are asked in parallel. Clearly, $\text{NP} \cup \text{coNP} \subseteq \text{P}_{||}^{\text{NP}} \subseteq \text{P}^{\text{NP}} \subseteq \text{NP}^{\text{NP}}$. The class has been further studied in [7, 11, 18, 19, 9]. $\text{P}_{||}^{\text{NP}}$ is extremely robust. In particular, $\text{P}_{||}^{\text{NP}}$ is the class of problems that can be solved in polynomial time by $O(\log n)$ sequential Turing queries to NP [7].

The results of this paper should be contrasted with a recent result of Agrawal and Thierauf [1], who showed that the Boolean Isomorphism problem, previously only known to be coNP-hard and in Σ_2^p, just like MEE, cannot be Σ_2^p-complete unless the polynomial hierarchy collapses. (See [2] for related problems.) Their result draws on recent work in learning theory [3]. However, our methods do not seem to apply to their case, nor does their method seem to work in our case.

The rest of the paper is organized as follows. The next section proves that MEE is NP-hard. It also shows that the same construction can be used to prove coNP-hardness for Minimal. Section 3 shows how to use the NP-hardness reduction for MEE in combination with a result of Wagner [18] to obtain the $\text{P}_{||}^{\text{NP}}$ lower bound for MEE. The last section describes how to adapt the proofs for MEE to MEE_{DNF}.

2. MEE is NP-hard. In this section, we prove that MEE is NP-hard by a reduction from Vertex Cover.

DEFINITION 2.1. `Vertex Cover`*: Given a graph $G = \langle V, E\rangle$ and a positive integer k, does G have a vertex cover of size $\leq k$; i.e., does there exist a set $W \subseteq V$ such that $||W|| \leq k$ and, for all edges $\{v, w\} \in E$, $\{v, w\} \cap W \neq \emptyset$?*

In the next section, we will use the following properties of the reduction from `Vertex Cover` to `MEE` to show that `MEE` is hard for parallel access to NP.

THEOREM 2.2. *There exists a polynomial-time computable function f such that for all graphs G, $f(G)$ is a Boolean formula, and for all positive integers k, G has a vertex cover of size $\leq k$ iff $f(G)$ has an equivalent formula of size $\leq k + m(1 + 2m)$, where m is the number of edges in G.*

Proof. Let $G = \langle V, E\rangle$ be a graph, and let k be a positive integer. Let $||V|| = n$ and $||E|| = m$. Without loss of generality, we assume that $V = \{1, \ldots, n\}$, that G has no isolated vertices, and that $k < n$.

We will use the following $n+2m^2$ propositional variables in the formula $\phi = f(G)$:

- a variable p_i for every vertex $i \in V$, and
- a set Q_e of $2m$ propositional variables for every edge $e \in E$.

Define $f(G) = \phi$ as follows:

$$\phi = \bigvee_{\{i,j\}\in E} \left(p_i \wedge p_j \wedge \bigwedge_{q\in Q_{\{i,j\}}} q \right).$$

The formula ϕ contains all the information about graph G. We will show that G has a vertex cover of size $\leq k$ iff ϕ has an equivalent formula of size $\leq k + m(1 + 2m)$. Since f is clearly computable in polynomial time, this will prove Theorem 2.2.

For the left-to-right direction, suppose that $\{\ell_1, \ldots, \ell_k\}$ is a vertex cover of G. Then for every edge $\{i, j\} \in E$, $\{i, j\} \cap \{\ell_1, \ldots, \ell_k\} \neq \emptyset$. Thus, we can partition the set of edges E into k disjoint sets $E_1, E_2, \ldots, E_k$ such that, for all $1 \leq i \leq k$, $e \in E_i$ implies that $\ell_i \in e$. (In general, it will not be the case that every edge incident with ℓ_i will belong to E_i, because the E_i's form a partition.)

Consider the following formula:

$$\phi' = \bigvee_{i=1}^{k} \left(p_{\ell_i} \wedge \bigvee_{\{\ell_i,j\}\in E_i} \left(p_j \wedge \bigwedge_{q\in Q_{\{\ell_i,j\}}} q \right) \right).$$

It is easy to see that ϕ' and ϕ are equivalent. In addition, $\mathit{Size}(\phi') = k + \Sigma_{i=1}^{k}(||E_i|| \cdot (1 + 2m)) = k + (\Sigma_{i=1}^{k}||E_i||) \cdot (1 + 2m)$. Since the E_i's form a partition of E, $\mathit{Size}(\phi') = k + m(1 + 2m)$.

This completes the proof of the left-to-right direction. It remains to show that if ϕ has an equivalent formula of size $\leq k + m(1 + 2m)$, then G has a vertex cover of size $\leq k$. We will first prove the following lemma, which will make the formula easier to handle.

In the remainder of the proof, let ϕ be the formula defined above. For each $e \in E$, let q_e be a new propositional variable. Define $\widehat{\phi}$ as follows:

$$\widehat{\phi} = \bigvee_{\{i,j\}\in E} (p_i \wedge p_j \wedge q_{\{i,j\}}).$$

LEMMA 2.3. *If $\langle \phi, k + m(1 + 2m)\rangle \in$ `MEE`, then $\widehat{\phi}$ has an equivalent formula $\widehat{\psi}$ such that $\mathit{Size}(\widehat{\psi}) \leq k + 2m$ and every q_e occurs exactly once in $\widehat{\psi}$.*

Proof. Let ψ be a formula such that $Size(\psi) \leq k + m(1+2m)$ and ψ is equivalent to ϕ. First note that all variables in ϕ occur at least once in ψ. Thus, there are exactly $n + 2m^2$ different variables in ψ.

Also note that for every edge $e \in E$, there exists a variable in Q_e that occurs exactly once in ψ. To see this, suppose for a contradiction that there exists some set Q_e such that all the variables of Q_e occur at least twice in ψ. Then $Size(\psi) \geq n + m \cdot 2m + 2m > k + m(1+2m)$.

For every $e \in E$, let $\widehat{q_e} \in Q_e$ occur exactly once in ψ. Let $\widehat{\psi}$ be the formula that results from ψ when we set every variable in $\bigcup_{e \in E} Q_e \backslash \{\widehat{q_e}\}$ to *true* and replace $\widehat{q_e}$ by q_e. $\widehat{\psi}$ is equivalent to $\widehat{\phi}$, and $Size(\widehat{\psi}) \leq Size(\psi) - m(2m-1) \leq k + m(1+2m) - m(2m-1) = k + 2m$, and every q_e occurs exactly once in $\widehat{\psi}$. □

For the remainder of the proof of Theorem 2.2, let $\widehat{\phi}$ be as defined before Lemma 2.3, let $\widehat{\psi}$ be equivalent to $\widehat{\phi}$, and for all $e \in E$, let q_e occur exactly once in $\widehat{\psi}$. To finish the proof of Theorem 2.2, we will show that if G does *not* have a vertex cover of size k, then $Size(\widehat{\psi}) > k + 2m$.

We will view $\widehat{\psi}$ as a binary tree. The nodes of the tree are labeled by operators, propositional variables, *true*, and *false*. We will freely identify node X in the tree with the formula represented by the subtree rooted at X. We adopt the convention that every node is an ancestor of itself.

LEMMA 2.4. *Let $e, f \in E$, let $i \in e$, $i \notin f$, and in the tree representing $\widehat{\psi}$,*

- *let X be the least common ancestor of the node labeled q_e and the node labeled q_f,*
- *let X_e be the child of X that is an ancestor of the node labeled q_e, and*
- *let X_f be the child of X that is an ancestor of the node labeled q_f.*

Then p_i occurs in X_e.

Proof. Since q_e and q_f occur only once in $\widehat{\psi}$, and since X and X_e (X and X_f) are on the path from the root to q_e (q_f), we have the following fact, which will be used repeatedly in the proof of Lemma 2.4.

FACT 2.5. *If the value of $q_e(q_f)$ is changed and this change causes the value of $\widehat{\psi}$ to change, then the values of X and X_e (X and X_f) change as well.*

Let $e = \{i, j\}$, and let $f = \{r, s\}$. We will allow freedom only in the assignments to propositional variables q_e, q_f, and p_i, and we will fix the assignments to all other variables in such a way that the following formula α is satisfied:

$$\alpha = p_j \wedge p_r \wedge p_s \wedge \bigwedge_{p \notin \{q_e, q_f, p_i, p_j, p_r, p_s\}} \neg p.$$

To prove Lemma 2.4, it is sufficient to consider only two cases, namely, that X is labeled $\wedge$ and that X is labeled $\leftrightarrow$. This is true since the connectives $\vee$ and $\rightarrow$ may be replaced by $\wedge$ and $\neg$ without an increase in the size of the formula (since $(p \vee q) \leftrightarrow \neg(\neg p \wedge \neg q)$ and $(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$).

Case 1: X is labeled $\wedge$. Consider what happens to X_e, X_f, X, and $\widehat{\psi}$ for the assignment that satisfies $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha$. Note that $\neg\widehat{\psi}$ holds, and note that we have four different possibilities for the values of X_e, X_f, and X, since X iff $X_e \wedge X_f$.

Case 1.1: $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e \wedge \neg X_f \wedge \neg X \wedge \neg\widehat{\psi}$. Applying Fact 2.5, we get

$$q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow X_e \wedge \neg X_f \wedge X \wedge \widehat{\psi},$$

which contradicts the fact that X iff $X_e \wedge X_f$.

Case 1.2: $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow X_e \wedge \neg X_f \wedge \neg X \wedge \neg\widehat{\psi}$. Applying Fact 2.5, we get

$$q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e \wedge \neg X_f \wedge X \wedge \widehat{\psi},$$

which contradicts the fact that X iff $X_e \wedge X_f$.

Case 1.3: $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e \wedge X_f \wedge \neg X \wedge \neg\widehat{\psi}$. Applying Fact 2.5, we get

$$\neg q_e \wedge q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e \wedge \neg X_f \wedge X \wedge \widehat{\psi},$$

which contradicts the fact that X iff $X_e \wedge X_f$.

Case 1.4: The only remaining case is $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow X_e \wedge X_f \wedge X \wedge \neg\widehat{\psi}$. Applying Fact 2.5, we get

$$q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e \wedge X_f \wedge \neg X \wedge \widehat{\psi}.$$

For a contradiction, assume that p_i does not occur in X_e. Then we get

$$q_e \wedge \neg q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e \wedge \neg X \wedge \neg\widehat{\psi}.$$

The first term on the right-hand side is justified by the assumption that p_i does not occur in X_e, and $\neg X$ follows from $\neg X_e$. Making use of Fact 2.5, we get

$$q_e \wedge q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e \wedge X \wedge \widehat{\psi},$$

which contradicts the fact that X iff $X_e \wedge X_f$.

Case 2: X is labeled $\leftrightarrow$. Again, we will consider what happens for the assignment that satisfies $\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha$. We will write

$$\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow X_e^* \wedge X_f^* \wedge X^* \wedge \neg\widehat{\psi}, \tag{1}$$

where $X^* \in \{X, \neg X\}$, $X_e^* \in \{X_e, \neg X_e\}$, and $X_f^* \in \{X_f, \neg X_f\}$. Since X is labeled $\leftrightarrow$, we know that

$$X^* \text{ is true iff } X_e^* \leftrightarrow X_f^* \text{ is true.} \tag{2}$$

Applying Fact 2.5, we get

$$q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e^* \wedge X_f^* \wedge \neg X^* \wedge \widehat{\psi}. \tag{3}$$

For a contradiction, assume that p_i does not occur in X_e. Then we get from (3) that

$$q_e \wedge \neg q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e^* \wedge X_f^+ \wedge X^+ \wedge \neg\widehat{\psi}. \tag{4}$$

The first term on the right-hand side is justified by (3) and the assumption that p_i does not occur in X_e. Since we do not know if p_i influences X_f or X, we cautiously write X_f^+ and X^+, where $X^+ \in \{X, \neg X\}$ and $X_f^+ \in \{X_f, \neg X_f\}$.

Making use of Fact 2.5, we get

$$q_e \wedge q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e^* \wedge \neg X_f^+ \wedge \neg X^+ \wedge \widehat{\psi}. \tag{5}$$

Case 2.1: $X^+ = X^*$ and $X_f^+ = X_f^*$. From (4) we get X^* is true and X_f^* is true, but X_e^* is false. This contradicts (2).

Case 2.2: $X^+ = \neg X^*$ and $X_f^+ = \neg X_f^*$. From (5) we get X^* is true and X_f^* is true, but X_e^* is false. This contradicts (2).

The remaining two cases need more work.

Case 2.3: $X^+ = X^*$ and $X_f^+ = \neg X_f^*$. We summarize the statements valid in this case.

$$\neg q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow X_e^* \wedge X_f^* \wedge X^* \wedge \neg\widehat{\psi} \tag{1}$$

$$q_e \wedge \neg q_f \wedge p_i \wedge \alpha \rightarrow \neg X_e^* \wedge X_f^* \wedge \neg X^* \wedge \widehat{\psi} \tag{3}$$

$$q_e \wedge \neg q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e^* \wedge \neg X_f^* \wedge X^* \wedge \neg\widehat{\psi} \tag{4}$$

$$q_e \wedge q_f \wedge \neg p_i \wedge \alpha \rightarrow \neg X_e^* \wedge X_f^* \wedge \neg X^* \wedge \widehat{\psi} \tag{5}$$

Comparing (3) and (4), we see that both X_f and X depend on p_i. Since q_e does not have any influence on the dependency of X_f on p_i, we get the following from (1) by switching p_i to $\neg p_i$.

$$\neg q_e \wedge \neg q_f \wedge \neg p_i \wedge \alpha \rightarrow X_e^* \wedge \neg X_f^* \wedge \neg X^* \wedge \neg\widehat{\psi}. \tag{6}$$

The reason for X_e^* occurring on the right-hand side is that p_i does not occur in X_e. $\neg X^*$ follows from (2), which always has to be satisfied. It is clear that $\neg\widehat{\psi}$ occurs because the assignment that satisfies the left-hand side of (6) falsifies $\widehat{\psi}$.

If $\alpha = \mathit{true}$, the value of $\widehat{\psi}$ depends only on the values of X and p_i. However, because of (5), X^* and p_i are false and $\widehat{\psi}$ is true, but because of (6), X^* and p_i are false and $\widehat{\psi}$ is false. This is a contradiction.

Case 2.4: $X^+ = \neg X^*$ and $X_f^+ = X_f^*$. This case is treated in the same way as Case 2.3.

This completes the proof of Lemma 2.4 □

LEMMA 2.6. *Let X be an internal node in $\widehat{\psi}$, and let Y be a child of X. If*

$$\bigcap\{\{i,j\} \mid q_{\{i,j\}} \text{ occurs in } X\} = \emptyset,$$

then for all $\{i,j\} \in E$, if $q_{\{i,j\}}$ occurs in Y, then so do p_i and p_j.

Proof. Suppose $q_{\{i,j\}}$ occurs in Y. We will show that p_i occurs in Y. Because

$$\bigcap\{\{i,j\} \mid q_{\{i,j\}} \text{ occurs in } X\} = \emptyset,$$

there exists an edge $f \in E$ such that q_f occurs in X, and $i \notin f$.

Let Z be the least common ancestor of the nodes labeled $q_{\{i,j\}}$ and q_f, and let $Z_{\{i,j\}}$ be the child of Z that is an ancestor of the node labeled $q_{\{i,j\}}$. By Lemma 2.4, p_i occurs in $Z_{\{i,j\}}$. It remains to show that Y is an ancestor of $Z_{\{i,j\}}$. This is simple: Since there is only one node labeled $q_{\{i,j\}}$, X, Y, Z, and $Z_{\{i,j\}}$ all lie on the unique path from the root to the node labeled $q_{\{i,j\}}$. In addition, X is an ancestor of the node labeled q_f and is therefore an ancestor of Z. It follows that Y is an ancestor of $Z_{\{i,j\}}$. □

To finish the proof of Theorem 2.2, suppose that G does not have a vertex cover of size k. We need to show that $\mathit{Size}(\widehat{\psi}) > k + 2m$.

Let $Y_1, Y_2, \ldots, Y_r$ be all nodes in $\widehat{\psi}$ such that, for all $1 \leq \ell \leq r$,

- $\bigcap\{\{i,j\} \mid q_{\{i,j\}} \text{ occurs in } Y_\ell\} \neq \emptyset$, and
- if X is a proper ancestor of Y_ℓ, then $\bigcap\{\{i,j\} \mid q_{\{i,j\}} \text{ occurs in } X\} = \emptyset$.

The following hold:

- For every $e \in E$, the node labeled q_e has exactly one Y_ℓ as ancestor,
- $r > k$ (because the Y_ℓ's induce a vertex cover of size r), and
- if $q_{\{i,j\}}$ occurs in Y_ℓ, then so do p_i and p_j by Lemma 2.6.

It follows that

$$\begin{aligned} \mathit{Size}(\widehat{\psi}) &\geq m + \sum_{1 \leq \ell \leq r} \left|\left| \bigcup \{\{p_i, p_j\} \mid q_{\{i,j\}} \text{ occurs in } Y_\ell\} \right|\right| \\ &\geq m + \sum_{1 \leq \ell \leq r} (||\{q_e \mid q_e \text{ occurs in } Y_\ell\}|| + 1) \\ &= m + m + r \\ &> 2m + k. \end{aligned}$$

This completes the proof of Theorem 2.2. □

We conclude this section by using Theorem 2.2 to show that `Minimal` is coNP-hard.

THEOREM 2.7. `Minimal` *is* coNP*-hard.*

Proof. We will reduce the complement of `Vertex Cover` to `Minimal`. Let $G = \langle V, E\rangle$ be a graph, and let k be a positive integer. Let $||V|| = n$, $k < n$, let W be a set of $n - k - 1$ new vertices, and define $G' = \langle V \cup W, E \cup (V \times W)\rangle$.

The vertex covers for G' are exactly those sets of vertices that either contain V or are of the form $W \cup \widehat{V}$, where $\widehat{V}$ is a vertex cover for G. It follows that G does *not* have a vertex cover of size $\leq k$ iff all vertex covers for G' have size at least n, and the latter is true iff V is a vertex cover of minimum size for G'.

Applying Theorem 2.2, this holds iff the minimum size of a formula equivalent to $f(G')$ is $n + m(1 + 2m)$, where m is the number of edges in G'.

Writing $\{1, \ldots, n\}$ for V and $\{n+1, \ldots, 2n-k-1\}$ for W, $f(G')$ is the following formula:

$$\bigvee_{\{i,j\} \in E \cup (V \times W)} \left(p_i \wedge p_j \wedge \bigwedge_{q \in Q_{\{i,j\}}} q \right),$$

where for each edge e, Q_e consists of $2m$ new variables. Since V is clearly a vertex cover for G', the following formula ϕ' is equivalent to $f(G')$:

$$\phi' = \bigvee_{i=1}^{n} \left(p_i \wedge \bigvee_{\{i,j\} \in E, i<j} \left(p_j \wedge \bigwedge_{q \in Q_{\{i,j\}}} q \right) \right).$$

Note that $\mathit{Size}(\phi') = n + m(1 + 2m)$. It follows that $G = \langle V, E\rangle$ does *not* have a vertex cover of size $\leq k$ iff $\phi' \in$ `Minimal`. □

Since the reductions to `MEE` and `Minimal` construct formulas that use only $\wedge$ and $\vee$, and since the minimum size formulas (i.e., ϕ') in both constructions in this section also use only $\wedge$ and $\vee$, it is immediate that the results go through if we restrict the problems to positive formulas.

COROLLARY 2.8. `MEE` *restricted to positive formulas is* NP*-hard, and* `Minimal` *restricted to positive formulas is* coNP*-hard.*

Also note that formulas over $\{\vee, \wedge\}$ can in polynomial time be transformed into formulas over any truth-functionally complete subset of $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg, \mathit{true}, \mathit{false}\}$ without an increase in size. This immediately gives the following corollary.

COROLLARY 2.9. *For any truth-functionally complete subset of* $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg, \mathit{true}, \mathit{false}\}$, `MEE` *restricted to formulas over this subset is* NP*-hard, and* `Minimal` *restricted to formulas over this subset is* coNP*-hard.*

3. `MEE` is $\mathbf{P}_{||}^{\mathbf{NP}}$-hard.

In this section, we will prove the following theorem.

THEOREM 3.1. `MEE` *is* $\mathrm{P}_{||}^{\mathrm{NP}}$*-hard.*

Proof. We will use the following theorem from Wagner [18].

THEOREM 3.2 (Wagner [18]). *Let D be an* NP*-hard set, and let A be an arbitrary set. If there exists a polynomial-time computable function h such that*[1]

$$||\{i \mid x_i \in D\}|| \text{ is odd } \Leftrightarrow h(x_1, \ldots, x_{2k}) \in A$$

for all $k \geq 1$ and $x_1, x_2, \ldots, x_{2k} \in \Sigma^*$ with $\chi_D(x_1) \geq \chi_D(x_2) \geq \cdots \geq \chi_D(x_{2k})$, then A is $\mathrm{P}_{||}^{\mathrm{NP}}$*-hard.*[2] *($\chi_D$ is the characteristic function of D.)*

Thus, in our case it suffices to construct a polynomial-time computable function h such that

$$||\{i \mid \phi_i \in \mathrm{SAT}\}|| \text{ is odd } \Leftrightarrow h(\phi_1, \ldots, \phi_{2k}) \in \texttt{MEE}$$

for all $k \geq 1$ and conjunctive normal form (CNF) formulas $\phi_1, \phi_2, \ldots, \phi_{2k}$ with $\phi_{i+1} \in \mathrm{SAT} \Rightarrow \phi_i \in \mathrm{SAT}$ for all $i < 2k$.

Before we can attempt to construct h, we have to look carefully at the properties of the NP-hardness reduction to `MEE`. We introduce the following shorthand for the minimum size of a formula equivalent to ϕ.

DEFINITION 3.3. *Minsize*$(\phi) = \mathit{min}\{\mathit{Size}(\psi) \mid \psi \text{ equivalent to } \phi\}$.

LEMMA 3.4. *There exists a reduction g from* SAT *to* `MEE` *such that, for all CNF formulas ϕ, $g(\phi) = \langle \psi, k \rangle$, where ψ is a Boolean formula, k is a positive integer, and*

- $\neg\psi \in \mathrm{SAT}$,
- $\phi \in \mathrm{SAT} \Rightarrow$ *Minsize*$(\psi) = k$, *and*
- $\phi \notin \mathrm{SAT} \Rightarrow$ *Minsize*$(\psi) = k + 1$.

Proof. We may assume (see Papadimitriou and Yannakakis [13]) that there exists a polynomial-time computable function g' such that, for all CNF formulas ϕ, $g'(\phi) = \langle G, k \rangle$, where G is a graph, k is a positive integer, and

- if $\phi \in \mathrm{SAT}$, then the size of a minimum vertex cover of G is k, and
- if $\phi \notin \mathrm{SAT}$, then the size of a minimum vertex cover of G is $k + 1$.

Let f be the function from Theorem 2.2. Then, for all graphs G, $f(G)$ is a Boolean formula, and for all k, G has a vertex cover of size $\leq k$ iff $f(G)$ has an equivalent formula of size $\leq k + m(1 + 2m)$, where m is the number of edges in G.

It is immediate that, for all graphs G and for all k, the size of a minimum vertex cover of G is k iff *Minsize*$(f(G)) = k + m(1 + 2m)$, where m is the number of edges in G. In addition, since $k \geq 1$, *Minsize*$(f(G)) \geq 1$, and thus $f(G)$ is not equivalent to *true* or *false*. Hence, both $f(G), \neg f(G) \in \mathrm{SAT}$. Now define $g(\phi)$ as follows:

$$g(\phi) = \langle f(G), k + m(1 + 2m) \rangle, \text{ where } \langle G, k \rangle = g'(\phi). \qquad \square$$

In our proof of Theorem 3.1, we will use the following well-known fact.

FACT 3.5. *If ϕ and ψ are disjoint (i.e., the sets of propositional variables that occur in ϕ and ψ are disjoint), then*

---

[1] Since h is a variable-arity function, we mean that h's running time is polynomial in $2k + |x_1| + |x_2| + \cdots + |x_{2k}|$; see [8] for a discussion on why we have to be careful here.

[2] In fact, Wagner actually proves hardness for $\mathrm{P}_{\mathrm{bf}}^{\mathrm{NP}}$, but that class is now known to be equivalent to $\mathrm{P}_{||}^{\mathrm{NP}}$.

1. $Minsize(\phi \wedge \psi) = Minsize(\phi) + Minsize(\psi)$ *if* $\phi \in \mathrm{SAT}$ *and* $\psi \in \mathrm{SAT}$,
2. $Minsize(\phi \wedge \psi) = 0$ *if* $\phi \notin \mathrm{SAT}$ *or* $\psi \notin \mathrm{SAT}$,
3. $Minsize(\phi \vee \psi) = Minsize(\phi) + Minsize(\psi)$ *if* $\neg\phi \in \mathrm{SAT}$ *and* $\neg\psi \in \mathrm{SAT}$,
4. $Minsize(\phi \vee \psi) = 0$ *if* $\neg\phi \notin \mathrm{SAT}$ *or* $\neg\psi \notin \mathrm{SAT}$.

Now we are ready to prove Theorem 3.1. Let $k \geq 1$, and let $\phi_1, \phi_2, \ldots, \phi_{2k}$ be CNF formulas such that $\phi_{i+1} \in \mathrm{SAT} \Rightarrow \phi_i \in \mathrm{SAT}$ for all $i < 2k$. We have to construct a polynomial-time computable function h such that $||\{i \mid \phi_i \in \mathrm{SAT}\}||$ is odd iff $h(\phi_1, \ldots, \phi_{2k}) \in \mathtt{MEE}$. Note that, for all m, $||\{i \mid \phi_i \in \mathrm{SAT}\}|| = m$ iff $\phi_1, \phi_2, \ldots, \phi_m \in \mathrm{SAT}$ and $\phi_{m+1}, \ldots, \phi_{2k} \notin \mathrm{SAT}$. Therefore,

- if $||\{i \mid \phi_i \in \mathrm{SAT}\}||$ is even, then $\forall i$ ($\phi_{2i-1} \in \mathrm{SAT}$ iff $\phi_{2i} \in \mathrm{SAT}$), and
- if $||\{i \mid \phi_i \in \mathrm{SAT}\}||$ is odd, then $\exists i$ [($\phi_{2i-1} \in \mathrm{SAT}$ and $\phi_{2i} \notin \mathrm{SAT}$) and $\forall j \neq i$ ($\phi_{2j-1} \in \mathrm{SAT}$ iff $\phi_{2j} \in \mathrm{SAT}$)].

Let g be the reduction from Lemma 3.4. For all i, let $g(\phi_{2i-1}) = \langle \psi_{2i-1}, k_i \rangle$. Then $k_i \geq 1$ and

- $\neg\psi_{2i-1} \in \mathrm{SAT}$,
- $Minsize(\psi_{2i-1}) = k_i$ if $\phi_{2i-1} \in \mathrm{SAT}$, and
- $Minsize(\psi_{2i-1}) = k_i + 1$ if $\phi_{2i-1} \notin \mathrm{SAT}$.

Define $m = \Sigma_{i=1}^{k} Size(\phi_{2i}) + 1$.

In order to apply Fact 3.5, we will need to make all formulas disjoint. We will also need a number of disjoint copies of the same formula. Without loss of generality, suppose that, for all i, every propositional variable in ϕ_{2i} and ψ_{2i-1} is of the form p_ℓ for some ℓ. The propositional variables in our new formulas will be of the form $q_{\ell,i,j}$, $r_{\ell,i,j}$, and $w_{i,j}$.

- For all i, we need m disjoint copies of ψ_{2i-1}. For $1 \leq j \leq m$, let $\psi_{2i-1,j}$ be a copy of ψ_{2i-1} with every occurrence of a propositional variable p_ℓ replaced by $q_{\ell,i,j}$.
- For all i, let ϕ''_{2i} be a copy of ϕ_{2i} with every occurrence of a propositional variable p_ℓ replaced by $r_{\ell,i,j}$, and let

$$\phi'_{2i} = \phi''_{2i} \wedge w_{i,1} \wedge w_{i,2} \wedge \cdots \wedge w_{i,m}.$$

Since all these formulas are disjoint, and $\neg\psi_{2i-1} \in \mathrm{SAT}$, we obtain the following by applying Fact 3.5:

- $Minsize(\bigvee_{j=1}^{m} \psi_{2i-1,j}) = mk_i$ if $\phi_{2i-1} \in \mathrm{SAT}$,
- $Minsize(\bigvee_{j=1}^{m} \psi_{2i-1,j}) = mk_i + m$ if $\phi_{2i-1} \notin \mathrm{SAT}$,
- $Minsize(\phi'_{2i}) = 0$ if $\phi_{2i} \notin \mathrm{SAT}$, and
- $m \leq Minsize(\phi'_{2i}) \leq m + Size(\phi_{2i})$ if $\phi_{2i} \in \mathrm{SAT}$.

For $1 \leq i \leq k$, define

$$\xi_i = \bigvee_{j=1}^{m} \psi_{2i-1,j} \vee \phi'_{2i}.$$

Note that $\neg \bigvee_{j=1}^{m} \psi_{2i-1,j} \in \mathrm{SAT}$ and $\neg\phi'_{2i} \in \mathrm{SAT}$. This implies by Fact 3.5 that

$$Minsize(\xi_i) = Minsize\left(\bigvee_{j=1}^{m} \psi_{2i-1,j}\right) + Minsize(\phi'_{2i}).$$

It follows that, for all i,

- $Minsize(\xi_i) = mk_i$ if $\phi_{2i-1} \in \mathrm{SAT}$ and $\phi_{2i} \notin \mathrm{SAT}$, and

• $mk_i + m \leq \mathit{Minsize}(\xi_i) \leq mk_i + m + \mathit{Size}(\phi_{2i})$ if ($\phi_{2i-1} \in$ SAT iff $\phi_{2i} \in$ SAT).

Now we define the reduction

$$h(\phi_1, \phi_2, \ldots, \phi_{2k}) = \left\langle \bigvee_{i=1}^{k} \xi_i, m\Sigma_{i=1}^{k} k_i + km - 1 \right\rangle.$$

Clearly, h is computable in polynomial time. Note that, for all i, $\neg\xi_i \in$ SAT. Thus, $\mathit{Minsize}(\bigvee_{i=1}^{k} \xi_i) = \Sigma_{i=1}^{k} \mathit{Minsize}(\xi_i)$.

We need to show that $||\{i \mid \phi_i \in \text{SAT}\}||$ is odd iff $h(\phi_1, \ldots, \phi_{2k}) \in$ MEE.

($\Leftarrow$) Suppose that $h(\phi_1, \phi_2, \ldots, \phi_{2k}) \in$ MEE, and suppose for a contradiction that $||\{i \mid \phi_i \in \text{SAT}\}||$ is even. Then $\forall i$ ($\phi_{2i-1} \in$ SAT iff $\phi_{2i} \in$ SAT). It follows that, for all i, $\mathit{Minsize}(\xi_i) \geq mk_i + m$. And thus, $\mathit{Minsize}(\bigvee_{i=1}^{k} \xi_i) \geq m\Sigma_{i=1}^{k} k_i + km$. But that contradicts the assumption that $h(\phi_1, \phi_2, \ldots, \phi_{2k}) \in$ MEE.

($\Rightarrow$) Suppose $||\{i \mid \phi_i \in \text{SAT}\}||$ is odd. Let i be such that ($\phi_{2i-1} \in$ SAT and $\phi_{2i} \notin$ SAT) and $\forall j \neq i$ ($\phi_{2j-1} \in$ SAT iff $\phi_{2j} \in$ SAT). It follows that $\mathit{Minsize}(\xi_i) = mk_i$ and that, for all $j \neq i$, $\mathit{Minsize}(\xi_j) \leq mk_j + m + \mathit{Size}(\phi_{2j})$. Thus,

$$\begin{aligned}
&\mathit{Minsize}\left(\bigvee_{j=1}^{k} \xi_j\right) \\
&\leq mk_i + \Sigma_{j \neq i}(mk_j + m + \mathit{Size}(\phi_{2j})) \\
&\leq m\Sigma_{j=1}^{k} k_j + m(k-1) + \Sigma_{j=1}^{k} \mathit{Size}(\phi_{2j}) \\
&\leq m\Sigma_{j=1}^{k} k_j + m(k-1) + m - 1 \\
&= m\Sigma_{j=1}^{k} k_j + km - 1.
\end{aligned}$$

It follows that $h(\phi_1, \phi_2, \ldots, \phi_{2k}) \in$ MEE. □

Careful inspection of the construction shows that if $\mathit{Minsize}(\bigvee_{i=1}^{k} \xi_i) \leq m\Sigma_{i=1}^{k} k_i + km - 1$, then this is witnessed by a formula over $\{\wedge, \vee, \neg\}$. Such a formula can in polynomial time be transformed into an equivalent formula over any truth-functionally complete subset of $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg, \mathit{true}, \mathit{false}\}$ without an increase in size.

COROLLARY 3.6. *For any truth-functionally complete subset of* $\{\leftrightarrow, \rightarrow, \wedge, \vee, \neg, \mathit{true}, \mathit{false}\}$, MEE *restricted to formulas over this subset is* $\mathrm{P}_{||}^{\mathrm{NP}}$*-hard.*

4. The Complexity of $\mathtt{MEE}_{\mathtt{DNF}}$. In this section, we investigate the complexity of MEE restricted to formulas in DNF. Note that we cannot use the reduction from Theorem 2.2 to show that $\mathtt{MEE}_{\mathtt{DNF}}$ is NP-hard, since the equivalent formula of minimum size (ϕ') is definitely not in DNF.

In 1965, predating NP and NP-completeness, Gimpel [4] reduced the general set covering problem to the prime implicant covering problem. As pointed out in Garey and Johnson [5], this implicitly gives a reduction from the NP-complete problem `Minimum Cover` to `Minimum Disjunctive Normal Form`, a problem that is closely related to $\mathtt{MEE}_{\mathtt{DNF}}$. A very careful inspection of the reduction implicit in [4] shows that this reduction establishes the NP-hardness of $\mathtt{MEE}_{\mathtt{DNF}}$. (The NP-hardness of $\mathtt{MEE}_{\mathtt{DNF}}$ also follows from Hammer and Kogan's result that MEE restricted to Horn CNFs is NP-complete [6].)

Just as in the case for MEE, we can combine the properties of the NP-hardness reduction with Wagner's Theorem 3.2 to obtain a $\mathrm{P}_{||}^{\mathrm{NP}}$ lower bound.

THEOREM 4.1. $\mathtt{MEE}_{\mathtt{DNF}}$ *is* $\mathrm{P}_{||}^{\mathrm{NP}}$*-hard.*

After our results appeared in [10], Umans showed that $\mathtt{MEE}_{\mathtt{DNF}}$ is complete for Σ_2^p [16], and that $\mathtt{MEE}_{\mathtt{DNF}}$ is Σ_2^p-hard to approximate to within an n^ϵ factor [17]. Hence,

we will not include the proof of Theorem 4.1 here. However, our proof can be found in [10].

Acknowledgments. We would like to thank Harald Hempel, Jörg Rothe, and especially Lane Hemaspaandra for helpful discussions and comments. We also thank Lane Hemaspaandra for suggesting this topic. We gratefully acknowledge that the paper has benefited from suggestions of two anonymous referees.

REFERENCES

[1] M. Agrawal and T. Thierauf, *The formula isomorphism problem*, SIAM J. Comput., 30 (2000), pp. 990–1009.

[2] B. Borchert, D. Ranjan, and F. Stephan, *On the computational complexity of some classical equivalence relations on Boolean functions*, Theory Comput. Syst., 31 (1998), pp. 679–693.

[3] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.

[4] J. Gimpel, *A method of producing a Boolean function having an arbitrarily prescribed prime implicant table*, IEEE Trans. Comput., 14 (1965), pp. 485–488.

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

[6] P. Hammer and A. Kogan, *Optimal compression of propositional Horn knowledge bases: Complexity and approximation*, Artificial Intelligence, 64 (1993), pp. 131–145.

[7] L. Hemachandra, *The strong exponential hierarchy collapses*, J. Comput. System Sci., 39 (1989), pp. 299–322.

[8] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe, *Exact analysis of Dodgson elections: Lewis Carroll's* 1876 *voting system is complete for parallel access to NP*, J. ACM, 44 (1997), pp. 806–825.

[9] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe, *Raising NP lower bounds to parallel NP lower bounds*, SIGACT News, 28 (1997), pp. 2–13.

[10] E. Hemaspaandra and G. Wechsung, *The minimization problem for Boolean formulas*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 575–584.

[11] M. Krentel, *The complexity of optimization problems*, J. Comput. System Sci., 36 (1988), pp. 490–509.

[12] A. Meyer and L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 125–129.

[13] C. Papadimitriou and M. Yannakakis, *The complexity of facets (and some facets of complexity)*, J. Comput. System Sci., 28 (1984), pp. 244–359.

[14] C. Papadimitriou and S. Zachos, *Two remarks on the power of counting*, in Proceedings of the 6th GI Conference on Theoretical Computer Science, Lecture Notes in Comput. Sci. 145, Springer-Verlag, Berlin, 1983, pp. 269–276.

[15] L. Stockmeyer, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.

[16] C. Umans, *The minimum equivalent DNF problem and shortest implicants*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA, 1998, pp. 556–563.

[17] C. Umans, *Hardness of approximating* Σ_2^p *minimization problems*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 465–474.

[18] K. Wagner, *More complicated questions about maxima and minima, and some closures of* NP, Theoret. Comput. Sci., 51 (1987), pp. 53–80.

[19] K. W. Wagner, *Bounded query classes*, SIAM J. Comput., 19 (1990), pp. 833–846.